gd

GAME DEVELOPER MAGAZINE

APRIL 1999

UBM Tech

Rock Star Woes

You have to love the month of February. And May. Heck, while I'm at it, let's throw in July and November. Why? Those are the sweeps months for television, when the networks throw everything they've got at you in an effort to boost their audiences. For instance, last week I caught "The World's Most Shocking Moments 2: Caught On Tape" on Fox, which included the usual assortment of plane wrecks, animal attacks, extreme sports accidents, monk riots, and other scenes of gratuitous carnage and mayhem. I often wonder why one person can be so fascinated with another's calamity. Whatever the reason, though, sweeps month certainly has its share of these types of shows.

For some companies in our industry, sweeps month came a month early. In January, all eyes were riveted on Ion Storm after a damning exposé about the company appeared in the Dallas Observer, in which reporter Christine Biederman ripped the company's management and cast doubt on the viability of the company. And, like the millions of spectators that tune in to watch tragedies unfold on Fox specials, many in our industry dropped what we were doing to rubberneck at the Ion Storm fender bender. In the interest of conserving this very limited space and owing to my strong aversion to ugly legal messes like that which ensued between various parties in the incident, I don't intend to recount the story. (Plus, frankly, I'm really tired of the whole thing.) But the situation is worth noting for other reasons. For as much as you may not want to witness (or personally experience) another public flogging of a game development company, I can almost guarantee that this type of situation will repeat itself again in the not-so-distant future.

The reason is that today's game companies rely on sophisticated, multi-tiered marketing plans to build up excitement about their products. Once upon a time, game publishers focused most of their time and energy marketing games, and the industry press was the only outlet for this information. However, games are now mainstream entertainment in America, and publishers now cozy up to newspapers and pop-culture magazines for coverage about their games, and compare their game revenues with movies over similar periods of time — a metric readily understandable by the lay person.

Add to that the fact that many companies have started to market the personalities behind the games, elevating designers, programmers, and artists to rock-star status. Just look at Take-Two Interactive, which just formed a new internal videogame label called (perhaps not coincidentally) Rockstar Games. Take-Two says this label "embodies an elite team of talent which has been assembled to redefine the videogame industry.... At Rockstar, we want to change the way the media looks at the game industry and give it some personality, which has been noticeable by its absence until now." Clearly, statements like this are designed to evoke interest from a broader cross section of media, those who know what a "rock star" is, but who don't necessarily understand what an "engine programmer" does during the day.

The combination of marketing personalities and products to mass-media outlets is a shrewd way of getting attention. The mass media readily gloms on to larger-than-life characters. On the other hand, it's that tendency to glom that results in coverage when it's not welcomed. What happened to Ion Storm could have happened to any number of companies, under a variety of different circumstances.

It's clear that many in this industry want the kind of coverage normally afforded true rock stars. However, those individuals ought to keep in mind that rock stars get their share of positive as well as negative publicity, much as they try to avoid the latter. The moral of the story is that companies which charge headlong into the mass-media spotlight must prepare themselves for what could follow — both welcome and unwelcome attention. Those who seek the white hot spotlight shouldn't be too surprised if they get burned by it now and again. ■

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

600 Harrison Street, San Francisco, CA 94107
t: 415.905.2200 f: 415.905.2228 w: www.gdmag.com

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

EDITORIAL

Editorial Director
Alex Dunne *adunne@sirius.com*
Managing Editor
Tor D. Berg *tdberg@sirius.com*
Departments Editor
Wesley Hall *whall@sirius.com*
Editorial Assistant
Jennifer Olsen *jolsen@mfi.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 e: *alanger@mfi.com* t: 415.905.2156
Eastern Regional Sales Manager/Recruitment
Ayrien Houchin e: *ahouchin@mfi.com* t: 415.905.2788

ADVERTISING PRODUCTION

Senior 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/Circulation 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

CEO/Miller Freeman Global Tony Tillin
Chairman/Miller Freeman Inc. Marshall W. Freeman
President Donald A. Pazour
Executive Vice President/CFO Warren "Andy" Ambrose
Executive Vice Presidents H. Ted Bahr, Darrell Denny, Galen A. Poss, Regina Starr Ridley
Senior Vice Presidents Annie Feldman, Howard I. Hauben, Wini D. Ragus, John Pearson, Andrew A. Mickus
Vice President/Development Solutions Group KoAnn Vikören
Executive Vice President/Division SF1 Regina Ridley

BPA International Membership Applied for March 1998

www.gdmag.com

St telli g i RPGs: T Be N t t Be

Warren Spector covered a lot of important topics in his article ("Remodeling RPGs for the New Millennium," February 1999). I'd like to take a few of them a step further, if I may. I agree wholeheartedly that games and game development have matured enough that it's time to start exploring what we do that other media don't or can't do. The issue now is how exactly we can accomplish that.

Chris Hecker proposed the creation of a common terminology in *Game Developer* a few issues back ("Spielberg Switches to Panaflex Cameras," Game Plan, January 1999). Perhaps we need to take his idea and expand upon it. How about coming up with a lexicon of stock devices for computers as a storytelling medium? We could then treat computers not as emulators of board games, movies, or books, but as a true and unique genre.

We would list the strengths and weaknesses of the various computer techniques as applicable to storytelling. This would be accomplished in much the same way Spector did in his article for the few techniques he covered, but in a more formal manner. The true objective is not to create the lexicon, but to stimulate discussion and explore its limits.

It's clear we need to tell different kinds of stories altogether, instead of stories that fail to fit our medium. We need to stop telling movie stories, or book stories, and create an entirely new type of story that emphasizes our strengths. I think Spector's approach to SYSTEM SHOCK was absolutely correct. If computers can't converse, it makes sense not tell stories that require conversation.

It would also help differentiate the true storytelling applications of computers (I hesitate to call them games) from the merely entertaining computer games. Nobody confuses comic books with dramatic literature, even though they're both printed on paper. I think it's time we separated the comic book computer games from dramatic stories told on a computer.

Here's an example. Take Shakespeare's *Hamlet* and apply it to the strengths of the computer. Instead of playing the prince (a theatrical convention) or watching him emote (a movie convention), let's play the ghost. The ghost can influence the action in the story, observe while remaining unobservable, but doesn't carry on conversations or engage in sword fights.

This kind of story caters to the computer's strengths and makes use of its expressive capabilities while avoiding its weaknesses. It is, however, a challenging kind of story to write. Perhaps the problem with RPGs today is not that the computer has its limitations, but that most designers (present company excepted) just aren't very good with the medium.

Mike Kelleghan
via e-mail

Defi i g a Game V cab la

The game design vocabulary Chris Hecker was looking for in his Game Plan column ("Spielberg Switches to Panaflex Cameras," January 1999) is possibly best defined in terms of examples, archetypes, or patterns. Since Erich Gamma and his co-authors wrote *Design Patterns* (Addison-Wesley, 1994), the very word "pattern" has become fashionable, but that does not diminish the idea's value. However, a workable definition of game genres has to precede attempts to list possibly genre-specific recurring patterns.

Many such patterns will be found in existing proprietary game design documents, few of which, if any, are publicly available. Design documents of classic games might be an excellent starting point for any attempt to create a useful design reference vocabulary. Patterns have probably been used for decades, but they need to be identified before they can be used as a language to communicate inside and outside your present team and project.

Games such as HALF-LIFE that try to evoke patterns from movies imply that game designers might do well in complying with the rules of scriptwriting as well — rules such as, "two dogs and one bone." Do not change objectives midway from "save your butt" to "save the world." Do not leave a location once established — stick to one place and one sequence of events. Have one main antagonistic force standing between the player and the objective. Make sure everything shown has meaning and importance. Review HALF-LIFE from this perspective, and you might find that in blatantly violating these rules following the Lambda Core levels, it takes a significant risk of diminishing its otherwise outstanding accomplishments.

I think a game design language is needed to discuss the potential major split in game design implied by the interactive movie approach to gaming. Scriptwriting is an exercise in control and manipulation — the quintessential linear flow. Movies have no choice here. In terms of control theory, a movie is an open loop scenario. Games, on the other hand, depend on the player being in the loop, and decisively so. Ultimately, the patterns experienced in games might just superficially resemble those merely observed in movies.

In my opinion, game design has probably just as much to learn from physics here: setting the scene in terms of initial conditions and equations of motion might ultimately turn out to be superior to scripting events. In interactive entertainment, you may just have to get used to the idea of letting things happen.

There are some interesting resources on the Web for patterns — object-oriented programming and its origin in architecture:
http://www.hillside.net/patterns,
http://st-www.cs.uiuc.edu/users/patterns/DPBook/DPBook.html,
http://www.math.utsa.edu/sphere/salingar/Chris.text.html,
http://public.logica.com/~stepneys/bib/nf/alexandr.htm#pattern.

Bernd Kreimeier
via e-mail

CORRECTIONS

In Dave Pottinger's "Implementing Coordinated Unit Movement" article in the February 1999 issue, several of the figures on pages 56 and 58 were mislabeled. Here's how it should read:

- **Figure 13 in the text should have referred to Figure 13A, Figure 14 should have referred to Figure 13B, and Figure 15 should have referred to Figure 13C.**
- **Figure 16 in the text should have referred to Figure 14A, Figure 17 should have referred to Figure 14B, Figure 18 should have referred to Figure 14C, and Figure 19 should have referred to Figure 14D.**

***Game Developer* apologizes for any confusion.**

BIT Blasts

News from the World of Game Development

New Products

by Wesley Hall

Right Speed, Right Price in 3D Engine

CANOPUS CORP. has just let loose the Rexf/x, a new 3D Digital Video Editing (DVE) system.

The Rexf/x is a hardware-based 3D effects engine and transition package for digital video editors. It combines a hardware 3D engine and 16MB of on-board memory to provide near-real-time rendering speeds. The package includes 40 preset 3D effects, including cube tube, mosaic, double doors, spinning cube, and environment effects such as haze and lighting. Adjustable parameters allow you to create custom effects, too. The engine is accompanied by two software codecs: DV and MJPEG. These codecs allow you to install Rexf/x as a remote editing adapter that can read files generated by other hardware video capture devices. This is a useful feature in a network environment where you want many people working on the same footage, but don't want to buy multiple hardware codec products. Rexf/x also ships with the Standard Effects Pack, Additional Packs can be downloaded from the Canopus web site as they become available.

Rexf/x is available in AGP and PCI configurations. System requirements include a Pentium II, 64MB, an available PCI or AGP slot, and Windows 95 or 98. Rexf/x supports Adobe Premiere 5.1, Ulead MediaStudio 5.x, Canopus RexEdit 2.0 software, and all popular Motion-JPEG and Digital Video (DV) nonlinear editing systems. It retails for $699.

- **Canopus Corp.**
  San Jose, Calif.
  (408) 954-4500
  http://www.canopuscorp.com

New CE miniDraw Library

INTELLIGRAPHICS INC. just announced the CE miniDraw Library, a Windows CE-based subsystem that provides support for portions of Microsoft's DirectDraw for Windows CE platform.

The CE miniDraw Library includes both an API and a hardware abstraction layer (HAL), and enables popular DirectDraw functionality that is currently unsupported by Microsoft on the Windows CE Operating System.

The CE miniDraw Library makes it feasible for you to port DirectX titles to Windows CE. Prior to development of this graphics subsystem, application manufacturers for Windows CE systems were required either to use GDI calls or develop their own hardware interface in order to enable high speed display and animation on the Windows CE platform. Intelligraphics included a number of DirectDraw function calls in the Library's HAL and API, and support for additional DirectDraw functions can be added upon request.

The CE miniDraw Library supports Windows CE version 2 and above. The Library, in the form of a DLL, is one of two components necessary for a DirectDraw solution on Windows CE. Manufacturers will also need a DirectDraw-compatible graphics driver for DirectDraw support.

- **Intelligraphics Inc.**
  Richardson, Texas
  (972) 479-1770
  http://www.intelligraphics.com

Codewarrior for Playstation

METROWERKS recently showed-off the shiny new features of Codewarrior for Playstation Release 5 as the company announced the product's release at the Playstation Developers Conference in San Francisco.

Release 5 will ship sometime before the end of 1999's first quarter. Codewarrior includes the updated IDE, a new MIPS compiler and linker, a Playstation remote debugger, as well as conversion tools and drivers that are specifically designed for Playstation software title development. New features for Codewarrior for Playstation Release 5 include: support for Windows 98: support for command-line development; the latest version of the Codewarrior IDE; an enhanced compiler with improved optimization of GTE macros; support for the DTL-H1500 network attachment box; support for PSNetD, which allows multiple PCs to remotely use the same Sony Playstation development card; a VRAM viewer application, which allows you to browse the current contents of VRAM while running a program in PSComUtil; MWDBcontrol library; and Symbol completion function, which allows the user to input a partial symbol name and have the IDE complete the name for the symbol.

CodeWarrior for Playstation runs on Windows 95/98/NT and retails for $3,000, which includes one free update and free technical support for a year.

- **Metrowerks Inc.**
  Austin, Texas
  (800) 377-5416
  http://www.metrowerks.com

Editor Window for Codewarrior for the Playstation.

Industry Watch

by Alex Dunne

THE UPPER RANKS OF EIDOS underwent changes recently. Rob Dyer, who once held the position of president at Crystal Dynamics, replaces Keith Boesky as the president of Eidos Interactive, and now oversees operations of the company's U.S. office. (Boesky left the company to become an independent consultant.) Joining Dyer at the helm is former Crystal Dynamics coworker Scott Steinberg, who was the vice president of marketing at CD and now assumes the position of senior vicepresident of marketing for Eidos Interactive.

GT INTERACTIVE SUES MIDWAY. GT Interactive, which has signed a number of distribution agreements with game developers over the years, filed a civil complaint against Midway. GT, which has been the exclusive worldwide distributor of Midway's titles since the two companies entered into a contract in 1994, claims that Midway broke this contract, acted in bad faith, and slandered GT Interactive. GT seems to think that Midway tried to hinder the distribution of its own titles so that Midway could minimize its costs, prevent GT from hitting performance goals for the distribution of Midway's games, and then have legal justification to back out of the contract. How could a company harm the distribution of its own titles? GT claims that the defendant did so through "repeated failure to give timely notice of new products; provide game development schedules; deliver usable, bug-free master game disks; provide artwork, advertising materials, and packaging materials; and obtain required third-party approvals." The suit also alleges that Midway's chairman and CEO, Neil Nicastro, "issued false and baseless public statements designed to undermine the confidence of the investment community and the game software industry as a whole in the ability of GT Interactive to effectively market and distribute games and to retain the Midway line." In a response statement, Midway said it would strongly defend itself against GT Interactive's claims, and hinted that it would file counterclaims against GT Interactive.

3DO'S THIRD QUARTER. 3DO issued results for the third fiscal quarter ending December 31, 1998, announcing that its software revenues were $10.3 million, up 178 percent from software revenues of $3.7 million in the third fiscal quarter of 1998. During the quarter, the company shipped five new games, including its first title for the N64, BattleTanx. While the revenues were up, the bottom line wasn't so rosy: 3DO lost $5.7 million during the same period, and has cash and cash equivalents of just $12.1 million. The company is hoping that upcoming titles such as Requiem: Avenging Angel, Heroes of Might and Magic III, Might and Magic VII: For Blood and Honor, and Army Men II hit the top 10 charts and provide a much needed cash infusion.

3DO's BattleTanx for the N64.

CONNECTIX WEATHERS LEGAL STORM. Connectix Corporation's Virtual Game Station, which uses emulation technology to let Macintosh owners play Playstation games, has taken some heat from Sony so far, but seems to be weathering the storm. Recently a San Francisco Federal District Court tossed out Sony's request for a temporary restraining order on shipments of the Connectix product. The Connectix Virtual Game Station, which sells for around $49, won "Best of Show" at the recent MacWorld Expo in San Francisco, Calif.

PATENT WATCH. Rtime Inc. received a patent for its Rtime Interactive Networking Engine, a client/server networking engine that lets real-time multi-user, interactive applications (primarily games) deal with the performance and scalability problems inherent with large-scale networks — such as the Internet. The patent (U.S. No. 5,841,980) is comprised of 54 individual claims that cover several facets of distributed, interactive communication systems, including real-time data filtering, in which only relevant data is distributed to each interested party based on object location, type, priority, or data rate. Additionally, Rtime received protection for its global time base that keeps disparate users synchronized within milliseconds to overcome high and varying network latency. The patent covers both single-server systems, which are predominant in small-scale applications supporting 8 to 32 concurrent users, as well as multiserver systems that support hundreds or thousands of concurrent users. How this patent affects existing games or game networks, or those under development, is not clear. ■

UPCOMING EVENTS CALENDAR

April 17-22, 1999

NAB99
Las Vegas Convention Center
Las Vegas, Nev.
Cost: starts at $150
http://www.nab.org/conventions/

April 19-22, 1999

COMDEX/Spring '99 and Windows World 99
McCormick Place
Chicago, Ill.
Cost: starts at $100 for Guest Pass only
http://www.comdex.com

May 10-13, 1999

3D Design and Animation Conference
Santa Clara Convention Cntr.
Santa Clara, Calif.
Cost: Early Bird rates available
http://www.3dshow.com

Raindrop's Decimator

by Josh White

Here's the punchline: buy Decimator if you need to reduce the polygon counts of .3DS or VRML files and don't need to preserve textures or hierarchy. Decimator has a great algorithm — it collapses edges without adding or removing holes and it's easy and obvious to use. However, version 1 doesn't support UV mapping coordinates, hierarchies, animation data, or any other data in your 3D model.

SETTING UP. Kudos to the online distribution. I downloaded the Decimator demo off of Raindrop Geomagic's web site and had it running on my workstation in about fifteen minutes. I can't imagine an easier way to buy art tool software than downloading it from a web site, and this software really proves that.

