gd

GAME DEVELOPER MAGAZINE

OCTOBER 1998

UBM Tech

It's First and Goal for Fantasy Sports

This fall, as the leaves turn shades of orange and the days grow shorter, one of the largest, most massively multiplayer games picks up steam and sucks in participants. It's a role-playing game that draws tens of thousands (gads, probably more) of players, and if my predictions are right, it will be one of the most popular attractions on the eventual TV set-top box. I'm talking about fantasy football leagues.

It's taken quite a bit of time for me to accept the fact that fantasy league sports (there are also fantasy leagues for baseball, hockey, and perhaps even pro beach volleyball for all I know) belong in the same category as "traditional" computer games. But the more I thought about fantasy leagues, the more I came to see them as kin to many other popular video and computer games.

As the owner of a sports team, your role is similar to that of most RPG players: put together a balanced, effective team based on attributes such as speed, strength, and toughness, and pit the team against opponents. The select players you designate as "active" for that week earn you points based on the actual statistics they earned in NFL games. (Scoring systems in fantasy leagues are often pretty complex.) Based on your weekly points, you either triumph over your opponent and improve your record, or not, and the teams with the best records at the end of the season go on to the playoffs and the eventual Superbowl. There's far more to it than this, but as you can see, assuming the role of team manager/club president/talent scout during the season holds the same addictive lure as any RPG. When you factor in the huge, rabid market for these pay-for-play leagues (recent estimates pegged the fantasy sports market at eight to ten million people), you begin to see the potential.

One company that's quickly become a force in the fantasy sports game genre is New York-based Small World Sports (http://sports.smallworld.com). SWS, a small company that started in a dingy apartment four years ago, looks to be one of this year's game development success stories. Unlike the traditional studio's royalty revenue model, SWS has two revenue streams: a two-year licensing agreement to develop more than 40 online games for CNN/SI (http://baseball.cnnsi.com), plus revenue from banner advertising displayed on the CNN/SI game's web pages, which garner 50 million page views per month. Surprisingly, and in contrast to most commercial fantasy leagues, some of the CNN/SI leagues are free for participants and offer cash prizes for winners. These are the guppy leagues which, hopefully, entice the most enthusiastic players to join the premiere leagues for $15.

With such tremendous success attracting players to the site and bringing in advertising revenue, and armed with the knowledge that the fantasy sports market is growing approximately 25 percent per year, SWS and CNN/SI have made plans to develop four games each for the NFL, NBA, and NHL seasons, including mid-season and playoff games, and games with different styles and scoring systems.

More traditional game developers have begun staking their claims in this market, too. Electronic Arts, the king of sports games, recently announced a new web site called "The EA Sports Edge" (www.easports.com/99/easportsedge/). The site offers recommendations on fantasy football league drafts, trades, and so on, for about $20 per season. I expect others will follow. As Chris Berman might say, computer-based fantasy sports games could go…all…the…way…

Heeere's Mel

This month, I'd like to welcome Mel Guymon to *Game Developer* as the new Artist's View columnist. Mel penned August's character animation tools article. He's a 3D artist who's recently worked on high-profile projects for Zombie, Eidos, Psygnosis, and others, and he brings a wealth of technical knowledge and creative experience to bear in the column. Welcome, Mel. ■

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

EDITOR IN CHIEF Alex Dunne *adunne@sirius.com*
MANAGING EDITOR Tor D. Berg *tdberg@sirius.com*
DEPARTMENTS EDITOR Wesley Hall *whall@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, Noah Falstein, Brian Hook, Susan Lee-Merrow, Mark Miller

COVER IMAGE Epic MegaGames

PUBLISHER Cynthia A. Blair *cblair@mfi.com*
WESTERN REGIONAL SALES MANAGER Alicia Langer (415) 905-2156 *alanger@mfi.com*
EASTERN REGIONAL SALES MANAGER Kim Love (415) 905-2175 *klove@mfi.com*
SALES ASSOCIATE Ayrien Houchin (415) 905-2788 *ahouchin@mfi.com*

MARKETING MANAGER Susan McDonald
AD. PRODUCTION COORDINATOR Dave Perrotti
DIRECTOR OF PRODUCTION Andrew A. Mickus
VICE PRESIDENT/CIRCULATION Jerry M. Okabe
ASST. CIRCULATION DIRECTOR Mike Poplardo
CIRCULATION MANAGER Stephanie Blake
CIRCULATION ASSISTANT Kausha Jackson-Craine
NEWSSTAND ANALYST Joyce Gorsuch
REPRINTS Stella Valdez (916) 983-6971