Decimator doesn't need much computing power. This initial release of Decimator is a stand-alone program for Windows 95/NT and SGI. The minimum system requirements are 16-bit color, a 100MHz Pentium, and 32MB RAM. It ran plenty fast on my tame $2,000 workstation (a 300MHz Pentium II with 16MB VRAM and 128MB RAM running Windows 95).

DOCUMENTATION. Decimator's documentation is simple: it's on the company's web page (take a look for yourself at http://www.geomagic.com/decimator1_0_guide/chapters/decimator_frame.html). As with most software, the manuals aren't much. I wanted more detail, such as technical explanations, hints, and advice from people who have used Decimator in production. (Does it move vertices? What does it do with coincident vertices? Will it preserve vertex order? What happens to UV coordinates?) Also, more real-world examples or case studies would help illustrate the various purposes of the tools that Decimator offers. Even after reading the documents, the purpose of functions such as Relax and Smooth wasn't clear.

As I read the documentation, I also wondered, "Why does the Decimator manual refer to Wrap so much?" It was confusing. Eventually, I learned that Wrap is Raindrop Geomagic's first product; it generates a smooth surface from "point-clouds" (piles of unconnected vertices).

BASIC USAGE. Here's a real-world example of how you might use Decimator. I built an example model of a warrior princess (Figure 1) in about 20 seconds. (Amazingly enough, I'm giving this fantastic model away free to anyone who writes. No, really.) So let's say we're totally thrilled with our 5,000 polygon character, but those darned programmers are whining about how they said 500 polygons, not 5,000.

Decimating a model is delightfully easy. We simply click the Decimate button and type in the face count we want. Presto! We now have an instant low-polygon version of our model (Figure 2). Decimator did a good job because the model doesn't look drastically different. It mainly killed all the tiny faces in the spheroid body and shin objects, though it did erase the chamfered corners on the boxes. If those chamfers are important, you can easily tell Decimator to preserve them by selecting a group of faces (everything but the boxes in our example) to modify. Unfortunately, this first version of Decimator doesn't import Max's hierarchy, mapping, materials, or animation, and it only reads .3DS files directly.

There's another handy feature: the History bar has thumbnails of each entry in the undo stack. This is convenient when you're experimenting with face count reduction (as you surely will). You can quickly go to any of these by selecting any thumbnail and clicking Go at the bottom.

DATA LOSS. The manual says, "Decimator currently works with raw triangle data only. All other information in input files, including color and texture, is currently ignored." To me, "raw triangles" means (x,y,z) points and the triangles that connect them, so you can't keep material assignments, object transforms, or anything like that when you load a 3D model into Decimator. This restriction isn't a problem as long as you reduce the face count early in your production cycle (as in, before applying any materials, mapping, animation, or other attributes).

REAL-TIME VIEW CONTROLS. The real-time view works reasonably well. When you rotate a model in Decimator's preview window, you'll start to appreciate the robustness of your modeling software's real-time tumble feature. While

FIGURE 1. *The fantastic 5,000-polygon warrior princess.*

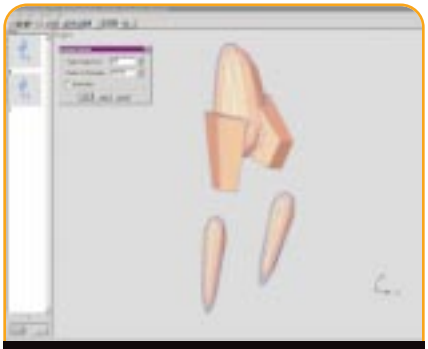


FIGURE 2. *Reduction from 5,000 to 500 — easy case.*

Josh White runs Vector Graphics, a real-time 3D art production company. He wrote Designing 3D Graphics *(Wiley Computer Publishing, 1996), has spoken at the CGDC, and cofounded the CGA, an open association of computer game artists. You can reach him at josh@vectorg.com.*

Decimator's preview works well enough to see what's going on, it's not as polished as a major modeling package's most visible feature. The overlayed wireframe lines penetrate the shaded surfaces in alarming and obscure ways.

If you zoom in on a detail of your model, you'll find that it gets sliced in a surprising way. You've just been introduced to the "near clip plane." Any geometry that gets too close to the camera is clipped or cut off. Graphics tools typically don't have this problem in their real-time preview windows, but Decimator does.

CORE ALGORITHM. I tried a number of different objects and found that I approved of the triangles it threw away. As with most polygon reduction algorithms, Decimator works best when your target polygon count is around several hundred triangles. I experimented with their example file, MONSTER.OBJ, and found that it reduced from 10,000 to 5,000 triangles nicely. Further reduction from 5,000 to 500 was not bad either. After that, I wanted to make the decisions myself — the algorithm started throwing away important faces. Decimator wasn't great at handling thin, two-sided surfaces such as fins. The results were somewhat convoluted, but far better than most typical polygon reduction tools that I've used.

FEATURES I WANTED. I was slightly frustrated by the lack of control that Decimator offered. If I need a specialized polygon reduction tool, then I probably don't have a simple problem to solve, and polygon reduction requires weighing several factors. Instead, we get a single slider for face count. I want the kinds of features that can only be produced by fast, flexible programmers who observe their customers in action, noting their needs and writing tools that fit those needs.

One of the most annoying problems that artists face is coincident vertices from detached faces. My models often must have two overlapping vertices in the same object, but I can't weld them. Perhaps I need to apply separate mapping coordinates on each vertex. In that case, I need them to act as a single vertex when I edit the geometry. Most modeling software, Decimator included, have no awareness of this issue. Their algorithms depend on finding continuous surfaces, and these detached faces resemble holes.

FEATURES I GOT INSTEAD. Relax and Refine are cute features, but basically unrelated to decimation. So far, I haven't needed either of these functions in my career, so I'm not thrilled to find them instead of the features demanded by game artists.

Relax smooths a surface as it reduces face count. It seems to work much like the Spherify modifiers in some 3D modeling software. Relax's main purpose is to remove unintentional variations on insanely detailed (digitized) surfaces.

I was somewhat surprised to find the Refine feature in Decimator, because it adds triangles. The new triangles lie on a surface that the existing triangles define, so the resulting new surface seems smooth and curvaceous. This feature isn't necessary in a polygon reduction tool.

Decimator does have a feature that I appreciate very much: selective reduction. I need the ability to select a few triangles out of a mesh and work on only those (Figure 3).

Decimator's Undo is robust and reliable. This feature is essential to production artists, and often is missing from small tools such as Decimator.

IMPORTING ISSUES. As I mentioned earlier, Decimator doesn't offer very thorough import and export options. This deficiency is one of the most common drawbacks of stand-alone art tools, and it's a difficult problem for any tool maker to overcome. Three-dimensional graphics file formats are constantly evolving, and every market (games, movies, simulation, legal, architectural, mechanical, and so on) has different favorites and different requirements. Plug-ins have the advantage of using their host's data directly, thus raising no import/export issues. On the downside, plug-ins have to be nearly rewritten if developers want to offer their tools for all software packages.

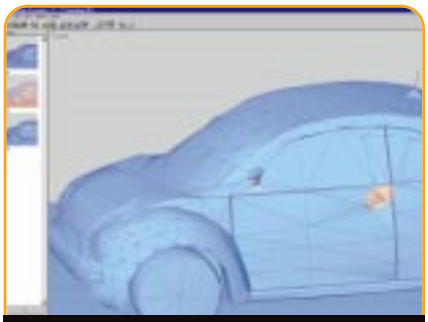


FIGURE 3. *Decimator mercifully allows the artist to reduce triangle count in a particular area, as seen in the rear-view mirror here.*

Decimator's developers assure me that version 2 will feature much more complete import/export capabilities. Raindrop is also working on a plug-in version for 3D Studio Max. These additions should make Decimator much easier to use during production.

HIGH HOPES. Despite its drawbacks, I like Decimator because its core algorithm is well designed and it's easy to use. Version 1 has obvious shortcomings (no texture support, for example) that would prevent game artists from using it interactively during production, but Decimator's developers are currently working on a second version, and I have high hopes for it. Raindrop Geomagic's CTO, Ping Fu, listed the following planned features: preserving texture UV coordinates, keeping hierarchy, ability to select decimation by tolerance, by frame rate, or by distance, real-time diagnosis of decimated surface accuracy by color map, plug-ins for 3D Studio Max, Maya, Rhino, and better handling with very large data (multi-million polygons). With these improvements, I think Decimator would be a solid choice for polygon reduction. ■

Decimator 1 ★★★½

Company: Raindrop Geomagic Inc.
Champaign, Ill.
http://www.geomagic.com
(800) 251-5551

Price: $295

System requirements: A 100MHz Pentium, 32MB RAM, and 16-bit color

Pros:
1. Solid algorithm
2. Single surface maintenance
3. Easy to install and use

Cons:
1. No support for materials, mapping
2. Limited "tweaking" options for decimation
3. Weak import/export options

Lone Game Developer Battles Physics Simulator

As a real-time 3D graphics developer, I need to wage many battles. I fight with artists over polygon counts, with graphics card manufacturers over incomplete or incorrect drivers, and with some producers' tendencies to continuously expand feature lists. However, some of the greatest battles I have fought have been with myself. I fight to bring back the knowledge I have long since forgotten. I fight my desire to play the latest action game when more pressing needs are at hand (deadlines, the semblance of a social life).

This month I document one of the less glamorous battles — the battle of the physics simulator. It's not going to be fun. It's going to be a bit bloody. However, if I ever hope to achieve a realistic and interesting physics simulation, it's a battle that must be fought. So, my brave warriors, join me. Sharpen your pencils, stock your first-aid kit with plenty of aspirin, drag out the calculus book, and fire up the coffeepot. Let's get started.

I hope you all had a chance to play around with the soft body dynamics simulator from last month. The demo highlighted an interesting problem — the need for stability. While creating my dynamics simulation, I waged a constant battle for stability. However, in order to wage the war effectively, I need to understand the roots of the instability in the system. Last month, I implied that the problem resulted from my use of a simple Euler integrator. But I didn't really explain why that caused the problem. Let me fix that right now.

Integrators and You

Many game programmers never realize that when they create the physics model for their game, they are using differential equations. One of my first programs on the Apple II was a spaceship flying around the screen. My "physics" loop looked like this:

```
ShipPosition = ShipPosition + ShipVelocity;
ShipVelocity = ShipVelocity +
               ShipAcceleration;
```

Look familiar to anyone? It's a pretty simple physics model, but it turns out that even here I was integrating. If you look at the Euler integrator from last month, I had

```
Position = Position + (DeltaTime * Velocity);
Velocity = Velocity + (DeltaTime * Force *
                       OneOverMass);
```

Now for my simple physics model, `DeltaTime = 1` and `Mass = 1`. Guess what? I was integrating with Euler's method and didn't even know it. If I had made this Apple II physics model any more complex, this integrator could have blown up on me. These sorts of problems can be difficult to track down, so it's important to understand the causes.

When Things Go Wrong

The reason that the Euler integrator can blow up is that it's an approximation. I'm trying to solve a differential equation by using an iterative numerical method. The approximation can differ from the true value and cause error. When this error gets too large, the simulation can fail. A concrete example may help to explain. Last month, I added a viscous drag force to the simulation to add stability. The formula for this force was

$$F_d = -k_d V \qquad \text{(Eq. 1)}$$

In this formula, k_d represents the coefficient of drag that is multiplied by the velocity of the particle. This coefficient determines how fast the velocity of the object is dragged down to zero. This is a very simple differential equation. In fact, it's simple enough to be satisfied for v directly by the formula. I can use this exact solution to check the accuracy of my numerical integrator:

$$V = e^{-k_d t} \qquad \text{(Eq. 2)}$$

Euler's method is used to approximate the integral curve of Equation 2 with a series of line segments along this path. Each step along this path is taken every time, interval h, via the formula

$$w_{i+1} = w_i + hf(t_i + w_i)$$
$$V_{i+1} = V_i + h(-k_d V_i) \qquad \text{(Eq. 3)}$$

In all cases, the viscous drag force should approach zero. However, the size of the step h and coefficient of drag k_d determine how well the approximation performs. Take a look at Figure 1.

With the given step size and drag coefficient, Euler's method may not be a great approximation, but it gives the desired result. The velocity converges on zero. But take a look at the relationship between the step size and drag coefficient in Equation 3.

If $h > \frac{1}{k_d}$ then the approximation step will overshoot zero, as you can see in Figure 2.

Jeff is the technical director of Darwin 3D where he spends time calculating his rate of procrastination with respect to his articles. E-mail optimization suggestions to jeffl@darwin3d.com.

By increasing the step size, I was trying to get a system that converged to zero more quickly — but I got something entirely different. Things really start to get bad when the drag coefficient increases more, as in Figure 3. As each step is taken, not only does the approximation oscillate across zero, but it also actually diverges from zero, and eventually explodes the system. This is exactly what was happening in the spring demonstration from last month, when the box blew up.

How Can I Prevent Explosions?

If you find a situation where your simulator blows up, there's an easy way to see if this kind of numerical instability is the cause. Reduce the step size. If you reduce the size of the step and the simulation works, then this numerical instability is the problem.

The easy solution is always to take small steps. However, realize that each step requires quite a few calculations. The simulation will run faster if it can take fairly large step sizes. Unfortunately, when you get lots of objects interacting, these instability problems appear even more. So, just when things start to get interesting, you need to reduce the step size and slow things down.

I'd rather create an integrator that would allow me to take large step sizes without sacrificing stability. To do this, I need to look at the origins of Euler's method.

FIGURE 1. *A decent approximation.*

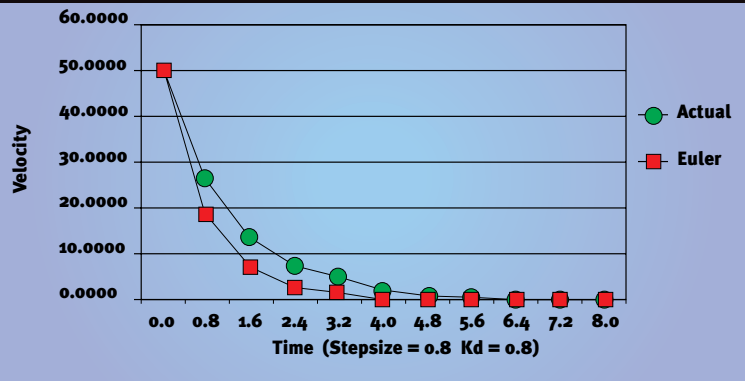


FIGURE 2. *This looks a lot worse.*

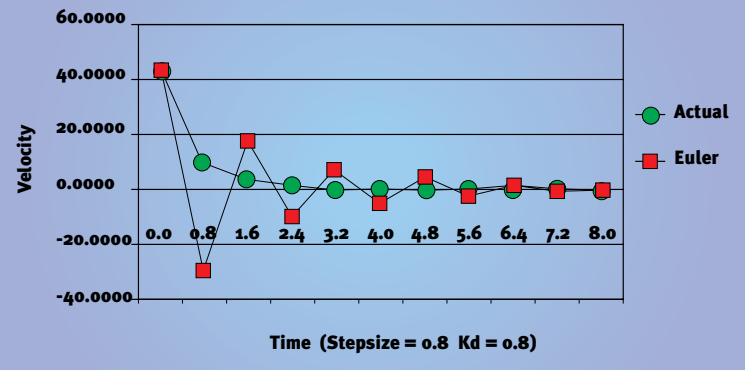
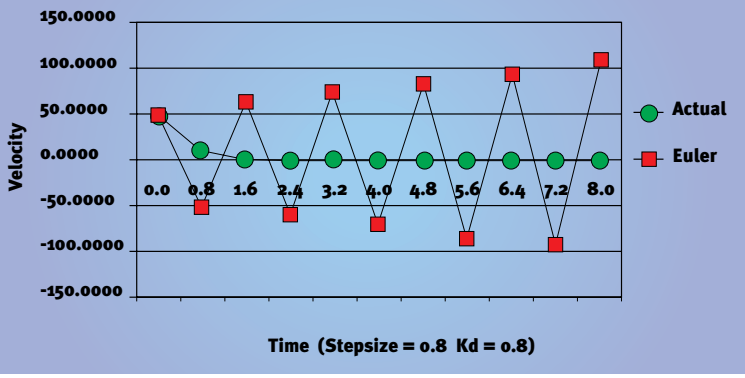


FIGURE 3. *Kaboom!*



Taylor's Theorem

You may remember Taylor's Theorem from calculus. It's named after mathematician Brook Taylor's work in the eighteenth century. This theorem describes a method for converging on the solution to a differential equation.

$$f(x) = P_n(x) + R_n(x)$$

$$P_n(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

$$R_n(x) = \frac{f^{(n+1)}(E)}{n!}(x - x_0)^n$$
$$x_0 < E < x$$

(Eq. 4)

In Equation 4, $P_n(x)$ represents the n^{th} Taylor polynomial. If you take the limit of $P_n(x)$ as $n \to \infty$, you get the Taylor series for the function. If, however, the infinite series is not calculated and the series is actually truncated, $R_n(x)$ represents the error in the system. This error is called the truncation error of approximation.

How does this apply to the problem with which we are working? If I only look at the first Taylor polynomial and do some substitution, I get Equation 5.

$$h = (x - x_0)$$
$$w'(t) = f(t, w(t))$$
$$w(t_{i+1}) = w(t_i) + hf(t_i, w(t_i)) + \frac{h^2}{2}w^n(E)$$

(Eq. 5)

Notice how similar this equation is to Equation 3. In fact, Euler's method is based on this equation. The only difference is that the last error term is

dropped in Equation 5. By stopping the series at the second term, I get a truncation error of 2. This gives Euler's method an error of order $O(h^2)$.

If I added another term of the Taylor series to the equation, I could reduce the error to $O(h^3)$. However, to compute this exactly, I would need to evaluate the next derivative of $f(x)$. To avoid this calculation, I can do another Taylor expansion and approximate this derivative as well. While this approximation increases the error slightly, it preserves the error bounds of the Taylor method. This method of expansion and substitution is known as the Runge-Kutta techniques for solving differential equations. This first expansion beyond Euler's method is known as the Midpoint method or RK2 (Runge-Kutta order 2), and is given in Equation 6. It's called the Midpoint method because it uses the Euler approximation to move to the midpoint of the step, and evaluates the function at that new point. It then steps back and takes the full time step with this midpoint approximation.

$$w_{i+1} = w_i + h\left[f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}f(t_i, w_i)\right)\right] + O(h^3)$$

(Eq. 6)

In fact, I can continue to add Taylor terms to the equation using the Runge-Kutta technique to reduce the error further. Each expansion requires more evaluations per step, so there is a point at which the calculations outweigh the benefit. I don't have the space to get into it here, however, I understand that smaller step sizes are preferred over methods above RK4 with an error of $O(h^5)$ (Faires & Burden, p. 195). Runge-Kutta order 4 is outlined in Equation 7.

$$\begin{aligned} k_1 &= (hf(t_i, w_i) \\ k_2 &= hf\left(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_1\right) \\ k_3 &= hf\left(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_2\right) \\ k_4 &= hf(t_i + h, w_i + k_3) \\ w_{i+1} &= w_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5) \end{aligned}$$

(Eq. 7)

RK4 gives the simulation a very robust integrator. It should be able to handle most situations without blowing up. The only issue now is what the step size should be.

Watch Your Step!

Even with a robust integrator such as RK4, there will be times when the simulation will be in danger of blowing up. To keep this from happening, you may have to reduce the step size at times. At other times, however, a large step size works fine. If my simulator only has a single fixed step size, I cannot take advantage of these facts. If I vary the size of the steps according to need, I could use large steps when possible without sacrificing stability.

This is how it works. I take full step using my current integrator, then take two steps half the current step size, and compare the results. If the error between the two results is greater than a threshold, then the step size should be reduced. Conversely, if the error is less than the threshold, the step size could actually be increased. This form of control is known as an adaptive step size method. Adaptive methods are a major area of research in numerical analysis, and can definitely improve simulation performance. I chose not to implement adaptive step size controls in my simulation. However, this is an area where you could improve the simulation.

Other Techniques

Differential equations are not easy to learn and understand. However, the programmer who pursues this knowledge has many weapons in his arsenal. As witnessed by the birthdates of Euler and Taylor, this research has been going on for centuries. If you ignore this work and strike out on your own, you're doing yourself a great disservice. Knowledge is available to the developer as never before. While working on these algorithms, I was able to cross-check formulas and techniques in many different sources.

In fact, I've barely scratched the surface of the field. The integrators I've described (all explicit one-step methods) represent only a subset of the methods available to the programmer. Implicit integrators will also work. For example, an implicit Runge-Kutta integrator trades greater computations per step for greater stability in particularly difficult differential equations. Also, the one-step nature of these integrators reflects the fact that the method does not consider any trends in the past when calculating a new value.

In addition to these one-step methods, there are also multistep methods, extrapolation algorithms, predictor-corrector methods, and certainly many others. Clearly, there is plenty of ground for the adventurous programmer to explore. The book I used, *Numerical Algorithms with C,* does a good job of comparing different methods during a variety of test conditions.

For this month's sample application (available from *Game Developer's* web site), I have implemented both the midpoint method and Runge-Kutta order 4 in the dynamic simulation from last month. You can switch between integrators and adjust the step size and simulation variables to get a feel for how each performs. ■

FOR FURTHER INFO

In addition to the references cited last month, a couple of other sources proved very valuable during this article.

- Faires, J. Douglas and Richard Burden. *Numerical Methods.* Second edition. Pacific Grove, California: Brooks/Cole, 1998. This book provided a great discussion of measuring error in numerical solutions. It also contains a great deal of source code for all the algorithms.
- Engeln-Müllges, Gisela and Frank Uhlig. *Numerical Algorithms with C.*, New York, New York: Springer-Verlag, 1996. In addition to the fine sections on the methods discussed in this column, this book describes and compares a great number of other numerical methods. Additionally, the book has a great number of references to articles on the topic.
- Press, William H. *et al.*, *Numerical Recipes in C.* Cambridge, England: Cambridge University Press, 1998. While not as strong a reference on these topics, this book may be interesting to many, as it is available in electronic form. See http://www.nr.com but also check out a critical discussion of it on http://math.jpl.nasa.gov/nr/nr.html.

Playing God: Putting it All Together

In January's column, part one of the "Playing God" series, we talked about game-play–critical architecture. In February's installment, we followed up with tips on how to build and populate an immersive environment. This month, we'll finish the series by taking a step back and seeing how it all fits together.

We'll look at art resources from each major part of a project, and discuss tips and tricks for getting these resources produced in a timely manner.

What's it Take to Make a Game?

Figure 1 shows a scene from Surreal's game, DRAKAN. This title has been in production for almost two years, and at one point had upwards of eleven artists and animators working on the resources for the game. Let's analyze the scene in Figure 1 and identify the building blocks necessary to construct it.

Player Characters

Arguably, the most important part of an artist's job is to work on the game's player character(s). The player character defines the personality of the game and gives it its flavor. There's little room for error because the player character is constantly on the screen, whether in the form of a weapon (in the case of first-person perspective), or in the form of the player's avatar (in the case of third person). Getting that character to feel just right to the target audience is one of the toughest challenges that we face as artists.

In a real-time 3D game — and in almost any character-based game for that matter — we are asking the player to identify with the character, whether by seeing through the character's eyes or by vicariously enjoying the character's activities. To that end, everything about the character must be convincing. The character's shape and structure, how the character moves, and what (if any) weapons the character uses all must fit together in a coherent package that the player can at once relate to and recognize.

The simple fact is that to have a good character, you need a good character concept — and you need to execute that concept correctly. Back in my October column, entitled "It's About Character," we looked at some of the aspects of character design for real-time 3D. Now, let's take this a step further and look at some of the particulars that compose a good player character.

In Figure 2, we can see all the pieces and parts that make up the main player character in DRAKAN. Two of the design goals that our team set while creating this character were that she be both

FIGURE 1. *Scene from DRAKAN.*

FIGURE 2. *DRAKAN's player character.*

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.

attractive and athletic. These qualities, among other concerns, dictated the character's body style and shape. After several iterations, the player character's shape evolved into Figure 2's design.

The art direction for the game mandated a stylized-yet-realistic look that involved hand-painted textures. The textures would augment and make up for the low-polygon nature of the character. If you look closely at the texture maps, you can see the subtle shading that give the texture it's 3D feel. This helped to give the character a smoothness and a roundness of form that belied the polygonal nature of the underlying geometry. (The in-game shot in Figure 2 shows the highest level of detail, roughly 500 polygons).

PLAYER-CHARACTER PLANNING. The character in Figure 2 is shown wearing one of the several different costumes in which she appears in the game. For each costume, three levels of detail (LOD) were generated, with polygon counts ranging from 120 polygons at the lowest to 530 polygons at the highest. For any given costume, the same texture set was used for all three LODs. Modeling and texturing, including the character conceptual work, took roughly two weeks for each version. This two week period, plus the time we spent on the over 200 animations that this character uses in the game, brought the total time commitment for the main character close to eight man-months.

PLAYER-CHARACTER TIPS. Details, details — spend as much time and effort as you can putting the subtle details and nuances into the character's textures. If you look at the preceding example, you'll notice that most of the character's shape and form comes from the textures, not from the geometry.

The character's silhouette needs to be convincing as well. The human brain perceives gross shape and color before it recognizes the minutiae of fine detail. Your character should be recognizable without any textures applied. This will help cement the character's personality.

Non-Player Characters (NPCs)

The NPCs in the game are the creatures and people with whom the player's character will interact. The NPCs are the supporting actors and actresses on a stage starring the player as the main character. They give the player something to do (most likely, you'll end up fragging most of these poor folks). And with their personalities, they help to create the mood and set the drama within the player's world. Essentially, the same guidelines apply to the NPCs as apply to the main character. NPCs need to be convincing in form and motion, and they need to fit with the art direction of the existing world.

In Figure 3, we can see one of the main antagonists from DRAKAN, a hefty Wartock. Weighing in at roughly 500 pounds, these brutish beasts tower over the main character at a height of seven to eight feet. The original concept for these creatures called for an intimidating — if somewhat dimwitted — thug. The body structure for the character shows this in its construction. The powerful, heavy-set shoulders, the overly large facial features and brutish tusks all serve to give the character a menacing appearance, yet at the same time emphasize the disproportionately small brain box.

Polygonal construction is more limiting for NPCs because (in most games) there will be situations where many of them are simultaneously on screen. In this instance, the three LODs for the NPC clock in at 300, 180, and 60 polygons, respectively. This is barely over half the resolution used in the main character. The low polygon count of the NPC puts even more pressure on the texture artist to hide the polygonal nature of the model, and, as before, you can see that almost all of the detail in the model is actually done with textures.

NPC PLANNING. This particular NPC came in three variants, which took about one week to conceptualize, model, and texture. Combined with the few dozen animations this character used, this specific class of NPCs took close to five man-weeks to complete.

NPC TIPS. NPCs are the guys you just love to hate. Until videogames become a kinder and gentler experience, the NPC crowd is going to keep getting fragged, slashed, pummeled, and vaporized by millions of game players worldwide. That being the case, one of the best ways to make NPCs interesting is to give them interesting deaths. Most, if not all game players, have a macabre streak running through them, and you don't have to look too deep into videogames to know that developers have recognized and capitalized on this all too common facet of the player psyche.

Game-Play–Critical Structures

Back in January, we looked at the theory and practice for game-play–critical structures. We analyzed how important these pieces were in the overall flow of the game, and how they fit within the project timeline. We also looked at ways in which design and art could interact to ensure that these structures conformed to both the game-play mechanic and the artistic direction.

Figure 4 is an example of a piece of game-play–critical architecture. This happens to be a building in which a

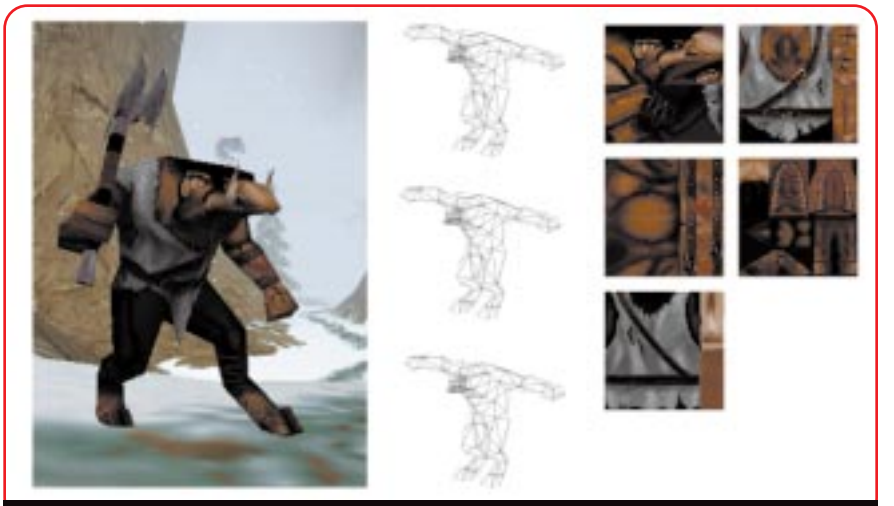


FIGURE 3. *An NPC from DRAKAN.*

FIGURE 4. *Village tavern (game-play–critical structure).*

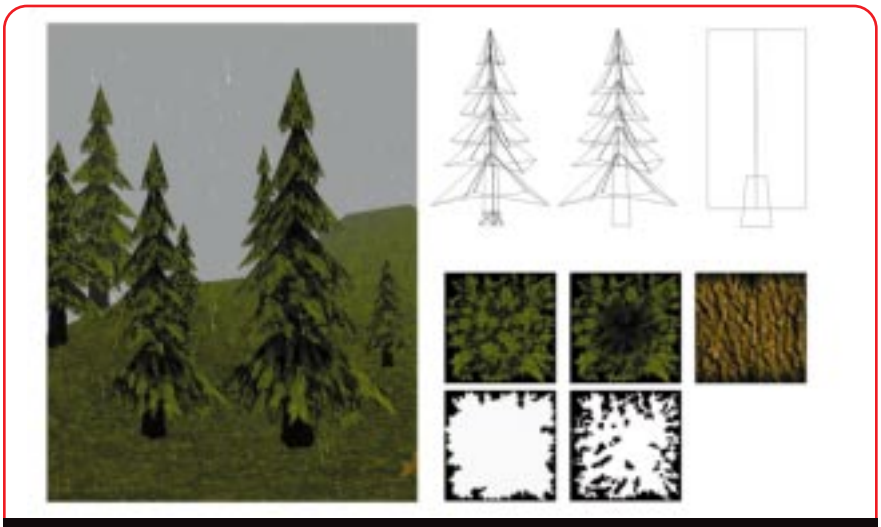


FIGURE 5. *Ambient objects.*

modest amount of exploration and combat takes place. Additionally, this building acts as a central hub for the rest of the surrounding village. The tavern acts as a focal point in the environment, and the player will have a large amount of interaction with this piece of architecture. Therefore, the tavern meets the criteria to be a game-play–critical object.

The initial conceptual criteria for this building were that it contain two stories and that it enclose sufficient space inside for combat to occur. Furthermore, the artistic vision called for the level and style of architecture to be at once unique and yet consistent with the rest of the level design. The standard A-frame construction immediately breaks up the rigid, orthogonal lines of classical human habitation.

As you can see from the cutaway, we built the interior and exterior portions of the building independently of one another. This was done to allow the engine to render only the internal geometry while the player was outside the structure, or vice-versa — this reduced the overall polygon count while easing the restriction on available polygons for internal construction. The bottom line is that the internal geometry could be much more detailed.

STRUCTURE PLANNING. From concept, to model, to textured building, this piece of architecture took approximately five days to complete. Although this particular piece is unique and only occurs once in the game, the work done on generating the textures and geometry overlapped into other buildings of similar shape and style. So, the work done here actually lessened the amount of work done on future buildings of this type.

Ambient Objects and Flora

In February, we discussed how to assemble complete sets in the environment. We talked about the need for filling your worlds with ambient architecture and the appropriate flora. In order to give your worlds the depth and believability that will make them feel immersive, you just can't skimp on these objects. In any given scene, you can probably expect to spend 20 to 60 percent of your polygon count and on-screen fill budget on ambient objects and flora. It's worth spending the polygons, though, because that's about what it takes to make the scene feel whole.

Figure 5 shows an example of a pine tree used in populating a mountainous region. This particular model is a pretty good design because the polygon counts are relatively low (48, 30, and 8 polygons for the three LODs, and yet the objects is still recognizable by its shape alone. Note that for the lowest LOD, a simple planar construction will suffice to give the depth necessary at distance, and will only cost you one-sixth of the polygons required for the highest LOD.

AMBIENT OBJECT PLANNING. This type of object takes about half a day to do, and that includes both modeling and textures. The different sized trees in the world were generated by randomly scaling and rotating the objects. This procedure gave us a high level of variation without the additional memory overhead required for different-sized objects.

AMBIENT OBJECT TIPS. Be fastidious in your level creation so that you have enough room in your polygon budgets for placing ambient objects around a level. When you sit down to put the final touches on a level, it can be extremely frustrating to find out that you simply don't have the overhead to do it. You end up with a geometrically complex level that is nevertheless stale and uninteresting.

These models for ambient objects are relatively easy to build, but don't be wasteful. The models need to built efficiently and with care because they will probably be appearing in groups. For example, in Figure 5 above, the real-time view shows a scene with seven trees. That means that for every five extra polygons you add to the tree, you're going to be paying for 35 extra

polygons in your scene. Remember to do your polygon math in multiples for ambient objects, just as you would for an NPC. Once again, it's all about well-designed economy.

Environments

If you think of your game as a kind of interactive play, then the environment is analogous to its stage. This is where everything comes together; the actors read their lines, the sets are built, the lighting and sound set the mood, and all parts cooperate to create the illusion of a complete world. Gone are the days when you could get away with using a plain, boring landscape populated by a few trees and out-of-place buildings. Whether composed of dugouts and pitfalls to shelter the player from enemies, or seamless, rolling vistas over which players will soar, the environments of today's real-time 3D titles are expected to be completely immersive, as well as an integral part of game play.

So what makes a good environment? In real-time 3D, a good environment provides a convincing, immersive area for the game play to take place. The environment must fit together seamlessly with the structures placed upon it and the style of characters moving in it. Whether your art direction is highly stylized or hyper-realistic, the environment plays almost as big a roll in defining the aesthetic as the characters.

A good environment is not necessarily realistic. Miles and miles of rolling hills or rambling corridors do not make for interesting game play. The vast majority of people don't play games to simulate reality. Instead, they play games to be stimulated. That doesn't mean that you can't get away with using real-world scenarios and geometries in your game. Just be sure you never send your players out to navigate the world without giving them something to do or a specific direction to follow. If players get lost or doesn't know where they're supposed to go, they'll get bored and lose interest very rapidly. Maps can help lessen this problem, but maps shouldn't be used as a crutch for a poorly designed world. Game players will see right through this and resent you for it.

To avoid creating a dull world, you should build the environment so that it either funnels players towards the game-play areas, or provides a constantly changing game-play dynamic that will keep players interested even if they stray off the beaten path. Obviously, there are some instances where the game may take place entirely indoors, in which case, the structures become the environment. Pay attention to the same design details when you create an internal environment as you do when you create an external one.

Figure 6 shows an example of a piece of landscape for DRAKAN. This outdoor shot shows a pastoral mountainside with geometry inspired by the rock formations of Utah's Arches National Park. The flowing, organic-style of the geometry is what gives the terrain its natural, realistic feel. The texture set enhances this effect with several dozen hand-painted textures to minimize overlap and repetition. Note the waterfall in the center of the shot.