Miller Freeman
A United News & Media publication

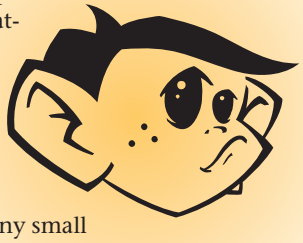
CEO-MILLER FREEMAN GLOBAL Tony Tillin
CHAIRMAN-MILLER FREEMAN INC. Marshall W. Freeman
PRESIDENT/COO Donald A. Pazour
SENIOR VICE PRESIDENT/CFO Warren "Andy" Ambrose
SENIOR VICE PRESIDENTS H. Ted Bahr, Darrell Denny, David Nussbaum, Galen A. Poss, Wini D. Ragus, Regina Starr Ridley
VICE PRESIDENT/PRODUCTION Andrew A. Mickus
VICE PRESIDENT/CIRCULATION Jerry M. Okabe
VICE PRESIDENT/GROUP DIRECTOR KoAnn Vikören
SENIOR VICE PRESIDENT/SYSTEMS AND SOFTWARE DIVISION Regina Starr Ridley

www.gdmag.com

Indie Game Festival

With regards to Alex Dunne's September editorial ("Where's Our Sundance?"), this is a major issue for developers, and proves to me yet again that while we want to make incredibly cool games, publishers want to make money with games they *know* will sell well because they resemble previous best-sellers.

One of the problems with creating an indie festival is that the games would need to be created to get there, and many small developers can't create the games without the funding. After all, making a top shelf game these days takes more than $1 million — not exactly chump change. Of course, the spirit of indie films is, as Dunne said, smaller budgets, so this may not be a true obstacle.



Perhaps in the first round or so, developers could submit games that they created that were market "flops," yet still garnered outstanding critical acclaim. God knows, there are enough of them to keep the festival full for several years. This could be an excellent extension of the GDC functions, as all the right people will be in the right place. E3 simply has too much going on (especially thanks to the massive budgets for booths like Sega).

Anyway, this is clearly an idea whose time has come.

Patricia Pizer
CogniToy

Mourning Newfire

Thank you for August's interesting perspective on Newfire. It's good to know that some people recognized the importance of my former company. As a former Newfire employee, I viewed my coworkers, for the better part of a year, as my family. It seems that whenever I wear a Newfire shirt, someone approaches me and asks about the company. I give much of this recognition to Harry Vitelli, who was the VP of marketing for Newfire.

The demise of the company is especially frustrating because I feel that our latest efforts were technologically superior to most titles out in the market. With a crack engineering team making technological improvements to the engine on a weekly basis, who could go wrong? Yet, apparently, there were still questions about our direction. Some people in the industry felt we had an identity crisis with respect to our customers. The question was often raised, "Is Newfire going after consumers or developers?". Some wondered whether people would pay a few hundred dollars for a superior QuakeC-type development environment. Still others felt we cheapened ourselves by selling short to small developers.

Your two cents. E-mail us at gdmag@mfi.com. Or write to *Game Developer*, 600 Harrison Street, San Francisco, CA 94107.

And yes, a number of us at Newfire did want to develop a game with the technology. Many people just can't believe that while we won so many awards, we had little sales, our funding cut, and as a result, folded a few months later. Thank you for your bittersweet retrospective.

Former Newfire employee
(name withheld by request)

Alex Dunne makes some good points about the nature of the industry in his article, "Requiem for a Game Engine Company," but the reason that Newfire went under had less to do with the nature of the industry, and more to do with selecting VRML as the foundation for a 3D engine. It's a challenging medium in which to work. Hats off to Newfire for trying to go the standards route, but their decision to follow the rocky path of VRML/Java led to a hard sell to offline developers who want C++ and cutting-edge engines.

In regards to their online run-time (Heat/Torch) and tool ambitions, their competition was huge: Cosmo and Intervista — free VRML plug-ins/3D engines with big backers. It was difficult to compete with free products (and huge marketing budgets).

"Credibility issue," "competition with in-house solutions," "lack of proof of concept," and pricing are all valid challenges to point to in the industry — but not the best explanation for Newfire's unfortunate death. Newfire would have been better off creating a custom 3D engine first (no standards) with tools/plug-ins to ease the art path and coding, possibly addressing the issue of standards down the road.