ENVIRONMENT PLANNING. Building the environments is arguably one of the most time-consuming parts of the real-time 3D process. Lots of trial and error and game-play tweaking, combined with the sheer surface area involved, combine to make this a mammoth task. For example, the landscape section in Figure 6 is part of one contiguous piece, so it's tough to estimate an exact timeline — however, for the entire level (approximately one square mile of in-game space), it took close to two months to get the geometry just right, while the textures were cranked out at a rate of 6 to 10 per day.

ENVIRONMENT TIPS. Polygon count is still a factor, even with today's high-powered processors. In the preceding example, note that the area is actually a canyon. This technique serves to channel the player towards the game-play areas while minimizing the actual viewing distance for the player. This sort of design results in a lower overall polygon count in the environment and allows more creatures and objects on-screen.

When building outdoor environments, it's important to remember that nature seldom employs straight lines or planar surfaces. Also, to complete the organic look, avoid any hard, orthogonal angles, and refrain from using clean texture boundaries; for example, use a transition texture at the boundary where snow meets rock, or sand meets grass, and so on.

Make the Parts Whole

This third and final installment of the "Playing God" series stitches together the concepts that I covered in January and February and attempts to present an overview of designing and creating the art for a world in real-time 3D. Meticulous details will create the illusion of reality in your game, but in order to make the details themselves a reality, you must be able to grasp the big picture and plan effectively. I can only hope that this series aids you in that process. ■

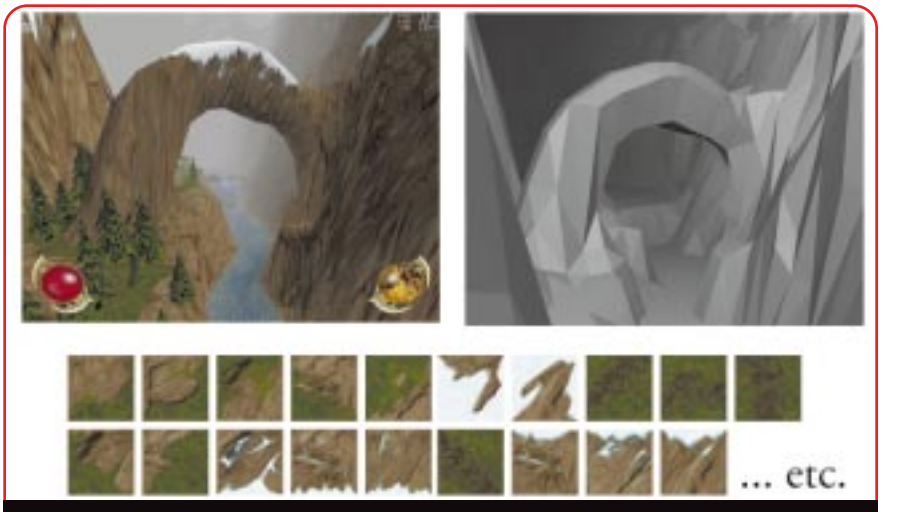


FIGURE 6. *DRAKAN environment inspired by Utah's Arches National Park.*

Nvidia on the Brink

As I write this column, Nvidia's closest competitor (3Dfx) has just purchased Nvidia's biggest customer (STB). With Nvidia fresh from its initial public offering (IPO), the time is right to take a closer look at the company that would be king of 3D.

History

Nvidia began operations in January 1993. The company was one of three high-profile 3D chip start-ups at the time (the other two being Rendition and 3Dfx). Even then, Nvidia's marketing message wasn't pure 3D, but was more about a coherent multimedia platform. The company was backed by a combination of venture capital and corporate technology funding, and had a strong manufacturing partner in ST Microelectronics (formerly known as SGS Thomson). Sega of America helped the company gain credibility when they established an exclusive licensing agreement with Nvidia to convert Sega's Saturn and arcade software to CD-ROMs for PCs equipped with Nvidia's multimedia accelerators. The deal may have come because of Nvidia's multimedia technology, or possibly because the company wanted to repurpose its content for the PC platform. Whatever the reason, the Sega deal helped Nvidia secure the support of Diamond, a company that bought over 80 percent of Nvidia's products in 1995 and 1996.

The relationship with Diamond had mixed results for Nvidia. The Diamond Edge 3D, arguably the first consumer 3D graphics board to hit the PC market, ended up being too expensive and lacking in enough game support to kick start the market. Diamond's CEO used a financial analysts' conference call in 1996 to decry the value of his own company's Edge 3D inventory, and wrote off $5 million in excess inventory charges as a result. Ironically, this inauspicious start to business may prove to be Nvidia's strongest hand in the 3D game. Nvidia was able make all the mistakes of any fledgling 2D/3D chip vendor before there was any real 3D market to witness the young company's errors.

Oddly enough, DirectX also helped Nvidia find its direction and purpose. By the fall of 1996, Nvidia and SGS Thomson had both said publicly that they were codeveloping a new part based on Direct3D that aimed at accelerating all of Direct3D's functions. When Nvidia announced the RIVA 128 (RIVA stands for Real-time Interactive Video Accelerator) in the Spring of 1997, the company was back on track in the graphics business. Unlike Nvidia's earlier attempt at multimedia acceleration, RIVA was pure graphics — and it hit all of the graphics hot points. It had Direct3D acceleration, and pretty good performance at that. It had a high-performance VGA, 2D, and video core.

The legacy of Direct3D acceleration carries the company forward today. Michael Hara, vice-president of corporate marketing for Nvidia, says, "A year ago, developers were using Glide to show off their games and get a good deal. If you look at the number of developers supporting Direct3D vs. Glide today it doesn't make sense to support Glide. So, in 1999 we'll see a lot of developers leveraging Glide and their investment in it, but Direct3D is the overwhelming choice for developers, and that's what we accelerate better than anybody else."

RIVA was, and is, an ideal product for PC OEMs as well. In the chip's first year of production (1997), Dell, Gateway, and Micron had each signed up for the RIVA 128. Dell and Gateway were to be serviced by STB Systems, while Micron went to Diamond Multimedia. Nvidia went from being on the chopping block in 1996 to prepping for an IPO in 1998. The company's most recent success comes from RIVA's big brother, the TNT — a chip that has put Nvidia at the helm of the graphics industry.

The Industry Landscape

Last year wasn't a great time for a graphics chip company to have an IPO. Intel virtually made it impossible for the graphics industry to do anything but fight fires, rumors, and innuendo about its future. By the end of 1998, Intel's Intel740 graphics chip

FIGURE 1. *Nvidia's abbreviated financial statements based on the company's IPO filings (figures in thousands).*

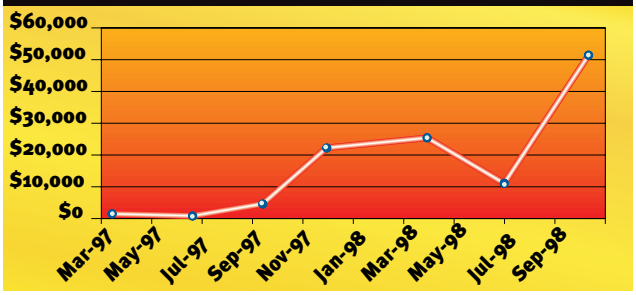
	Year ended December 31, 1995	Year ended December 31, 1996	Year ended December 31, 1997
Product sales	$1,103	$3,710	$27,280
Royalty revenues	79	202	1,791
Total revenue	1,182	3,912	29,071
Gross profit (loss)	(367)	874	7,845
Research and Development	2,426	1,218	6,632
Sales, general and administrative	3,677	2,649	3,773
Net income (loss)	(6,377)	(3,077)	(-2,691)

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.

had fallen flat. In addition, 3Dfx and Nvidia had created a healthy retail market for graphics boards based on the companies' respective Voodoo2 and TNT chipsets. As a result, Creative Labs and Diamond stocked the retail store shelves with competing products based on both 3Dfx's and Nvidia's chips, and STB gainfully serviced the PC OEM market. At the same time, 3Dfx managed to acquire a board business — STB — which was more eager than its competitors at Diamond or Creative to get proprietary silicon (or so the story goes).

Now, in 1999, the landscape for 3D graphics doesn't look any more stable. 3Dfx has the challenge of merging with STB and making that deal work to its advantage. Nvidia has the two largest brand name board makers in the world almost all to itself. 3Dlabs is going to make a run at the consumer 3D market with Permedia 3, and S3 is promising to raise the bar at the low-end as a result of a cross-licensing agreement with Intel. In addition, NEC/Videologic has finally shipped PowerVR to Sega for Dreamcast, and it's ready to try for the PC market again. Intel is going to avoid the consumer 3D sweet spot for now, but has designs on the sub-$1,000 PC market and the high-end workstation business. In addition, mobile graphics vendor NeoMagic is rumored to have its eye on the desktop. The good news for Nvidia is that it's the only company with a proven architecture and roadmap to rival these competitors and to go after ATI's stranglehold on the PC OEM market.

FIGURE 2. *Nvidia's quarterly product revenue (in thousands of dollars).*



Nvidia's Road Ahead

TNT2 will drive Nvidia's sales effort in 1999. The chip launches on the back of the Pentium III, and hits the feature list that pleases the PC OEMs. According to Michael Hara, TNT2 will feature a 32MB frame buffer, significant improvements in the graphics and memory clock speeds, will support AGP 4X, will have integrated transceivers for flat panels, will be optimized for K6 3DNow and the new instructions in Pentium III, and will have a full 32-bit rendering pipeline. It's the strength of Nvidia's TNT that sets the company apart from the crowd in 1999.

Nvidia has many obstacles in its path, too. First, there are two uncertainties: if the company has a successful IPO it comes at a time when the company needs the capital investment, but is also facing its strongest competi-

FIGURE 3. *Year-on-year change in the share of Nvidia's revenues due to STB, Diamond, Creative, and others. Total sales by year are 100 percent.*

	1995	1996	1997	9 Months of 1998
STB			63%	40%
Diamond	86%	82%	31%	28%
Creative				12%
Others	14%	18%	6%	20%

tion ever, from all quarters. The twin pressures of competition and Wall Street are considerations. Secondly, the company has outstanding intellectual property and patent issues to resolve in three lawsuits from Silicon Graphics, S3, and 3Dfx. While the company is confident of its own case in each suit, it still faces a drain on management time and resources just when the competitive arena is at its most active.

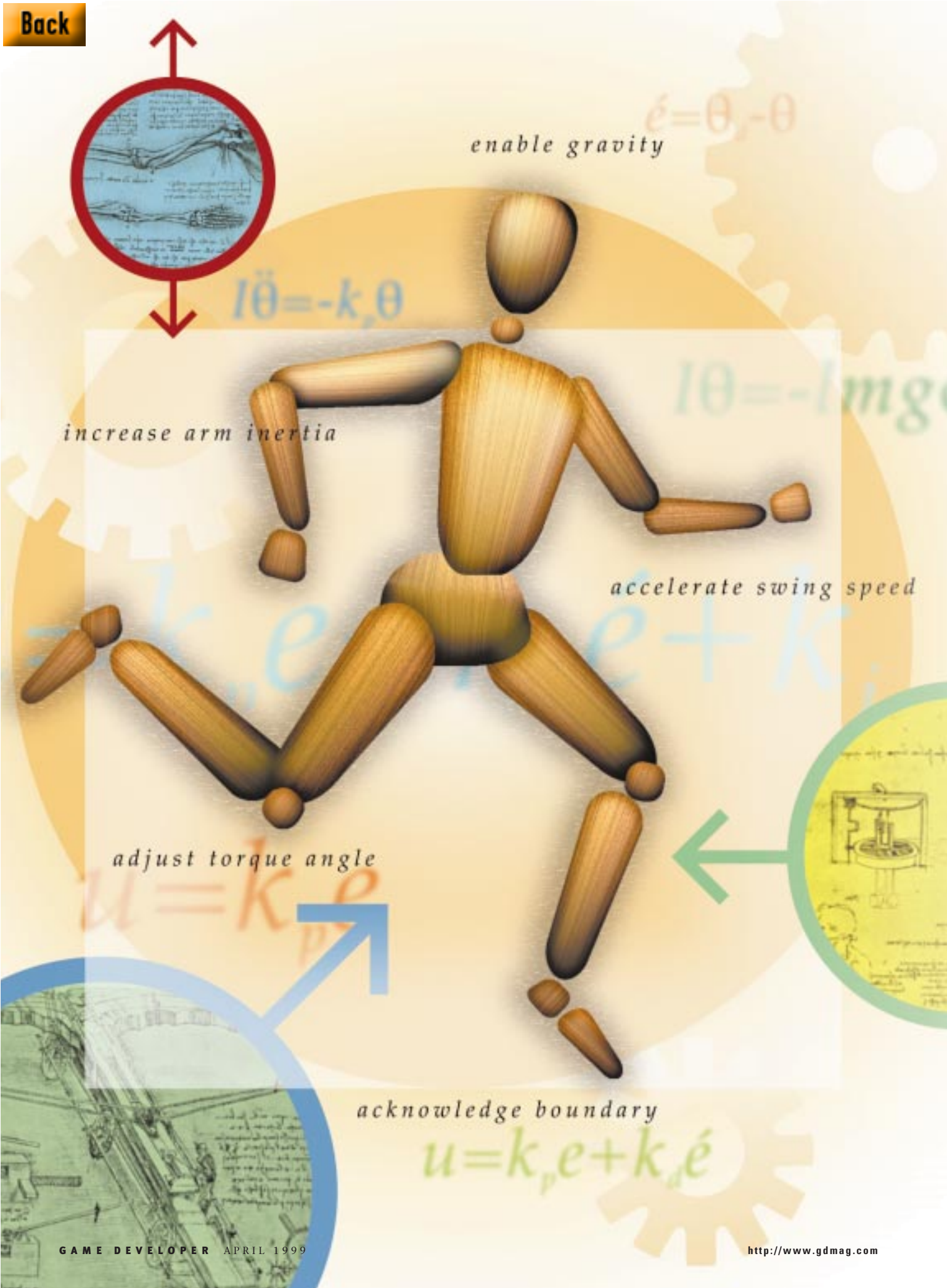
There is a significant backlash against 3Dfx brewing among Creative, Diamond, and other board OEMs left high and dry by the STB deal. That means that Nvidia will probably have as much marketing muscle in its corner as any graphics chip company has ever had. This may push the company's brand awareness over the current champion, 3Dfx. Nvidia's support of both Direct3D and OpenGL, coupled with the strengths of TNT2, put the company at the head of the pack. If Nvidia's chip manufacturing partners can deliver enough product, there is little to stop the company continuing on a high ramp up in 1999. All of this action, especially from a company primed to serve the PC OEM market, will help to proliferate the kind of premium 3D performance that game players value.

Nvidia has to maintain its momentum in the coming year to avoid falling by the wayside like so many other graphics chip companies. But how will it react to industry moves such as the 3Dfx/STB merger? ATI and Matrox have proven the value of chip/board combos. Undoubtedly, 3Dfx/STB will be clubbing that message into the heads of Nvidia's PC OEM customers. At some point, Nvidia will have to do something. Michael Hara, however, doesn't believe Nvidia has to change.

"Vertical integration is not the only answer. The chip has to be good or else the board doesn't matter. The market leader has to be a company that innovates and executes successfully. In some ways, ATI didn't win the market, but maybe S3 lost it. We don't plan on repeating the mistakes of the past that others have made."

There are a few other chip manufacturers (not to mention board vendors) that are relying on Nvidia being correct about the market. These guys are veterans, so when they move, it's worth paying attention.

Shortly after this column was finished, Nvidia completed its IPO. It was relatively successful, but very low key. Nvidia's IPO set the agenda for 3D graphics competition in the months and years to come, but it also signals the end of an era in the PC graphics industry. It may have been the last great graphics chip IPO that we'll see. In the next two years, all graphics companies may look like ATI. Nvidia harkens back to the days when a company such as S3 dominated OEM and add-in board channels with equal ease. ■


Back
enable gravity
increase arm inertia
accelerate swing speed
adjust torque angle
acknowledge boundary

Physical Controlles: Re-inventing Game Movement

by Chris Hecker

I advocate physics as an important game technology because I believe we must increase the interactivity of our environments. If we're to accomplish this leap in interactivity, we need to let the behavior of objects emerge through interactions with the player, not just try to prescript a fixed number of behaviors. By "behavior," I could mean something as simple as a box's motion as it tumbles down a hill and lands in a pile at the bottom. A good dynamics simulator will make the right thing happen regardless of how the player tosses the box, and the movement will be convincing and consistent with the player's expectations. However, I choose to define "behavior" in a much broader sense, encompassing not only passive situations such as the motion of a tumbling box, but also active situations such as machines with moving parts reacting as the player tries to jam them, or a creature's hobbled escape after the player hacks at its leg with a sword.

These active situations call for more than just dynamic simulation. What causes the machine to move? What causes the creature to limp after it's struck, or at an even lower level, how does the creature move in the first place, and how does the player swing that sword? The answer to these questions is a piece of code called a controller, and I'm willing to bet that implementing robust and interesting controllers will be one of the holy grails for the game industry over the next ten years.

Will Animate for Food

To put it mildly, animation loops leave a great deal to be desired. These are interactive games we're working on, not movies. Animation loops don't react to their environments nor to the player. What does an animation of a character swinging a sword do when the sword hits a wall? Most do nothing: the animation keeps playing, the sword and maybe even the arm penetrate the wall, and the game looks silly. Another alternative is to stop the animation, but again, the movement halts midswing with no momentum, and the game looks silly. Either way, the player's immersion is significantly diminished as the world's consistency is violated. The correct solution — cause the sword transmit forces back through the arm such that it looks as though the sword

Chris Hecker is a total control freak, and I don't just mean that he likes control theory. Tune his gains at checker@d6.com.

actually hit something and the character actually noticed — isn't possible with animation loops.

Of course, you could have a canned animation of the sword hitting a wall, and play that if the sword collides (we'll assume we've solved the problem of blending this strike animation over the currently playing animation). However, the player can move the character and hit any wall or box or pillar at any angle, and a really strong bad guy can grab the blade of the sword and wrestle with the character. Clearly, trying to pre-animate everything quickly becomes intractable.