Galen Tingle
via e-mail

After reading your August GamePlan, "Requiem For a Game Engine Company," I was moved and distressed. I'm an artist who is interested in game development and game graphics. I needed a game engine that was affordable, and one that could offer me the ability to create a small demo to pass around to game development companies. Eventually, I decided on the Acknex engine by Conitec. Though I am still learning to use it, I am thus far very satisfied with it. I'm relieved that I could find a tool that would allow me to control a game environment without going into years of training to learn source code. Premade game engines should be encouraged among independent developers not only because of the time and money saved, but also because it allows fresh new thinking into the industry by encouraging talented people outside to explore it. It's the industry's loss if they can't warm up to the idea of a packaged game engine. I believe that in the future those who learn to use this kind of tool creatively will have a huge advantage over those who spend extra (and often unnecessary) time churning out code. A little creativity and a pre-made game engine could accomplish the task as well, if not better.

Leon Wisocki
One Angry Goldfish Productions

Motivate's New Pricing

Thanks for Dan Teven's in-depth review of our Motivate game authoring software in your August issue. We are happy to say that we've addressed his key concern — pricing.

We have introduced a $3,500 per seat pricing model that enables users to purchase a perpetual license for the Motivate Development Tools and SDK.

David Pritchard
CEO, The Motion Factory Inc.

INDUSTRY WATCH

by Alex Dunne

BLACK ISLE. Interplay launched a new RPG division, Black Isle Studios, which will release BALDUR'S GATE and FALLOUT 2 later this year. Led by FALLOUT veteran Feargus Urquhart, Black Isle Studios also employs experienced RPG designers Guido Henkel and Zeb Cook. In other Interplay news, the company signed an exclusive deal with 3000AD, Inc. to distribute Derek Smart's BATTLECRUISER 3000 AD 2.0. The original BATTLECRUISER 3000 AD was released by Take Two in 1996, and quickly became embroiled in controversy due to rampant bugs, incomplete documentation, and a laundry list of other problems. Interplay will release the version 2.0 through its "value products division" for about $20.

SUPPLY VS DEMAND. Chip maker 3Dlabs filed suit against one of its own customers, charging STB Systems with breach of contract. 3Dlabs claims that, in violation of contracts between the two companies for the sale and purchase of chipsets, STB failed to pay 3Dlabs over one million dollars. As we go to press, STB has not issued any statement regarding the suit.

IN A DEFT MOVE, Hasbro acquired MicroProse in a deal valued at approximately $70 million. Hasbro formerly employed no internal developers, but this acquisition brings aboard a deep reserve of development experience. The deal also strengthens Hasbro's overseas distribution, thanks to MicroProse's strong distribution network in Europe. In a separate announcements, MicroProse released its first fiscal quarter results, and reported a loss of $7.8 million on revenues of $12.2 million. In the past year, the company burned through almost 90 percent of its cash, going from $41.2 one year ago to just $4.3 million. The acquisition came at a good time.

HOLD ONTO YOUR HATS, FOLKS, because Mattel is releasing a game based on John Gray's best-selling book, the infa-

Physics for All

ReelMotion's user interface.

MOTIONAL REALMS AND LATERAL LOGIC have both developed tools to assist with game physics. Motional Realms has just released ReelMotion for Windows. ReelMotion is a new stand-alone software program that uses physics to animate vehicles and rigid-body objects. Import rugged terrains, fly helicopters, roll cars, add external forces (think missle explosions), and crash into inanimate objects (with varying resistances, of course). The tool is ideal for animating space and aerial dogfights, accident recreations, and cut scenes, among other applications. As the animator, you use a mouse or joystick to drive or fly a vehicle through a scene. ReelMotion's real-time OpenGL display will then present you with a 3D view of the action. Reel Motion sells for $795 and is now available for computers running MacOS, Windows 95 or 98, and Windows NT.

Lateral Logic has also recently released a physics-based product, the Lateral Collision Engine (LCE), a comprehensive collision detection system designed for interactive graphical environments. Instead of creating application specific solutions for collision detection, developers can now integrate collision detection capabilities right into their game. Lateral Logic claims that the product may be adapted to your application within only three to five days (as opposed to the weeks or months this usually takes). LCE provides game developers with exact penetration regions, contact points, and collision normal vectors for every collision. Another object-oriented product from Lateral Logic, the Lateral Dynamics Engine (LDE), is due out in the Fall of this year. The LDE has three main modules, a modeler, run-time solver, and an API.

LCE Pro began shipping on September 15, 1998. The LCE Pro developer toolkit will be $3,500 for one nodelocked run-time license. Additional licensesare $1,500. Volume pricing is available.