Rather than trying presumptively to create the behaviors and motions of our creatures, I say we need to take a different tack. We need to teach our characters how to behave, so when slightly different circumstances present themselves, the creature can react in a meaningful way. Put more concretely, we need to write code to tell our characters how to walk and run and do all the things that animation loops currently do in our games. If we want our character to walk, it needs to balance on its legs, pick one foot up, move the foot forward, push off with its other foot, and let the simulated friction between its foot and the floor propel it forward. We write the code that controls the muscles of our virtual character, and its bones and interactions with the world are run through the dynamics simulator. With a little luck — O.K., a lot of luck and hard work — our character walks.

But, our character doesn't just walk. If it does, we wasted a lot of expensive programmer time writing the controller. The character also limps if its leg is chained to a heavy ball. It stumbles if you tie its legs together. It leans over in a strong wind or going up a hill. It walks slower if it's carrying a lot of weight. It collapses in a heap when you blow its leg off. All of these believable behaviors simply emerge from the simulation. This character will always react to its environment because it's simulated, as is everything else in the scene. If a log is in the character's way and the controller isn't smart enough to step over it, the character won't just pass right through the log, or stop moving forward while still walking in its cycle (and looking stupid). The character will trip and fall. Its legs will collide with the log, the force of those collisions will propagate up through its body, its center of mass will keep moving forward because of momentum and will move outside the support of the feet, and the character will fall over. That's interactivity, and you just can't get there with animation loops.

What I'm asking for is very complicated and will take a lot of effort, especially compared to yelling down the hall to the animators and asking for yet another run cycle. But what I'm asking for is vital if we're ever going to achieve truly deep levels of interactivity, where the player's expectations and physical intuition are never violated and the world reacts absolutely to his or her presence.

Unfortunately, when I say, "very complicated," I'm understating the difficulty of writing this controller code. In fact, let me tell you how bad it is right here up front. Dynamic simulation — the topic upon which I spent almost 20,000 words in my old Behind the Screen column, barely scratching its surface — is a piece of cake compared to writing robust controllers. A sufficiently motivated programmer can go read all the dynamics references I list on my web site, implement what he or she learns, and have a top-of-the-line rigid body simulator. Rigid body dynamic simulation is basically a solved problem. By contrast, physical control of locomoting creatures is very far from a solved problem. Even the highest-end SIGGRAPH research is still nowhere near adequate for us to implement controllers that are robust enough and interesting enough to play the part of the main character in an action game. People often talk about the DOOM guy running at 90 miles per hour, not slowing down to go up stairs, turning on a dime, and carrying 10 times his weight in ammo and weapons. Well, the current highest-end human locomotion controllers can't actually get up when they fall down while walking. They fall down a lot. On flat smooth ground. Don't fire the animators yet.

Control Theory

At this point, for the few of you I haven't scared off with my sweeping philosophical prognostications, let's talk about controllers and how we can get started using them today. I've probably given the impression that controllers are all about locomotion. This is not true; locomotion is a subset of the problems that controllers try to solve, and thankfully, most of the other problems are much simpler than locomotion. Best of all, we'll learn some neat math that's used by almost all controllers. Let's start at the beginning.

Control theory centers around the concept of a system that has inputs, a process (sometimes called the plant, as in processing plant), and outputs. The idea is to get a desired result on the output by changing the inputs, possibly in the face of disturbances and uncertainty. The task might be something relatively simple, such as controlling the temperature in your living room by turning the heater and air conditioning on and off. It might be something moderately complicated, such as controlling how thinly to slice the potatoes and long to cook them to get a tasty potato chip. It might be something really complicated, such as getting a biped to walk across the room by controlling the muscle contractions in its legs.

Control theory tells us how to analyze the system that we're trying to control, and then tells us how to design a controller that will get the desired results. At least, that's the idea. For simple systems, the theory has been very well developed and you can basically plug in your system and out comes a controller that works really well. For more complex systems with lots of nonlinearities and discontinuities, controller design is an active area of research (read: there's still a fair amount of heuristic voodoo involved).

There are infinitely many places to use controllers in games. I've already talked about locomotion, and even though I was a bit harsh on the current state of the art, there are many interesting nonhuman locomotion controllers out there (creatures that successfully hop, swim, fly, and so on). And to be fair, robust and convincing human locomotion is the hardest possible locomotion problem. Besides locomotion, the list of controllable systems goes on: automatic doors, elevator platforms, cars or other motorized moving objects, sidekicks following the player, homing missiles, and so on. Basically, controllable systems include anything that has outputs that you want to control with known inputs.

Additionally, hybrid simulations may be of interest to game developers. Perhaps we could design a system whereby only the sword arm is controlled physically, while the rest of the body is animated. At least with a hybrid technique, the arm will react correctly, and the controller writer doesn't have to figure out how to solve the entire character control problem all at once.

A SISO Controller

Control theory is a rather large discipline with some pretty intense math associated with it, so we're only going to touch on some of its basic aspects in this article. We'll focus on the simplest type of system, the Single Input Single Output (SISO) system, and we'll build a commonly used controller for our example. In our case, and in most cases that you'll come across when controlling physical objects in your simulator, our input to the system will be forces and torque, and the output will be some kind of position or velocity.

Let's say we want to control the way in which a character's arm reaches out to place its hand on a door handle or swings the hand in a circle at a certain velocity for a punch. The control that we'd exert over this system would be the torque at the joints, which simulate the muscles of the character. By limiting our input to exerting torque, we allow the system to interact with the world: if a wall is in the way, its solidity will overpower the muscle torque and the arm will stop moving, as you'd expect.

Conversely, in a system controlled by inverse kinematics, the arm can't respond to collisions because the joint angles are fed in directly — it's just another way of generating an animation. You can use inverse kinematics to generate goal angles for a controller, however. The controller tries to attain the goal angles using joint torques, and we get the world interaction we desire (for more information on IK, see Jeff Lander's Graphic Content columns "Oh My God, I Inverted Kine!," September 1998 and "Making Kine More Flexible," November 1998).

We'll use an arm for our example SISO system, but because we only want one input, our arm only has one joint (Figure 1). Our arm isn't incredibly exciting or useful, I admit, but it'll do the job. Now that we've got a system to control, let's look at the steps for building a controller:

1. Study the system.
2. Model the system.
3. Write the controller.
4. Test the controller.
5. Iterate.

In step 1, we ask ourselves a bunch of questions so we can get to know our system. What are the inputs and outputs of the system? What is our goal for the system? That is, which outputs do we want to control? How does the system behave without a controller (called its open-loop response)?

For step 2, we build a mathematical model of our system. If our system is running open-loop in our game, then we must already have a model for it in some sense. However, that model might not be very clear or appropriate for designing a controller. The level of detail that you build into this model depends on how precisely you want your controller to perform. An exact model might enable you to design a controller that exactly controls the system, but it might be very expensive in terms of time to develop. An approximation might do almost as well for much less cost. Or, the exact model might not yield to analysis as easily as a carefully chosen approximation.

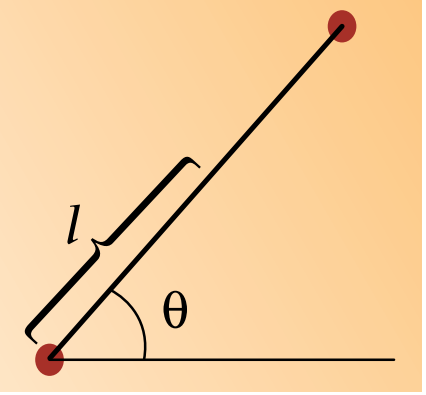
Now we choose a controller and implement it in step 3. Choosing the right controller is a matter of experience, knowledge, and a bit of luck. Implementing it is a matter of understanding the math behind the controller you've chosen, picking the parameter values for the controller, and writing the code.

Next, in step 4, we run the controller in the game. One should always back-up one's work before trying this step.

Finally, because the previous step didn't turn out very well, we iterate in step 5. We can loop all the way back to step 1 and try to learn more about the system, we can make only minor tweaks in the controller parameters and retest, or we can try anything in between.

I should point out that we have it much easier than our real-world control engineer friends. Not only won't we break a multimillion dollar machine or kill a hapless patient in step 4, but we also know exactly how our systems work and what the disturbances on them will be. Be aware that control theory books spend a fair amount of time explaining problems that we simply won't face in building our simulation.

FIGURE 1. *Our incredibly nonintuitive single-jointed arm.*



Step One

Our system is the one-jointed arm shown in Figure 1. The angle of the arm, θ, is measured off of the horizontal axis. The input to the system is torque about the joint, and the output is the resulting angle. We could have said that the output was the end-effector position, but it's clear that our end-effector is going around in a circle and that's about it. I decided to keep it simple and just use the angle.

We want to set a desired angle (sometimes called a setpoint), and have our arm achieve it quickly and reliably. We don't want our arm to oscillate around the point, or overshoot the point by a wide margin as it approaches. Our arm should also reach the desired angle whether or not gravity is in effect.

If you run the sample application that accompanies this article (which you can download from my web site), you can see the open loop behavior of the system. If the arm is not on the setpoint and gravity is not turned on, then the arm just sits there regardless of where you put the setpoint. If gravity is on, the arm falls down to a dangling position, but because our virtual world doesn't have damping, it oscillates about the vertical like a pendulum. Neither of these behaviors is remotely close to what we want, so clearly a controller is needed.

Step Two

Now we need to build a mathematical model for the system. By this I mean we need to write out the differential equations of motion for the arm so we can see how the system behaves over time. Going forward, I need to assume that you understand a bit of calculus or have read at least the first column in my physics series in *Game Developer* ("Physics, The Next Frontier," Behind the Screen, October/November 1996); the columns are available on my web site, which is referenced at the end of this article.

We're going to model this arm as a single-degree-of-freedom generalized-coordinate system. The equation of motion for the arm is

$$I\ddot{\theta} = -lmg\cos\theta + u \qquad \text{Eq. 1}$$

where I is the moment of inertia of the arm about the joint, θ is the arm's angle (remember that the dots above the symbol mean time differentiation, so the left-hand side shows the second derivative of θ with respect to time, $d^2\theta/dt^2$ or angular acceleration), l is distance to the arm's center of mass (as shown in Figure 1), m is the mass of the arm, g is the gravitational constant, and u is the (currently unknown) torque we're going to generate with our controller.

As we look at this equation (setting u=0 for now, because we haven't actually designed the controller yet), we can see a few things already. First, if gravity is enabled, this is a nonlinear ordinary differential equation. It's nonlinear because of that θ in the cosine term. This term arises because gravity torques the joint differently depending on the angle of the arm, and that torque in turn affects the angle of the arm through integration.

Assume that gravity is disabled. Then the entire right-hand side is 0, which means when you solve for the acceleration by dividing both sides by I, you get 0 — the arm's velocity isn't changing. That is, if the arm isn't moving, it won't start moving, or if it's already moving, it won't change the speed at which it's moving. In other words, our model has no damping. If you want to test this assertion, you can recompile the sample application to start the arm with an initial angular velocity and see if it changes with no other inputs to the system.

Next, assume that gravity is enabled. The arm will have an acceleration most of the time, except when the cosine term equals 0, namely straight up (90 degrees) and straight down (270 degrees). If the arm is already pointing straight down, then the angle won't change under gravity. If the arm is vertically balanced the arm at the joint, then the slightest perturbation in the straight up position will give gravity that bit of tangential movement it needs to accelerate the arm. That's why vertically balancing a pole is difficult, but holding it pointing down is easy.

The acceleration is greatest when the arm is parallel to the horizontal axis, at either 0 degrees or 180 degrees. In these positions, all the force of gravity goes into rotating the arm (as opposed to working against the joint, as in the vertical case). If we want to counteract gravity with our controller, we'd better be able to deal with gravity exerting different amounts of torque in different positions.

At this point in the analysis, we could study the behavior of the differential equation after linearizing it about a few points of interest. Linearization is a technique that turns a nonlinear differential equation, such as our Eq. 1, into a linear differential equation. This linear equation accurately emulates the original in the neighborhood of a chosen point. We'd linearize because the theory surrounding linear differential equations is much more developed than for nonlinear ones. This advanced theory lets us figure out exactly how the equations behave over time and in response to different types of inputs. Obviously, if the equations that describe your system are linear in the first place (as is our equation without gravity), you get all of these tools without having to linearize. Unfortunately, we're not going to have space to cover anything more about linearization, but you'll definitely learn more about it if you read about control theory on your own.

Step Three

At last we get to choose and implement a controller for our system. Rather than attempt to enumerate all the different kinds of controllers that are applicable to our SISO system, I'm simply going to pick the very common Proportional Integral Derivative (PID) controller. The PID controller is actually three controllers in one that work together to control the system.

PID controllers, like almost all controllers, use feedback to regulate the system. In other words, the controller uses the current outputs to modify the inputs to get the desired result. Another type of controller, the feedforward controller, relies on its internal model to be accurate enough that it doesn't have to look at the output state to tell how it's doing. We don't have room to discuss feedforward in detail, but because we have perfect models for our systems (by virtue of our simulation), feedforward is a viable alternative to feedback.

A PID controller uses the difference between the current output state and the desired setpoint as the controller's input. This difference is called the error, and is denoted by

$$e = \theta_d - \theta \qquad \text{Eq. 2}$$

Here, θ_d is the desired angle, and θ is the actual angle. The controller tries to drive e to zero.

The proportional part of the PID controller's name refers to its act of directly feeding back this error, multiplied by a positive constant called the proportional gain, as the control torque.

$$u = k_p e$$

Let's look at what happens when we substitute this control torque into Eq. 1, assuming gravity is 0. We'll also assume that the desired angle is 0 just to simplify things.

$$I\ddot{\theta} = -k_p\theta$$

You may recognize that this is an equation for an undamped spring. This is the same differential equation that we'd get if we had a particle in 1D with a spring attached to the origin. As we know, an undamped spring will oscillate about the setpoint forever. So, while this proportional controller gets us to the setpoint, it doesn't keep us there. You can test this in the sample application by turning on the proportional part of the controller and watching it work.

In the real world, people use proportional controllers without oscillation all the time, but in the real world there's no such thing as an undamped system. Friction robs all real systems of energy, so in the physical world, proportional control is often all you need.

We can add our own damping to our system with the derivative part of PID. The derivative controller uses the time derivative of the error to try to force the velocity of the output to 0 (or actually, to force the velocity of the output to the desired velocity, but for our arm that's 0). The error derivative is

$$\dot{e} = \dot{\theta}_d - \dot{\theta}$$

Adding this derivative to our control torque (with a derivative gain, k_d) gives

$$u = k_p e + k_d \dot{e}$$

When we substitute this equation into Eq. 1 (again, assuming no gravity), we get the equation for a damped spring. You can see the results by running the sample application with the proportional and the derivative controllers active. This controller is often used in the real world as well, where it's known as — surprise — a PD controller, or PDC.

The final part of our controller, the integral, needs a bit of motivation, because it appears that the PD controller is doing just fine without the integral. This observation may seem valid at first glance, but what happens when you turn on the gravity? The arm droops and hangs below the desired angle, not quite getting up the steam to reach its goal.

This error is called a steady state error. The arm will sit there forever, never getting worse, but never getting better. We need to add something to the system to eliminate this steady state error. You might ask why we can't just crank up the proportional gain to eliminate the error. We could do that, but we'd never eliminate it completely using only the proportional gain. Increasing the gain is equivalent to making the spring stronger, but it's always going to sag a bit when a constant force, such as gravity, is applied. Furthermore, really large gains make for unstable systems. There is a better way.

We need our controller to detect when it's not quite doing its job and to add a little extra torque automatically when necessary. We can implement this capability by integrating the error over time and using that integration as another control torque (the I in our PID controller).

$$u = k_p e + k_d \dot{e} + k_i \int e dt \qquad \text{Eq. 3}$$

Think about how the integral of the error (implemented in discrete code as a running sum of the e term on each timestep) will affect the control torque for a steady state error. If we start out with some steady state error, the integral will begin increasing and the error will start decreasing as the controller exerts the extra torque that's needed. As the error gets smaller, the integral is increasing at a much slower rate, until finally the error goes to 0 and the integral term is now exactly the constant extra torque needed to eliminate the steady state error. Neat, huh? In reality, the integral term will probably exert too much torque and the arm will overshoot. Then the negative error will start decreasing the term and it will eventually settle down to 0 steady state error. Integral controllers not only increase the overshoot, but also have some other negative performance effects that you can experiment with in the sample. As you might expect, using the integral term offers tradeoffs.

To implement a PID controller, you need to pick initial values for the three gains in Eq. 3. You can do this in any number of ways, ranging from the completely *ad hoc* to the highly analytical. We don't have space to go into the various techniques, but you can read about them in the references.

Step Four

If you've been following along, you've already been playing with the sample application. But now you should play with it a bit more to get a feel for how the various parts of the controller behave. One interesting thing that I discovered is that the integral controller will suck up any steady state error, not just the gravity. Turn off gravity, turn on the proportional and derivative controllers, and then hold down the [i] key, increasing the desired arm angle, until the keyboard repeat sets in. You'll see the arm chase its desired angle around in circles, but it will always be a bit behind, as you'd expect. Now turn on the integral controller as well. The arm will start out behind, but will soon catch up and sit at the desired angle as it moves around the circle. The integral controller has eliminated the rotating steady state error. Of course, when you stop increasing the desired angle, the arm overshoots, but then comes back to rest on the setpoint.

Step Five

I already iterated the sample application while I was writing it, but you should play with it by changing the gains and recompiling to see what effect they have on the behavior. See if you can improve the performance of the arm. Notice how the lame Euler integrator handles (or doesn't handle) cranking up the gains.

Tuning gains for PID controllers is a huge topic in control theory, and is far beyond the scope of this article. Several of the references talk about a number of different ways to tune the gains to achieve various performance goals, such as overshoot, rise time, settling time, and steady state error. You'll also find ways of generating the gains directly from an optimization procedure.