- Motional Realms, LLC
  Reston, Va.
  (703) 860-0714
  http://www.reelmotion.com

- Lateral Logic Inc.
  Montreal, Canada
  (514) 287-1166
  http://www.llogic.com

Softimage Ships 3.8

SOFTIMAGE announced at SIGGRAPH the shipment of Softimage|3D 3.8, the latest version of its 3D animation tool. New features in 3.8 provide better data management and more efficient game production. They include an animation sequencer, interactive polygon reduction, resolution-independent enveloping tools, and 3D Paint (which-will allow you to paint directly on all geometry types). This version also increases texture-mode performance by up to 40 percent.

Version 3.8 is available, free of charge, to all Softimage|3D customers under a maintenance contract. For new clients, the base rate begins at $7,995.

- Softimage, Inc.
  Montreal, Canada.
  (514) 845-1636
  http://www.softimage.com

ASTS

OF GAME DEVELOPMENT

4D Paint 2.0

RIGHT HEMISPHERE LTD. has announced 4D Paint 2.0, due to ship in Q4 of 1998.

Known for its natural media paint tools, 4D Paint caters to professional 2D artists who wish to make the transition to interactive 3D environments without learning complex modeling techniques. The 3D paint system for the NT platform has made improvements which CEO Mark Thomas describes as "a quantum leap." 3D Rendering quality and speed are both juiced up, and a full quality raytracing render option has also been added to generate high quality stills. 4D Paint 2.0 will have all the same integration with 3D modeling systems, plus a new UV mapping optimization system named "Mercator." Further, the tool will now support Adobe Photoshop plug-ins and a bi-directional interface to Photoshop.

4D Paint supports Windows NT on both Intel and Alpha platforms, and Windows 95 or 98. It also supports a Wacom or compatible pressure-sensitive tablet. Pricing is expected to be $695 for the Intel platform and $895 for the Alpha version. The plug-in interface will be priced separately.

This sphere from 4D Paint 2.0 has its color, bump and shininess rendered. This is all done in real-time,and the sphere can rotate and pan, too.

■ Right Hemisphere Ltd.
Auckland, New Zealand
+64 (9) 309-3204
http://www.righthemisphere.com

Animate the Face

TECHIMAGE recently unveiled Artiface, a new tool which allows you to create 3D facial animations (both realistic and fanciful) without turning to traditional key-framing.

Artiface captures the facial movements of an actor in 2D video, extracts 3D motion data via image analysis, and transfers the movements to any 3D model by applying an anatomical muscle structure. Because of its precision, Artiface can also perform accurate lip synching. The tool requires no specialized image markers or motion capture hardware equipment. Only one video camera is required to record the motion of the actor's face. The software includes specialized sculpturing and enhancing tools to produce special effects.

Artiface supports SGI workstations running Softimage|3D and Alias|Wavefront PowerAnimator 3D packages. It sells for $7,995 for a single nodelocked version, and for $9,995 for a floating network license.

■ TechImage, Ltd.
Herzelia, Israel
972 (3) 673-4591
http://www.techimage.co.il

Artiface translates an actor's motions onto a 3D model.

mous *Men Are From Mars, Women Are From Venus.* The game, named after the book, is Mattel's "flagship adult game" and "is a great vehicle to talk about relationships in a casual setting," according to Gray. Since you're undoubtedly dying to know just how they managed to squeeze a game out of this gem, I reluctantly offer the following. Two teams, men (Mars) and women (Venus) play against each other. Players can be couples or singles. Starting on their respective planets, players answer questions about how they think, what they like and for what they wish. Teams advance on the game board path when they guess correctly how others will answer. The first team to reach Earth wins. Of course, like any feel-good game, there are no right or wrong answers. Yikes, I need to shower now.

NINTENDO announced a barrage of interesting marketing programs recently. The company signed a $17 million deal to promote Pokemon monsters with Kentucky Fried Chicken meals, snack-maker Keebler is going to market BANJO-KAZOOIE with its foods, and apparel manufacturer Tommy Hilfiger will put N64 kiosks in almost 1,000 mall department stores where it sells clothes, allowing customers to play titles such as 1080 SNOWBOARDING and F-1 WORLD GRAND PRIX.

3DFX SETTLED its year-old lawsuit against Sega, Sega of America, NEC, and VideoLogic. An agreement was reached behind closed doors to drop the $200+ million claim, which was originally filed after Sega terminated a contract with 3Dfx to develop graphics hardware for the Dreamcast console. In the suit, 3Dfx had claimed breach of contract and threatened misappropriation of trade secrets.

LOSS AND GAIN. 3DO reported a narrower-than-expected loss for its first fiscal quarter ending June 30, buoyed by strong sales of its MIGHT AND MAGIC IV and ARMY MEN titles. 3DO's net loss came to $2.6 million, compared to a profit of $21.2 million for the same period last year. The good news for the company is that its software revenues were far ahead of last year, reaching $9.5 million, compared to $2.3 million a year earlier.

Taking a Break for SIGGRAPH

SIGGRAPH time has come and gone again, so I thought I would update readers on the state of technology available for real-time 3D developers. Walking the exposition floor, I looked into where companies are going with real-time 3D.

Inverse Kinematics Solutions

Last month, I left off creating an inverse kinematics system. I thought I would take a look at the programming libraries available to handle such a task. You certainly will get the most flexibility by creating your own IK system, but given manpower, budget, and deadline constraints, this isn't always possible. As it turns out, a couple of companies with a great deal of experience in robotics research are ready to do the work for you.

Motion Factory's Motivate system has a very sophisticated solution for game developers who want to add intelligence to their games. Motivate is a finite-state-machine-driven animation system that can be programmed to perform complex tasks. The system has an internal 2D path planning and hierarchical inverse kinematics solution that brings life to the world. You can bring in hierarchical characters, give their joints degree-of-freedom restrictions, and then keyframe animate them. The tools are very polished. Motivate's programmers have clearly taken the time to simplify their tools and make them very robust. Most game programmers never have time to clean up the production tools and make them nearly this easy to use. The Motion Factory just announced a perpetual development license at $3,500 per seat. This will enable you to prototype and test out your game without paying the full license fee. Once the game ships, the run-time distribution fee of $50,000 still applies. With the development license, however, you can try out your ideas, and then make the decision whether you want to take the time to develop your own technology or simply buy the license. Check out the more complete product review of Motivate in the August 1998 issue of *Game Developer* for more information.

Katrix also provides inverse kinematics technology. The company has focused primarily on driving virtual characters around the desktop for web pages. However, the libraries they have created for inverse kinematics would also work well in a real-time game environment. Katrix makes licensing arrangements on an individual basis.

MultiResolution Modeling

A hot topic at the past two SIGGRAPH conferences, and at the CGDC this year, is the issue of automatic generation of levels of detail for 3D polygonal models. A couple of different companies provide commercial solutions to the multiresolution modeling issue. If you recall from my August 1998 *Game Developer* column ("Looking Forward with a Backward Glance at the CGDC"), multiresolution models can be used to create different levels of detail on-the-fly. You can accomplish this by dynamically increasing or reducing the polygon count in the model as needed to make the object highly detailed and to keep the frame rate of the game constant.

In August, I mentioned that MetaCreations had teamed up with Intel to create a toolset and API for creating and displaying these multiresolution models. This technology has been licensed by Microsoft and was being demonstrated in their ChromeEffects web technology. The specification for the Metastream "open" file format was not ready at SIGGRAPH, but was expected to be released shortly. Currently, MetaCreations has announced support for 3D Studio MAX R2, as well as their own products, Infini-D and Ray Dream Studio. Use of an open format for multiresolution models would create a convenient production pipeline for game producers — but it would have to handle textures, hierarchies, and animation as well as geometry to be very useful. We will have to see if the format is robust enough to fit the needs of developers. I know one thing that has been sorely lacking in 3D production is a method of exchanging animated and textured objects between graphics tools. In order to gain wider acceptance, either MetaCreations or some third party would need to create import/export plug-ins for the other mainstream 3D programs used in game production. MetaCreations' decision to release the reader code as well as the file format publicly will certainly aid developers in getting these models into their own production tools and game engines.

MetaCreations is not the only technology company working on this issue. Sven Technologies has a continuous level of detail solution for game developers called MRG. They focus exclusively on high-level real-time 3D performance. This allows them to work one-on-one with their clients to tune the technology to suit the individual developer's needs. They take a unique approach in providing two different libraries to support the technology. MRGPlay is the run-time library that manages playback of the models on a triangle level. This way, it's renderer independent and will work with any

When not vacationing in the humid swamps of the southern United States, Jeff actually runs a small real-time 3D graphics company called Darwin 3D. Feel free to send him ideas of what he should really be doing with his life at jeffl@darwin3d.com.

triangle-based 3D rendering system. The compression routines also work strictly on the polygon edge connectivity, leaving the actual vertex locations alone. This means that any form of vertex animation or deformation is completely compatible with their polygon reduction algorithm, as long as it doesn't change vertex ordering. If your game engine requires vertex-level animation, the MRG system offers an advantage.