Beyond Physical Objects

Controllers don't have to be limited to controlling joints on creatures, or even to controlling physical objects at all. A piece of code that looks at how long the previous frame took to render and then adjusts the level of detail of the scene to maintain a constant frame rate is a controller. The code that your enemy fighter planes use to track the player is a controller. In general, any piece of artificial intelligence code is a controller, and can be analyzed using the tools of control theory. Try adding some controllers to your game, and let me know about your experiences. ■

FOR FURTHER INFO

- **Here are some URLs for interesting controller tutorials on the Web:**
  http://www.engin.umich.edu/group/ctm/index.html
  http://www.eng.uml.edu/Dept/Chemical/onlinec/white/sdyn/s7/s7intro/s7intro.html
  http://www.manufacturing.net/magazine/ce/archives/1998/ctlo801.98/08abas.htm
- **You can read my physics articles and references and download the sample application at my web site:**
  http://www.d6.com/users/checker/dynamics.htm

Online Justice Systems

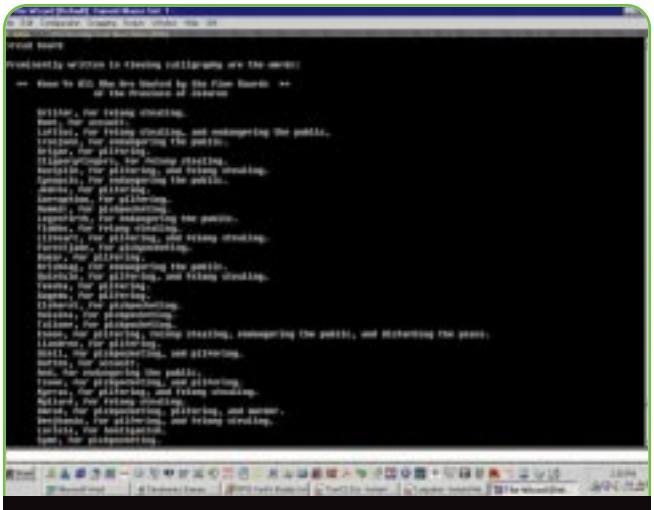
by Derek Sanderson

In order to build a successful online game, you must build a sense of community among your players. One of the biggest challenges to successful community building in online role-playing games is tempering the problems caused by players killing or stealing from other players. Such problems are known more generally as player-vs.-player conflict (PvP). Over the course of time, different games developed by different companies have sought to control this problem through a variety of methods. This article touches upon PvP control strategies used by Simutronics Corp., where I am currently employed, as well as strategies used by Origin Systems in ULTIMA ONLINE and by 989 Studios in its upcoming game EVERQUEST. While there is no single correct way to maintain order in an online game, by examining these companies' strategies for restraining nonconsensual PvP, I have created a set of general guidelines that should be considered when designing an online justice system.

The Current Methods

While existing systems for controlling PvP show some very creative design solutions, each of the following strategies nonetheless suffers from certain flaws that arise from the different priorities assigned to game play elements.

ADMINISTRATIVE CONTROL (SIMUTRONICS' GEMSTONE AND DRAGONREALMS). Simutronics gives its players wide leeway in resolving conflicts among themselves, and generally limits its hard-coded restrictions on attacking other characters. New players may not be attacked and are not strong enough to harm one another. Stealing from a person's inventory is limited to coins and small gems, and corpse looting is either not possible (as in GEMSTONE III) or has safeguards that allow careful players to prevent it from happening (as is the case in DRAGONREALMS). The leeway afforded the players allows the responsible players a great degree of freedom in how they play their characters. To balance out this freedom, though, Simutronics strictly polices its player base. Its players' terms and conditions agreement, for example, states, "What is not acceptable is to initiate combat against unsuspecting victims. Anyone exhibiting such behavior, especially one who



DRAGONREALMS' rogues' gallery.

Derek Sanderson has held several design and customer service positions during his tenure with Simutronics, and has recently settled in as the company's lead designer. He is currently pondering the career ramifications of commuting to work in a go-kart, and welcomes input on this subject at rpgvault@aol.com.

An innocent is attacked in ULTIMA ONLINE

ULTIMA ONLINE's facility for reporting crimes.

chooses to prey upon weaker players for his or her own enjoyment, may be in violation of…policy."

"Unsuspecting victims" can be a difficult standard to enforce. It's fairly obvious when someone is on a mass-murder spree, and we remove such characters from our games immediately. If the offender is an experienced player who knows better, we generally penalize the account with official warnings and restrictions from playing for a period of time. If the player is new to our game, we explain our policies. If any player is unwilling to abide by the rules, Simutronics usually recommends that he or she try a product better suited to his or her tastes.

Simutronics' methods are effective for controlling those players who understand the rules and deliberately choose to violate them. The system's main weakness, however, is in handling conflicts in which the two parties disagree over whether consent to violence was given. For example, if player A makes a few choice comments about Player B's suspected lineage, and Player B attacks, is the conflict consensual? Some Simutronics staffers would say consent was implicit in the insult, but others consider consent to be something that must be explicitly stated by the victim prior to any attack. When staff tread such nebulous ground, they're fighting a battle that is impossible to win. No matter how they handle the conflict, their intervention often creates hard feelings among the players. Resolving these squabbles also uses staff time that could be spent on game development, requiring a higher developer-to-player ratio than would otherwise be necessary. Simutronics has made the choice to incur these higher costs in order to maintain games in which our customers may play in relative safety from arbitrary attacks. Whether such a solution would be viable in another game depends on the developers' goals and budget.

PLAYER POLICING (ORIGIN SYSTEMS' ULTIMA ONLINE). ULTIMA ONLINE's developers decided to forgo administrative policing and leave its justice system entirely in the hands of the players. Raph Koster, ULTIMA ONLINE's lead designer, said Origin designed the game this way in the hopes that, "given the tools to police their own environment, [players] would do so…. Our experience was that every method of administratively imposed policing either failed or led to intense resentment of the administrators of the game. We were particularly concerned because traditional models on MUDs for enforcing social mores were very administrator-intensive, requiring a large number of skilled administrators willing to devote a lot of time to soothing ruffled feathers on the part of players who felt wronged. In a commercial venture of a large scale, we didn't think this was sustainable."

Allowing your customers to police themselves is a noble goal, but one that is difficult to implement. The most infamous result of the ULTIMA ONLINE hands-off policy was the gangs of player killers (PKs) that formed. Such gangs would station themselves at key locations in the game and ambush any poor soul foolish enough to travel with a group smaller than a mob. "I just got PK'd," was a refrain commonly heard outside the game's banks, where naked adventurers would come to beg for money to re-equip themselves. Some players formed anti-PK militias, but, as Raph says, they were "inadequate for handling the problem of player killing. The actions of the few police were both insufficient in quantity and inadequate in severity to curb the activity of the player killers and the player thieves."

In response to the problem, Origin instituted a variety of tools to allow the players even greater control over their environment. Under the current system, all characters begin the game flagged as "innocent," with their names highlighted in a bright, happy blue. Steal from, attack, or loot the dead body of an innocent — including an NPC — and your character's name will be highlight gray, branded a criminal and open to attack by anyone. Kill an innocent player character, and that person is given the option to report you as a murderer and place a bounty on your head. Kill five or more innocents in a short period of time, and your character is flagged a "murderer," unable to use shops or access your bank account, and subject to being slain on sight by other adventurers who wish to collect the bounty on your head.

ULTIMA ONLINE's greatest strength is that it places administration of PvP entirely in the hands of its players, giving them an unrivaled sense that they, and not the Origin staff, control their world. The benefit of this feeling among players shouldn't be underestimated; it's a powerful contributor to a sense of immersion in the game environment. The system is

Bounty available for the heads of ULTIMA ONLINE criminals.

weak, however, in controlling random aggression. Only after five reported kills does PvP activity have any real repercussions for the aggressor, and the game does little to track long-term aggressive behavior. If a player waits just eight hours of online time between murders, he can kill one player a day without ever reaching the murderer threshold. No penalty exists (other than being flagged a "criminal" for a short period of time) for attacking someone unless that person dies as a result of his or her injuries. Harassment attacks that fall short of a murder are still extremely common in ULTIMA ONLINE. I was, for example, attacked by total strangers an average of once a day over three weeks of playing while writing this article, and killed three times. (Note to game designers: other game designers get really grumpy when your players kill them, especially when their colleagues make fun of their poor fighting skills.)

PLAYER-TOGGLED FLAGS (989 STUDIOS' EVERQUEST). The developers of 989 Studio's EVERQUEST (which should be going on sale at around the same time you read this article) plan to implement a flagging system that will mark characters either as able to attack and be attacked by other players (+PK), or completely unable to engage in such activities (-PK). The method is a common one for controlling violence in small text-based MUDs, but my experience suggests that in a large-scale game, where the community is of sufficient size to allow true anonymity, the use of "throwaway" (also known as "mule") troublemaker characters with -PK flags will abound. Such characters, immune from physical harm, can do many nonviolent but extremely annoying things to other players, such as following another character around wherever he goes, blocking entries to important areas, attacking monsters other players are already fighting, engaging in verbal harassment, holding goods stolen by +PK characters, running cons and scams, refusing to leave someone's home, and more.

Brad McQuaid, EVERQUEST's producer, says his team is aware of the PK flag's potential abuses and is prepared to combat them. The game will have a squelch command to combat verbal harassment, and out-of-context (non-role–played) harassment will result in punitive measures against the offender's account. As for killing the creature another person is fighting, Brad says, "…the player or group that does the most damage to an NPC gets to loot it and receives the experience for the kill. This stops the jerk who comes along and gives the killing blow to a creature even though another person or group had engaged the NPC long before. He's welcome to deliver the killing blow, but he will receive no experience for doing so."

The general principle behind the EVERQUEST kill-stealing prevention is sound, but what does one do about the high-level, -PK player who goes to a low-level hunting ground and steals kills repeatedly, doing more damage to creatures than the new players fighting them by virtue of an incredible advantage in skill? Does one block entry to such areas for high-level players? Does one prevent high-level players from attacking low-level monsters? Does one simply warn the player for disruption? The number of ways to get around the game design illustrates the greatest danger to the PK flag solution, namely that it creates an invulnerable subclass of character that players will be unable to police, thus shifting the burden (read: increasing staffing costs) to the game administrators. However, I suspect the flagging solution will be popular with a significant portion of EVERQUEST's customer base, because it allows responsible players who don't enjoy PvP to play without interference from their more aggressive cohorts.

A Few General Guidelines

If the perfect solution has yet to be implemented, then what is the answer to managing PvP? A complete system design is beyond the scope of this article, but here a few things to be considered when designing an anti-PvP system.

REDUCE OVERHEAD BY MINIMIZING STAFF INTERVENTION IN PLAYER AFFAIRS. Minimizing staff intervention in player affairs is a principle that should be followed across all aspects of your game design, and it's particularly true for your game's PvP controls. Players will always exploit loopholes in your design to their advantage, and when they do, the best way to resolve the problem is to alter your code to prevent the undesired activity.

A good example of players using a system contrary to its intended design is the process by which a character of the cleric class may resurrect another character in DRAGONREALMS. When a character dies, a counter starts tracking skill loss, and the longer a character has been dead when resurrected, the larger the loss will be. Clerics are able to cast a Soul Bond spell that will neutralize this skill loss, and players generally expect that a cleric will do so before performing the resurrection. In the early implementation of this system, however, players used the process to force skill loss by intentionally resurrecting characters without first casting the Soul Bond spell.

Although the system mechanics allowed such activity, it didn't fall within the behavior expected by the staff, and several clerics had their ability to cast the spell temporarily taken away for abusing the loophole before we coded changes to close it. Although our intervention took care of the immediate problem and made the victims of the aggression happy, it tended to make the players against whom we took action resentful. A pattern of such staff interference can result in an antagonis-

ULTIMA ONLINE's guild system allows a sort of regulated PvP.

tic relationship between your customers and staff and increased expenses to cover the lost development time spent correcting player behavior. It can also foster an environment in which players expect staff to handle their disputes, generating an ever-increasing number of assistance calls as your customer base grows. Whether you are willing to pay such costs is up to you.

MAKE ALL METHODS OF PVP REPORTABLE TO A HARD-CODED JUSTICE SYSTEM. Players should be able to report all forms of PvP to the game's justice system. Possible methods include presenting murder victims with a pop-up window, such as the one ULTIMA ONLINE display, or allowing players to file complaints with NPC guards or magistrates. Whatever the reporting mechanism, it's important to include all forms of PvP, such as theft, corpse looting, casting offensive spells, being harmed by an area-effect spell, being attacked with a weapon, being killed, or having player-controlled NPCs or creatures perform any of these offenses. The reporting mechanism should be intuitive and easily accessible, but should involve some effort on the part of the reporting player so only the truly important attacks are reported.

Extra care must be taken with area-effect spells. (Area-effect magic spells affect all characters within a certain radius of the character who cast the spell.) A common player-killer tactic in ULTIMA ONLINE is to enter someone's area-effect spell deliberately, then kill that person after the system flags them as "criminal" because of the damage the spell causes to the player-killer's character. If you cannot detect such behavior with your code, possible solutions are either not to design such spells or to make them non-harmful to other player characters. You'll lose a bit of realism, but the loss will be vastly offset by your closure of this common PvP loophole.

Another area of special attention should be your PvP theft-detection mechanism. Here's an all-too-common scenario: Character A has a pocket full of coins, and encounters Character B. Character B steals the coins from Character A, but Character A fails his skill check to notice the theft attempt. The person playing Character A, however, notices the coins are gone, and draws the very reasonable conclusion that they were stolen by Character B. Under most game systems, however, Character A has no recourse, and will be labeled criminal if he attacks the thief in an attempt to recover the money. Such thefts are generally the most frustrating for your customers, because the person playing the thief will often use his immunity to taunt his victims. The solution is to allow the victim to report anyone to the justice system, with penalties for false accusations.

MAKE ANTI-PVP SYSTEMS ACTIVATE ONLY UPON PLAYER REQUEST. Only the victim of an online crime truly knows whether the actions against his or her character merit a reaction by the justice system. The person who harmed the character, for example, may be engaged in a friendly duel, or the violence may be a role-played conflict that the victim wishes to avenge personally.

ULTIMA ONLINE has an excellent implementation of player-initiated justice, although it contains a few loopholes. If one attacks an innocent, for example, one is automatically flagged "criminal," even if the attack were accidental or entirely consensual. A character of mine was once killed and looted by a stranger for being "gray" (indicating criminal status) after I accidentally hit a companion while in combat. My killer wasn't impressed with my explanation, and I signed off that day much poorer than when I began. If I'd been murdered as "innocent," however, I would have been able to report the attacker and place a bounty on his head.

ULTIMA ONLINE also has a way to remove players entirely from the justice system if they are members of a player-run guild. The guildmaster of a guild may issue an official declaration of war on another guild, and if the declaration is reciprocated, the two guilds enter into a state of conflict in which members may attack, kill, steal from, and loot each other freely without becoming criminals. A second level of warfare offers even more uncontrolled PvP conflict. When a guild's leader becomes famous enough, he or she is given the power to declare the guild an Order or Chaos guild. Upon doing so, the guild enters a state of perpetual warfare with all guilds of the opposing type, and members may fight with opposing guild members at any time, anywhere. "This provides the 'ambush around every corner' feeling that this type of player values," says Raph. "The warfare system proved to be very popular, with 10 percent of guilds converting over to the 'free-for-all' guild type as soon as it became available."

In EVERQUEST, those characters who are flagged +PK will be able to attack and kill other +PK players at will, but a hard-coded race and alignment system will determine how the rest of the world reacts to the slaying. A player-character ogre, for example, is from a classically evil race. If that ogre kills a player-character elf, which is a classically good race, the ogre will, says Brad, "[when] he returns to his home town, be welcomed as a hero…. In this sense, player killing is encouraged in EVERQUEST where it makes sense." However, if elven guards observed the ogre attacking the elf, they would likely intervene. Furthermore, a character that kills members of his or her own race would eventually find NPCs of that race reacting poorly to the character.

MAKE REVENGE AN OPTION. This section merits an entire article in itself, so I'll mention it only for completeness and keep my comments brief. The key idea here is that many players would prefer to fight back when subjected to a PvP attack and wouldn't enjoy reporting the activity to an NPC system. When a

In EVERQUEST, players who wish to attack other players must be flagged +PK and even then can only attack other +PK characters.

character is the subject of aggression, the aggressor should be flagged so the victim may fight back without penalty, whether it be an immediate response or a later ambush. ULTIMA ONLINE has implemented this principle by flagging an aggressor attackable for two minutes per attack. My experience suggests that this isn't enough time; for me, at least, my initial reaction to an attack was to flee to heal myself. By the time I'd recovered from the initial ambush, my aggressor was usually no longer eligible for a penalty-free attack.

MAKE PVP MUCH LESS PROFITABLE THAN PLAYER-VS.-GAME ACTIVITY. A certain percentage of those who kill or steal from other players do so simply because player-characters tend to be far more wealthy relative to the level of danger they present than NPCs or creatures. Take away the profit from PvP, and you'll curtail a certain percentage of it. Ensure a stable supply of player-vs.-game activity, and you'll decrease it even more.

The most obvious way to lessen the profitability of player-killing is to restrict looting of dead characters, without removing the ability for players to help their fallen comrades. One method is to prevent looting entirely, although a corresponding mechanism must be created to allow recovery of stolen items if a thief is tracked down and slain. Another is to allow looting only under specific circumstances. ULTIMA ONLINE flags looters as criminals, making them vulnerable to attacks or having the NPC guards called to execute them.

EVERQUEST had not yet finalized its corpse-looting restrictions at the time I wrote this article, but Brad says that the developers are considering several options. "…If you are -PK and you die, only you or someone to whom you give consent may loot your corpse. If you are +PK and come across the corpse of another +PK character, you currently are free to loot it. We are, however, experimenting with some limitations to make player killing more viable. We will test a system in which the killer may loot his victim only once, and may take only one item of choice from the corpse."