The second API in support of the MRG technology is MRGAuthor. While Sven provides a tool to create MRG models from standard 3D files, direct control of polygon reduction is limited. However, the MRGAuthor library gives the programmer access to and control of the algorithm that processes the 3D model and creates the MRG data. This API will make it quite easy to integrate the MRG compressor into custom production tools. I believe that allowing developers to control the creation process is critical to creating realistic multiresolution models.

Licensing for the Sven MRG libraries is $15,000 plus a $5,000 annual service contract for royalty-free single-title usage. Site licenses and royalty-based schemes will be discussed on a case by case basis. MetaCreations licensing is handled on an individual basis, so no standard pricing is available

Deformable Single-Mesh Animation

Many of you who read my column in the May 1998 issue of *Game Developer* ("Skin Them Bones: Game Programming for the Web Generation") sent me e-mail asking what 3D modeling packages besides Softimage support weighted mesh deformation. I took the opportunity at SIGGRAPH to talk to the vendors and see what was available.

Many of the major 3D modeling programs now support a form of weighted mesh skeletal deformation. It even looks as though this data is available to game developers directly either via an SDK or a public file format.

On the high end, Maya from Alias|Wavefront supports a sophisticated system of skeletal deformation. Through the Maya Artisan package, it's even possible to paint weight values through the 3D paint interface. This is an elegant and efficient way to set the weight values. You can then extract the data through the DevKit that comes with Maya.

N-World from Nichimen also supports many levels of deformation. In addition to weighted envelope deformation, you can manipulate vertices directly through the animation. This provides a form of interpolated shape animation on top of the skeletal deformation. The data is available to developers in the Game Exchange 2.0 file format. The specification for this format is publicly available from Nichimen. Just in time for SIGGRAPH, Nichimen announced that Alias|Wavefront would also be supporting Game Exchange 2.0 in Maya.

By far, most of the e-mail I received on skeletal deformations contained questions about the ability to create single-skin meshes in 3D Studio MAX. In Character Studio R1, it was possible only to assign vertices to a single bone. This didn't look nearly as good as fully weighted skins. Luckily for us, Kinetix has released Character Studio R2. This new release includes a new version of Physique that uses weighted envelopes to handle the deformation. The tools to create default vertex assignments and weightings are very sophisticated and easy to use. However, if you find you really need to get in there and do the weighting manually, it's now possible. One of the features that I really like is the ability to control how many bones can influence a single vertex when automatically weighting. This can either be N Links, 2 Links, 3 Links, 4 Links, or No Blending. The No Blending case is similar to Character Studio R1, where each vertex is assigned to exactly one bone. By allowing the choice of restrictions, it is possible to set up a character that is exactly optimized for your game engine. If for example, your game engine is restricted to each vertex being blended between two bones, you can set that up easily. You can see a screenshot of the new Physique in action in Figure 1. Game developers interested in getting this information out of MAX should definitely check out the file PHYEX.P in the CStudio/Docs directory on the Character Studio disc. This file describes the export interface for Physique and shows how you would extract the vertex weights from a MAX scene.

FIGURE 1. *A screenshot of Kinetix's Physique, which uses weighted envelopes to handle deformation.*

FIGURE 2. *Nichimen's Nendo is an affordable 3D modeling program.*

Affordable 3D Model Creation

The vast majority of the e-mail I receive comes from people seeking advice about an affordable 3D modeling program. It seems that many people working on their own 3D game engines and projects cannot afford the $2,000-$10,000 required to play with the nice toys. While I can certainly understand and relate to this, I've never had a very good answer. Unfortunately, most shareware applications are not very polished or full-featured and the consumer 3D applications lack the export and texturing features needed for real-time asset creation. Luckily for us, Nichimen created Nendo. At $99, Nendo is a real bargain. It provides the same easy-to-use polygon modeling tools that Nichimen made famous in their high-end modeling package. It also includes a variety of texturing tools as well as an easy-to-use 3D paint system. Most importantly to game developers, it exports 3D models to VRML 2.0, .3DS, and .OBJ file formats as well as Nichimen's own Game Exchange 2.0 format. This should make it quite easy to integrate into any 3D game engine. All that for $99 seems

like a bargain to me. Check out an image of Nendo in action in Figure 2.

Graphics Hardware

SIGGRAPH isn't really oriented towards consumer graphics hardware. However, there was plenty of graphics power around for the animators and other graphics professionals out there. One big news item is that right before the show, 3Dlabs acquired Dynamic Pictures. Dynamics Pictures has never been much of a player in the 3D consumer hardware market, but 3Dlabs is clearly in the middle of it. This move certainly boosts 3Dlabs' position in the professional card market, as the Dynamic Pictures Oxygen series of graphics cards are a strong force within Windows NT 3D animation hardware market. I can easily see this technology trickling down to the consumer space at 3Dlabs.