If you implement such restrictions, provide enough monsters or NPCs to meet player demand, and give players enough wealth for them to feel they are making reasonable progress, you'll drive player activity towards the monsters. Make your monsters and NPCs too poor, or fail to spawn enough of them, and your players will turn on each other. Similarly, it's important to give player thieves enough creature or NPC targets to make the class economically viable, else a percentage of those who would normally only steal from non-players will turn to PvP stealing.

RESTRICT THE ABILITY OF NEW CHARACTERS TO HARM OTHERS. Players will always use system loopholes to maximize their gains, and if a new character is able to accomplish a highly dangerous task and realize the same gain as an older character, then the use of the aforementioned throwaway characters will abound. This is especially true when more than one character is available to the customer on the same account, or if free trial accounts are accessible to the players without the purchase of a retail product.

Stealing and scamming are the most common uses for throwaway characters, and is usually accomplished in a way that circumvents the system's intended design. In DRAGONREALMS, for example, throwaways are frequently created to loot weapons dropped on the ground by dead adventurers. The throwaways then pass the weapons through friends and back to the main character on the thief's account, disguising the true identity of the thief and making redress impossible. Throwaways also exploit loopholes in our player-to-player item exchange mechanisms, tricking adventurers out of their goods and then passing the profits to the real character on the account.

Simutronics is not alone in facing problems with throwaway characters. One common trick in ULTIMA ONLINE used to be for two players to create thief characters, stand near a bank, and steal items from adventurers. If the victim detected the theft and called for the guards, the thief was executed. The thief's partner would then lift the item from the dead body, and the original victim would have no way to get it back. Origin recently fixed this exploit by having the guards return stolen items to the victim if the thief were killed within city limits within two minutes of the theft.

CHARGE PLAYERS FOR PVP ACTIVITY IN A CURRENCY THAT IS VALUABLE TO THEM. When an activity has a perceived cost, the frequency of that activity will always decrease. It's a basic supply-and-demand formula; make PvP more expensive, and fewer people will choose to purchase it. Those who choose to engage in PvP activities will therefore have to decide before every assault whether they are willing to pay the price. Scale the cost of that activity as its frequency increases, you'll prevent repeated abuse by older, richer players.

You could, for example, make the tax a monetary one, and charge an aggressor 25 coins for his or her first reported murder, 50 for the next, then 200, 400, 800, and so on, allowing the character's criminal past to decay one fine level per week if he or she refrains from all criminal activity. Characters who are unable to pay the fine could be restricted to a debtors' area from which they couldn't leave until they had performed enough low-paying menial tasks to pay off their debts. Preventing someone from leaving until their fines are paid would stop savvy players from offloading their valuables onto storage characters before going on a killing spree; allowing them to perform work to escape

would prevent anyone but the worst cases from being trapped inside.

A tax doesn't have to be a monetary one. A character could, for example, lose an increasing number of experience points or skills with each subsequent PvP report against him. If he or she reaches a certain threshold, that character would be hit with a curse that prevents all aggressive activity for extended periods of time. Another possibility would be to toss characters in a jail cell for increasing periods of time, where they must stay until their sentences are served.

Whatever you choose for your tax, the penalties should start at negligible levels and scale up exponentially rather than linearly. Low initial penalties, when combined with a slow decay of criminal histories, will allow new players to learn the system, allow all players to make the occasional mistake, and allow normally law-abiding characters to take the occasional swing at someone who really, really deserves it. Only those who try to make a career of harming other characters without their consent will be subject to heavy fines.

Balancing Good and Bad

There is no magic bullet solution to solve all of the PvP problems inherent to an online role-playing game community. No matter what methods one uses, players will always find ways to harass, pester, and annoy each other, so trying to eliminate all forms of aggression isn't a realistic goal. If, however, you give proper reporting tools to the victims of non-consensual PvP, allow players outlets for consensual attacks, and make everything else costly, you'll find the problems reduced to manageable levels. There is room for all styles of play in a properly designed role-playing game, and finding the correct balance is key. ■

Acknowledgements

I wish to extend a special thanks to Raph Koster and Teresa Potts of Origin Systems, and to Brad McQuaid of 989 Studios for their timely assistance. This article would not have been possible without them. Thanks also to Simutronics designer Emily Jacobson for her ruthless editing.

POSTMORTEM

Multitude's FIRETEAM

by Art Min

Our goal with FIRETEAM was to create a complete online game experience. The Internet gives game designers the ability to take multiplayer gaming one step further by creating a community, something that wasn't possible before online games came about. Multitude wanted to take the next step in gaming evolution by making the community a significant part of our product. In other games, such as DIABLO or QUAKE, the players were creating communities themselves, mostly through their own web sites. Multitude, on the other hand, devoted significant development time to creating tools that would help the community. We spent as much time on FIRETEAM's lobby and community web pages as on the game engine itself. Our goal was to create a game

Art Min is FIRETEAM's project leader and the cofounder of Multitude. He lives in the San Francisco Bay Area. Art relaxes by hitting a little black object on a cold surface and attempting to learn how to paint, though not at the same time (yet). He can be reached at minman@multitude.com or minman@alum.mit.edu.

that would make people say, "Wow, this is what I've wanted from an Internet game."

FIRETEAM is an online-only gaming experience. The actual game play is a squad-based tactical combat. Players can communicate with other members of their team using Multitude's voice technology. Each player controls one character in the battlefield. The game uses an isometric, three-quarter view 2D graphics engine. There are four different FIRETEAM scenarios, and each game session is ten minutes long. The scenarios are very sports-like in their design to help promote team play. Equally important to the FIRETEAM experience is the lobby, where players can view other players' statistics, chat between games, and find squad mates and enemies for their games. The last component of FIRETEAM is the community web pages, which display players' complete statistics and provide support for FIRETEAM Companies (which are similar to QUAKE Clans). On the community web pages, players can create companies, add/kick members, and access private Company bulletin boards.

Brief History

FIRETEAM evolved dramatically over its first year of development. Multitude was originally founded to create the "ultimate online game," which was to be a large persistent science fiction world. We knew that there would be some competition because ULTIMA ONLINE had already been announced, although Origin hadn't yet performed any alpha or beta testing. We spent months writing and planning for a massively multiplayer online game set in a futuristic world. The project was to have a server team of around 10 people and a game team approximately double that size. As we were designing around our original concept for the game, our desire to make a persistent-world game work as well as a single-player game presented us with many hard technical and design problems. On the good side, it was during this process that we finalized the design spec for our combat engine. The combat engine was inspired by X-COM: UFO DEFENSE, emphasizing squad combat with features such as line-of-sight.

We soon realized that our new company's financing was coming along very slowly and that we needed a much more easily attainable goal (due to lack of resources, both financial and human) that would still showcase the unique voice technology that we'd developed. We looked at the combat engine specification, our voice technology, and the Internet technology that we were designing and realized that we could make a great tactical team game. So at that point, we decided to abandon the large persistent world and make team play the essence of the game.

Designing a multiplayer game is very different from designing a single-player game. I've heard that in many games, the multiplayer component was added on only because marketing had requested the feature; this approach can make the multiplayer experience less than ideal. In a single-player game, the player is the hero and the focus of the game experience. The player should be able to win 100 percent of the time (with some effort). In a multiplayer game, a player should win 50 percent of his or her games against an equivalently skilled player. The thrill of a multiplayer game shouldn't be in the winning, but more in the process and the actual competition. Team play gives players a deep gaming experience, even if they lose. Our efforts to create engaging multiplayer game play were made even more effective by our voice technology, which allowed players to hear the emotions of their fellow players.

FIRETEAM's Components

FIRETEAM's network architecture is client/server-based. We chose a client/server architecture because of the benefits that it offered us in the areas of performance (especially with the voice technology), cheat prevention, and centrally located statistics. The clients all run on Windows 95/98 and the servers run on Windows NT boxes, where we use Microsoft Chat services to do the intercommunication between our server processes. We also have a Microsoft web server running the community web pages, with a Microsoft SQL server maintaining the database. Our servers are at one location, our ISP Globalcenter, in Sunnyvale, California.

FIRETEAM

Multitude Inc.
Burlingame, California
(650) 685-2001
http://www.multitude.com or http://www.fireteam.com
Team Size: 14 full-time developers. Some number of contractors.
Release Date: December 1998
Target Platform: Windows 95/98
Budget: Approximately $2.5 million
Time in Development: Two and a half years
Tools: Microsoft Developer Studio 5.0, Microsoft SQL Server, Microsoft IIS, 3D Studio Max, Microsoft Interdev 6.0, Microsoft Chat Service, and Windows NT

FIRETEAM uses the Elemedia SX2.0 Voice Codec to do its voice compression and decompression. Multitude's proprietary software wraps around this voice codec and interfaces with the Windows sound system for both input and output. The game mixes multiple voices on the client side rather than the server side. Clients simply send voice packets to the server, the server then routes them on to the appropriate teammates. In the future, spectators or enemies will be able to listen in on the voice chatter. Our voice software handles both DirectSound and non-DirectSound drivers because some sound cards work with DirectSound in full duplex. Full duplex means recording from microphone and playing sound at the same time.

Who Worked on Fireteam

Ned Lerner and I started Multitude and began working on the original project in April 1996. The development team grew gradually over the course of the project. Jim Morris was brought on during the summer of 1996 to be the chief technical officer, and his first project was to develop the voice technology. Alan Murphy was brought on to provide art for the prototype and eventually was named art director. Conroy Lee, Harvey Smith, and Harry Schaffer were brought on in early 1997 to help take FIRETEAM from a prototype to the real game that we showed off at E3 1996. Bill Money, James Poelke, and David Reese came on in late 1997. The team has a very diverse group of products to its collective credit. Lerner and Morris were two of the first people to work on 3D in the game industry. Murphy's art credits include GALAXIAN, PAC-MAN, DEFENDER, TAZ, and X-MEN. The others have worked on games such as SYSTEM SHOCK, TERRA NOVA, MAGIC SCHOOL BUS, ULTIMA VIII, and FRONT PAGE SPORTS: BASEBALL.

What Went Right

1. COMBINING TEAM PLAY AND VOICE TOGETHER. FIRETEAM's design focus was on team play. Just as we've seen in team-oriented sports, the cooperative nature of playing as a member of a team has proven to be a very addicting and powerful gaming design. FIRETEAM's cooperative nature was a symbiosis of our voice technology and team play design. We needed to give people a reason to talk to strangers on the Internet. Team play

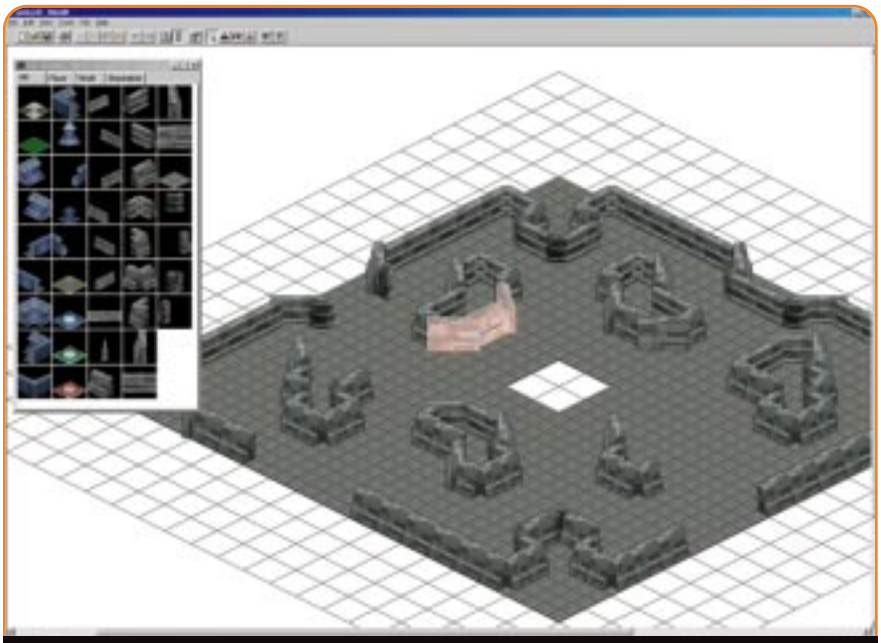
was that reason; it gives people the ability to say "Watch out behind you!" or "Good job!" Teammates can share the joys of victory or the agonies of defeat. Because there is no button to push to transmit your voice (it transmits automatically when you talk), players can hear the spontaneity of teammates yelling and laughing. Emotion comes across very clearly with voice and is definitely preferable to typing in ALL CAPS or emoticons. The ease of vocal interaction brings the team together.

In a fast-paced tactical game such as FIRETEAM, players don't have time to coordinate movements with the keyboard. Without voice, you limit team communication to select macro keys (or players who can type very fast). In FIRETEAM's Basetag scenario, for example, teammates protecting the base can give instant information on where the enemy is making its attack. Over the course of their lives, people have already learned how to talk; it's an interface they understand. Vocal communication doesn't require a key card list for communication hotkeys, just a microphone to talk into.

2. DESIGNING THE PROJECT AROUND CONSTRAINTS. Multitude was founded to do a game for the Internet. The Internet offers many problems that we had to solve in order to make a fun game. The biggest technical problem was Internet latency. Fundamentally, latency causes each player to have something different on his or her screen, so there is always a delay between a player performing an action and the other players seeing that action carried out.

For example, we decided early on that we wanted the game to respond as quickly as possible. When you shoot at something during a FIRETEAM game, you'll see instantly whether or not you hit your target. The actual damage will take a small amount of time to be applied to the target. So it's possible that you'll see someone get shot, walk a bit, and then die. The game cannot provide a perfect view of the world to each player; that's not possible given the limitations of the Internet. So we decided not to show players their opponents' health. If you can see that your opponent has only a sliver of health and that one shot could kill him or her, then you would expect that player to die instantly with the next shot you made. By hiding opponents health from our players, we hide the perception of lag.

A project's constraints can also be exploited to the project's benefits. Because the constraint was that we



This is Tile Edit, our basic world builder. We quickly prototype the physical layout of the maps with this tool. It's very easy to move walls around to achieve the right game balance.

needed to be on the Internet, we created the community web pages to give players the ability to look at their statistics and create FIRETEAM Companies. We wanted a strong community for FIRETEAM, and the community web pages were an easy way for people to have an identity in this community and to join a group of other players.

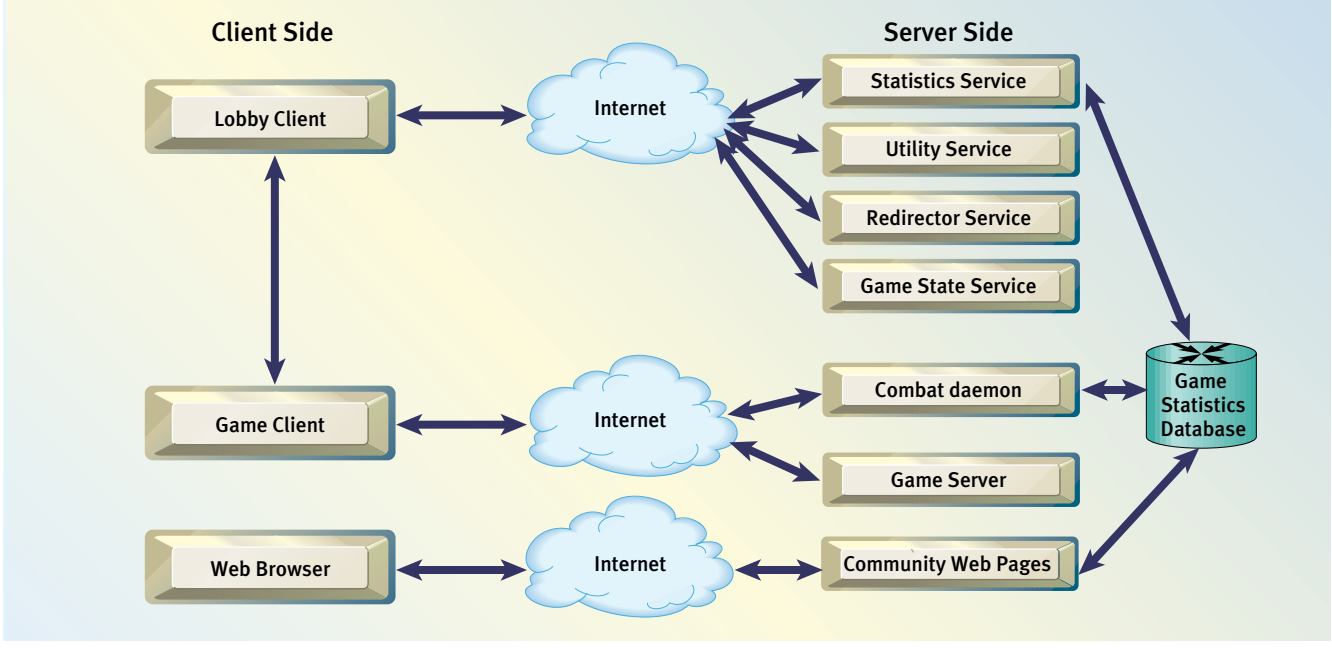
3. SPENDING SUFFICIENT TIME TO DEVELOP TOOLS. We used several proprietary tools to create FIRETEAM's game environments. Early on, we spent a lot of time building easy-to-use tools that allowed us to create content rapidly. Our internal testers used these tools to create new arenas for FIRETEAM. And because FIRETEAM is an online game, we were able to test new maps very quickly with our beta testers.

With an online game, game balance is crucial. Players will find any competitive edge and map imbalance they can and exploit it. Especially in an online game with a lobby, word of cheats or advantages spreads very fast. Your tools must allow you to tweak your maps, so you can quickly fix any small problems. Many of our maps changed during the course of testing as our testers would point out weaknesses that they found.