On that topic, 3Dlabs' weakness in the Permedia line of consumer hardware has been low fill rate and lack of blending modes. To address this issue, 3Dlabs has announced the production of the Permedia 3. This graphics processor boasts 250 megatexels per second. Each of these texels are two-pass textured, bilinear filtered, and perspective correct. The chip appears to accelerate the entire DirectX 6 feature set, including all blend modes, and supports 32-bit color depth and 32-bit depth buffer precision. One interesting feature announced is the inclusion of voxel rendering. While this probably refers to support for 3D textures, no one I talked to at the 3Dlabs booth was clear on what exactly was happening with this feature. More information should become available as the chips go into production later this year. Given the Permedia 2's rock solid performance, by addressing the blend-mode issue as well as increasing color and Z-buffer precision while increasing fill rate, the Permedia 3 sounds like it's ready to go head-to-head with the next generation of 3D cards, such as the Nvidia TNT.

While there are currently no plans for 3Dlabs' gamma transformation accelerator technology to drop down to the consumer space, there was another professional-level graphics card with geometry acceleration entering the market. Diamond Multimedia has combined Evans & Sutherland's impressive REALImage 2100 chipset with a new geometry engine that was developed with Mitsubishi. The new FireGL 5000 will directly compete with the 3Dlabs Glint GMX product line. The fast fill rate of the REALImage 2100 should really give the 3Dlabs card a run for the money. It's clear to me that it's only a matter of time before these features end up in the consumer graphics card market. So, get ready to start raising those polygon counts again.

Technology

Many readers have sent e-mail asking where I get my ideas for where to go with my graphics technology. In large part, the answer to that is right here at SIGGRAPH. Until recently, I have felt that we were a bit behind the times in the world of game development. Much of the research I have done is on papers that are more than a decade old. However, the times are changing. This year at the conference, there was much information that was very relevant to my real-time 3D work. The courses and papers presented help influence the direction my own research will take me, and coincidentally, they represent the kinds of topics you will see in upcoming *Game Developer* issues. So, let me preview some of the technology I may be exploring both for my own projects as well as the magazine.

After seeing several talks on the subject, it struck me that subdivision surfaces may offer an ideal solution to the issue of continuous level of detail.

Anyone at the show who saw Pixar's outstanding short *Geri's Game*, (and who didn't? It was shown everywhere.) had to be impressed with the work Pixar had done. Subdivision surfaces is the underlying technology used to create their amazing organic shapes. This technique is amazing both for its realistic results as well as its elegant simplicity. Creating smooth surfaces from a low-polygon base representation is both quick and effective, as you can see in Figure 3. After seeing several talks on the subject, it struck me that subdivision surfaces may offer an ideal approach to the issue of continuous level-of-detail. Although it sounded as though the issue of real-time application had not been explored, it's certainly worth a look as an alternative to both dynamic polygon reduction and higher-level primitives. It just may solve the issue of dynamic levels of detail.

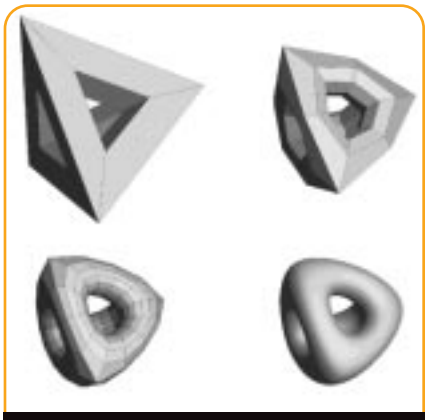


FIGURE 3. *Subdivision surfaces create smooth, organic shapes.*

In another talk, Michael Gleicher, formerly of Autodesk's Vision Technology center, demonstrated a very interesting technique for applying motion data to a model of different dimensions. This method, which he termed "retargetting," will certainly find its place in tools for working with motion capture data and other forms of character animation. If you're working on a project where you need to apply motion capture data to a variety of different character types, you owe it to yourself to check out this work.