I can recall a particular controversy over whether Gunball (a FIRETEAM scenario similar to combat football) was balanced enough. Many of the advanced players were complaining that Gunball's offense was too hard. Using our Tile Edit tool, we quickly created a few maps with two endzones for each team (Gunball maps normally only have one endzone). Through testing the new maps, we discovered some of the problems with Gunball were unrelated to the maps themselves, but that the offense simply had a disadvantage when trying to score. So instead of redoing all of our map designs, we tuned the Gunball game by giving the Gunball carrier a protective drone.

An added bonus of easy-to-use tools is that you can make them available to the public and let your players customize the game and create their own content. We haven't yet taken that last

FIGURE 1. *FIRETEAM's technical architecture.*



We take the output from 3D Studio Max (with our plug-ins) and, with our proprietary ZHMPView tool, convert the files to a format that FIRETEAM can read.

step, because we're not sure how we want to store the maps on our servers and present them to the community.

4. MANAGING RISK: VOICE TECHNOLOGY. The biggest risk in developing FIRETEAM has been the voice technology. Many smart people initially said it was impossible, but we knew that the game's design objective was a cooperative team game, and voice was very important to accomplishing the goal. So FIRETEAM's first technical project was to determine whether or not voice on the Internet was even possible. Once we had the technology running over the Internet, we still faced the possibility that it wouldn't work with the wide spectrum of sound cards in the market.

We tried to minimize the problems that voice would cause by providing users with a tool that would configure the sound card and microphone during installation. Multitude was very aware (almost scared) of the fact that FIRETEAM would represent most users' first use of voice technology on their computers. So we had to make sure that it worked on as many sound cards as possible and that it was very easy to use. We eventually released FIRETEAM as two executables — one for systems with DirectSound and one for systems without it.

One feature that we explicitly did not put into the game was the ability for players to talk to their opponents. We wanted players' first experience with voice to be a positive one. We didn't think a 12-year old telling you where to put your gun in his shrieking voice would convince people that voice is a wonderful addition to gaming. Similarly, people asked for the ability to eavesdrop or steal another team's radio and listen to the other team. This feature would impel team members not to talk if they believed they were being monitored. These types of behavior would weaken the voice feature.

5. PROMOTING COMMUNITY. If you're going to design an online game, you cannot ignore the community. Any online game, from FIRETEAM to Poker on AOL to ULTIMA ONLINE, will have a community because the players will be able to communicate with each other. Online game developers should take advantage the fact that their product inherently has a community. Most online games go through alpha and beta online tests mostly to test the software, but few deliberately create or test the community aspects of a product. Players are not only a source of revenue for a project, but they are a feature of your game. In an online environment, the players' game experiences are dictated by their teammates and the opponents against whom they play. You want the players to follow guidelines and really care about the game and the community. If your population is full of a bunch of player killers, then that's the experience that the players will get.

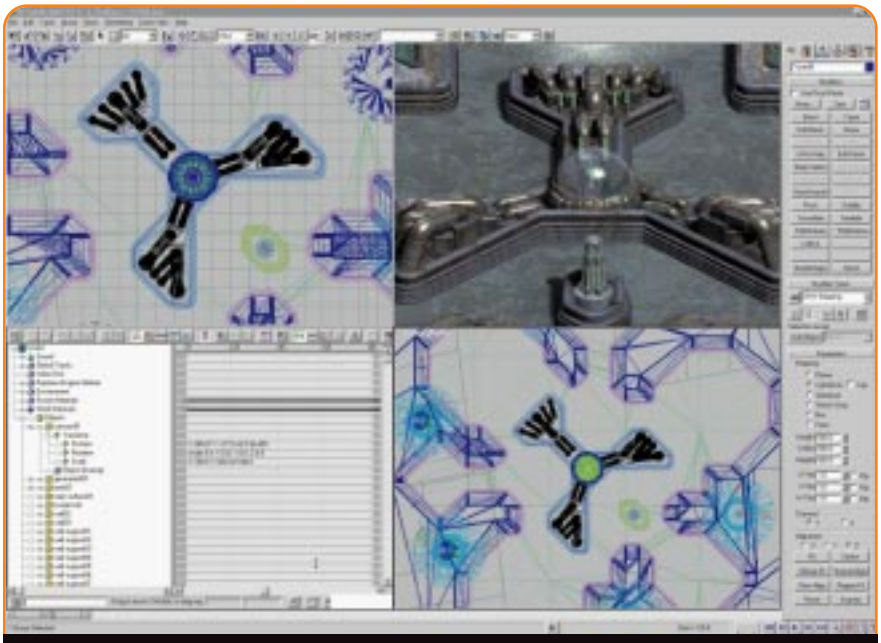
Multitude succeeded in developing community-enabling tools. We spent significant time and discussion on our lobby and community web pages. Given FIRETEAM's team nature, we wanted players to feel a sense of belonging so that they would want to save each other's lives. The FIRETEAM product is not just the game itself. The game is an important piece of the FIRETEAM experience, but it's only a piece. The community plays a large part of the whole experience.

What Went Wrong

1. MISJUDGING MARKET CONDITIONS. When Multitude was founded in April 1996, there was a lot of buzz in the online game space. Mpath and Ten had big plans. ULTIMA ONLINE was about to go through testing. We believed that an online-only game, sold directly to customers via the Internet, would be acceptable to the market when we eventually shipped. What we've discovered is that the online game market has not matured to the level that we expected. Very few online-only games have been released, with ULTIMA ONLINE being the only clear success. We made two decisions early on that should have been reexamined when it became clear that customer acceptance of an online-only game was not a forgone conclusion.

FIRETEAM would have been more willingly accepted by the market if it contained some artificial intelligence (AI). With such an implementation, players could practice using the interface by themselves and, more importantly, players could practice as a team against AIs. With computer-controlled opponents in place, players could play offline or possibly on a LAN against computers opponents. Many users are intimidated by having to learn a new game while playing against other more experienced human players. AIs would have helped ease players into the online-only part of the game, providing a feature that many players expect to find in games today.

FIRETEAM also should have had a demo available on day one. As an online game, providing a demo pre-



A 3D Studio Max layout of one of the maps. Our artists take the game-tested layout from Tile Edit to create backgrounds for each map.

sented an interesting problem because of the server issues. With a traditional game, developers can hand out a million demo disks and never think about the problems their users might experience. If we gave out a million demo disks, then we would need to have enough servers to support all those people that actually play the demo. We didn't create an infrastructure to support a demo mode of FIRETEAM. FIRETEAM is a new type of game — people aren't yet accustomed to online tactical team games with voice technology. We should have made some extra promotional effort to get potential users to make that initial leap and try out the game.

2. MANAGING CONTRACTORS. FIRETEAM was a large and extremely challenging project. We had to look outside of our own company for help with certain parts of the project. Our mistake was in assuming that these experts held the same priorities as the rest of the development team. These groups have their own objectives and aren't expected to understand the big picture or know how to create fun games. We realized that when working with contractors, we needed to give those contractors a very precise and clear specification. Because a good game design evolves over the course of a project, a project manager must constantly make certain that the contractors are following the latest version of the specification.

As we were developing FIRETEAM, our attention was also focused on growing our new company. We found that it was easy to forget what the contractors were doing and how they fit into the project. We made the naïve assumption that they would be willing to work with an evolving specification. However, when a project is fix-bid, an external developer will only do so much tuning and reworking of code before he or she starts charging you for it. If you don't manage this relationship closely, these costs add up very quickly. For example, the cost for developing the community web pages doubled from the original quote because the design evolved. The final version of the community web pages was great, but a more thoughtful initial design specification and better management of the process would have saved Multitude significant money and time.

3. INTERNET TECHNICAL ISSUES. The Internet poses significant problems for developers. Although we did our best in designing the game around the limitations of the Internet, we did have some technical problems. We originally designed FIRETEAM around TCP/IP because it's a reliable transport protocol for network traffic. However, the reliability comes at a very high cost: retransmission times. If a packet is lost on the Internet (which happens a lot), it takes some time for the machines on both ends to realize this and resend the data. TCP/IP guarantees that all packets are in order; therefore, all of the packets after the lost packet will be delayed until the lost packet is sent again. In a fast-paced game such as FIRETEAM, lost packets can really cause problems. As soon as we started doing real Internet tests, we realized that we needed to start sending some packets unreliably via UDP. These packets could get lost, be out of order, or even duplicated, but they wouldn't be delayed by other packets. We learned that different packets require different sets of reliability and timeliness, and that developers should use all the tools available to them, both TCP/IP and UDP. We initially labored under the idea that only one protocol should be used for the sake of simplicity, but it's best to use the appropriate tool for each job.

Packet loss and high ping times are simply part of the reality of dealing with the Internet. You do your best to deal with these issues, but they'll still cause you endless headaches as routers over which you have no control go down throughout the country. Many online games come with a little utility that does a trace on the route the packets take between a player's machine and the servers. The information that the utility returns can help the player and his or her ISP determine where the bad connection is along that route. Developers should be aware that while they cannot fix the Internet infrastructure, it's important to understand its limitations and deal with them as best they can.

4. SERVER SPAGHETTI. FIRETEAM is a very complicated project with many processes running on both the client and the server. Add in the complication of the Internet, and you can get one confusing mess (Figure 1 shows just how complicated the FIRETEAM architecture is). We tried to break our server components down into smaller, more manageable pieces, each with its own function. We hired some experts in various disciplines to help us better understand parts of the server technology that were new to us. Our mistake was in thinking that these experts could just come in and solve our problems. As we busied ourselves with other parts of the project, it was easy for us to

The home page for the community web pages. Players can access a wealth of information about their statistics or other players' statistics.

say to ourselves, "They know what they're doing." In the end, however, the development team needs to understand the whole picture and how the pieces really fit together. One of FIRETEAM's unique properties is that its server-side components run remotely at an ISP's facilities. In order to debug something as complicated as our server architecture remotely, our key programmers — not just the client/server experts —needed to understand the whole system.

One or two weeks spent planning and discussing the entire project with everyone involved will save you months down the road. The process of actually finishing and shipping a game is the hardest part of the development cycle; not many people actually know how to ship a game. During the final stage of the project, it's essential that the entire team understand all the pieces of the puzzle.

5. COPING WITH THE COMMUNITY. As I mentioned previously, when you create an online game, you need to embrace the community. At the same time, a direct connection with a community of testers who aren't 100 percent aware of your objectives is something that needs to be managed very carefully. The testers will always want something different. When is the last time you played a game and said, "This is perfect"? I've often said that even my favorite game would be better if it had feature X. Most beta testers are young people who have a lot of time on their hands; that's great for finding bugs, but it can also be a problem because some of them lack perspective. All players have an equal voice in the FIRETEAM lobby, so we had to watch over the lobby constantly because a few testers could ruin the fun for others, even to the point of instigating a mini online riot.

From what I can tell, some online game companies simply ignore their testers' constant demands. After the experience of developing FIRETEAM, I must admit that this is a possible solution, though not an optimal one. Many of us on the development team spent many hours justifying our design decisions in order to educate the testers on why we were doing things a certain way. While this education does make them better testers, it takes up a lot of time. And it's a dangerous black hole that you can be sucked into if you're not careful. I believe that the true balance is to pay attention to your community, but sometimes to sacrifice the battle in order to win the war. You should involve your intended community in the evolution of your game, but don't let it take over your design process or time.

Evolving Right Along

In building FIRETEAM, we as developers accomplished our goal of providing a complete online gaming experience with true team play, innovative voice technology, and extensive community building tools. The Internet offers brand new gaming experiences; game players can compete in ladders such as Battle.net or tournaments such as the PGL. Also, an online game lets players meet new friends with whom they can share true social gaming experiences.

However, the Internet introduces a lot of negatives to the gaming experience. Instead of lightning-fast LAN connections, players must now tolerate latency. Instead of a small group of friends, a player's opponents may be complete strangers who aren't polite and may even be cheaters. Because to the newness of this market, FIRETEAM may be ahead of its time. Or it may not have exactly hit the sweet spot that online multiplayer gaming should be. But, FIRETEAM has helped online game evolution along by demonstrating that voice technology does work and that team play and community are compelling elements that don't have to be accidental. ■

by Ernest Adams

Immortality for Game Developers

I was thinking in the shower the other day about the notion of immortality. It's among the most ancient of human fascinations, and a subject of philosophy and spiritual thought since before recorded history.

If you want to live forever, you have some options: spiritual immortality (religion), practical immortality (don't die), or virtual immortality (fame). The disadvantage of spiritual immortality, at least for the rationally-minded, is that there doesn't seem to be any evidence that it exists. And we're a long way, technologically or medically, from achieving practical immortality and conquering death. So, that leaves virtual immortality: fame, and the knowledge that you counted for something and will be remembered by those who follow you. How can I, as a game developer, be remembered?

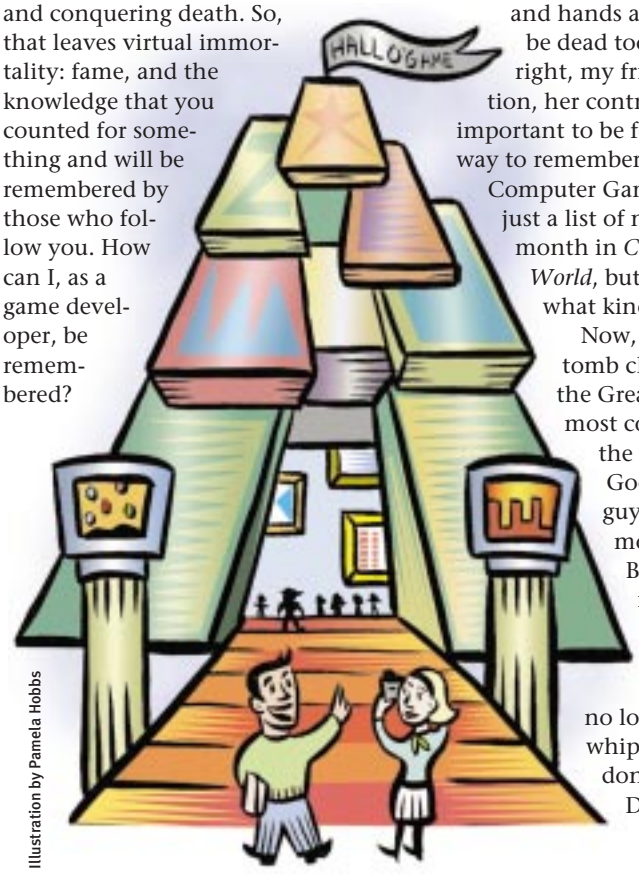


Illustration by Pamela Hobbs

Now, you might say, "So what? The vast majority of the world leaves no legacy. What entitles you to a monument?" I don't have an answer to that, except that I know that I want one. And not just for myself. There's someone else that I want people to remember as well.

Danielle Bunten Berry is dead. And in a few years the work of her heart and hands and mind are going to be dead too, and that is not right, my friends. Her imagination, her contribution, was too important to be forgotten. We need a way to remember her. We need a Computer Game Hall of Fame. Not just a list of names printed every month in *Computer Gaming World*, but a real memorial. But what kind?

Now, I have stood in the tomb chamber at the heart of the Great Pyramid. One of the most common reactions to the Great Pyramid is, "My God! What a ego that guy had, to build such a monument to himself." But there's nothing intrinsically evil or immoral about building monuments, even to yourself. We no longer have to use whips and slaves to get it done. Why shouldn't Dani get a pyramid, if we want one for her?

Well, pyramids are expensive, and they take up a lot of space. So, we turn to the question of leaving a legacy in memory, rather than stone. But the work of game developers suffers from a kind of technological decay that is not experienced by other artists. To illustrate this I want to quote Bruce Sterling, the science fiction author, from a speech he gave at the 1991 Computer Game Developers' Conference. He was talking about a hypothetical — and now not-so-hypothetical — device, the "electronic book," and he said:

"Now I'm the farthest thing from a Luddite ladies and gentlemen, but when I contemplate this particular technical marvel my author's blood runs cold. It's really hard for books to compete with multisensory media, with modern electronic media, and this is supposed to be the panacea for withering literature, but from the marrow of my bones I say get that little sarcophagus away from me. For God's sake don't put my books into the Thomas Edison kinetoscope. Don't put me into the stereograph, don't write me on the wax cylinder, don't tie my words and my thoughts to the fate of a piece of hardware, because hardware is even more mortal than I am, and I'm a hell of a lot more mortal than I care to be. Mortality is one good reason why I'm writing books in the first place. For God's sake don't make me keep pace with the hardware, because I'm not really in the business of keeping pace, I'm really in the business of marking place...

"You folks are dwelling in the very maelstrom of Permanent Technological Revolution. And that's a really cool place, but man, it's just not a good place to build monuments."

He's right, of course. Our work is as bright and as beautiful as the wildflowers of a Sierra mountain springtime... and just as ephemeral. Our games cannot serve, unaided, as our monument. When we die, we leave nothing to remember us by. We need something else.

Continued on page 71.

Ernest Adams has been a computer game developer for ten years and has a longstanding interest in the industry's professional culture. He was a founder of the Computer Game Developers' Association and is a frequent lecturer at the GDC.

Continued from page 72.

The Computer Game Hall of Fame would be a place where the great games are kept, and talked about, and studied for the wonder and truth that they contain. Above all, it would be a place where their designers are honored. It should consist of two things: First, a permanent site on the World Wide Web (which, in my opinion, is soon to be the collective cultural memory of mankind). Second, a physical place. A building, a museum — an arcade if nothing better — where people can go and admire, play, learn, and remember.

Now some will say, "A museum about outdated video games? Pathetic." But consider this: I work on a game about professional football. There's nothing very world-shaking about professional football. It doesn't change the fate of the human race. Professional football is about the exercise of athletic skill for the purpose of excitement and entertainment. Excitement and entertainment is our business too. If professional football can have a Hall of Fame, then by God, we're entitled to one.

Who's going to build it? I don't know. I don't have the time. I don't have the money. But it needs to be done, so that our great works can live on. They can be remembered, but only if we choose to remember them. ■