While this year I again found Andrew Witkin and David Baraff's course very helpful in my quest to understand the complete picture when it comes to physically-based modeling, one of the most interesting developments in real-time dynamics was found on the expo floor. Dr. Michael Shantz of Intel Corp. was a speaker at the CGDC this year in a session on dynamics. At SIGGRAPH, he was in the Intel

booth showing off the results of his continuing work on a real-time dynamics engine. The engine displayed a Tyrannosaurus Rex walking across a height field terrain map. The entire locomotion engine is driven by dynamics. While this work still has a ways to go to make it robust enough to use in an interactive application, it's certainly very promising. I was amazed to find out that Intel was not sure what to do with this research. If you are interested in adding true physics to a game engine, but dread digging into the complicated math involved, I suggest you contact Intel and talk to them about licensing this technology. Dr. Shantz has clearly put a lot of work into this and is certainly eager to see it used.

With a show the size of SIGGRAPH, it is impossible to catch all the interesting lectures and side discussions that go on throughout the week. If you attended and saw something that you think will really change the direction of computer graphics, let me know about it. I was probably catching a nap in the back of the big lecture room.

If you have never made it to a SIGGRAPH conference, and are interested in the direction of 3D graphics technology, you definitely should make a point of attending the show next year. You can look at the research in the library after it's all published, but there is nothing quite like being there. I find it both inspirational and educational. It really is the best time to meet and bounce ideas off of the leaders in computer graphics. I have never failed to return from the show without a large list of things I can't wait to try out in my own projects.

FOR FURTHER INFO

Companies mentioned:

http://www.sven-tech.com
http://www.motion-factory.com
http://www.katrix.com
http://www.metastream.com
http://www.alias.com
http://www.nichimen.com
http://www.ktx.com
http://www.3dlabs.com
http://www.diamondmm.com
http://www.intel.com

Next Month

Now that my little "vacation" in Orlando is over, it's time to get back to work. Next month, I will return to the inverse kinematic algorithm. So, dig out those trigonometry books and get psyched-up for it. It's going to be a very kinematically constrained ride. ■

REFERENCES

DeRose, Kass, and Truong, "Subdivision Surfaces in Character Animation," *Computer Graphics, SIGGRAPH proceedings* 1998: pp. 85-94.

Gleicher, Michael, "Retargetting Motion to New Characters," *Computer Graphics, SIGGRAPH proceedings* 1998: pp. 33-42.

Shantz, Michael and Alexander Reshetov, "Physically Modeling for Games," *CGDC proceedings* 1998: pp. 685-738.

Witkin and Baraff, "Physically Based Modeling," *SIGGRAPH Course Notes* 1998: (ACM SIGGRAPH), Course 13.

It's About Character

Tips on creating solid-skin characters for real-time 3D games.

Modeling and texturing solid-skin characters for real-time 3D games can be one of the most fun and rewarding parts of being 3D artist. As target platforms become increasingly powerful, the limits we have to set for ourselves become increasingly less restrictive. When id's QUAKE hit the shelves a few years ago, their 100-polygon solid-skinned characters were seen as cutting edge. Nowadays, characters can reach polygon counts in the thousands, and the in-game avatars of today are looking better than their pre-rendered counterparts of just a few years ago. That said, you will need to adhere to some key steps in the process if your character is to have, well, *character*.

Concept Sketches

Generating a good concept sketch is the first and most important part of the process. Without a good concept, you may as well go back to the proverbial drawing board. Once your character makes it into the game, it's a little late to realize, "Hey, that's really not what I wanted the character to look like!"

I don't want to get off on a rant here, but it seems that I increasingly run into industry artists who neglect this crucial aspect of character design. They neglect the "Artist" part of the 3D Artist title, and instead, focus on the dazzling 3D technology, becoming more of a technical modeler than an aesthetic one.

Taking the time to work through and critique the character on paper can save you hours of headaches and weeks of redesign. These days, it's all too easy for artists to depend on the technology in front of them and forget why most of us got into this business in the first place. So get out your pens and pencils, and start sketching! (Color is optional here, but full-color concept pieces are the best because they will ultimately help the texture artists when it's time to generate the character's texture maps.) For more tips on generating a good concept piece, check out Josh White's article in the November 1997 issue of *Game Developer*, "Birthing Low Polygon Characters."

Modeling

With the technical advances in hardware acceleration and consoles, we have so many special effects and tricks available to us that it's easy to rush through this first step of character creation. Our polygon budgets have simply never been higher. It wasn't too long ago that we had to make do with less than 100 polygons for a single character. These days, you can easily spend ten times that amount on a single NPC.

And while there are at least as many techniques for character modeling as there are tools with which to model (for some tips on modeling technique, check out Paul Steed's article "The Zen of Low-Polygon Modeling" in the June 1998 issue of *Game Developer*), you need to pay heed to a few important points.

THE SILHOUETTE TEST. Your character must be recognizable for what it is by its outline. Form defines function, and the outline of a character can add to or detract from the emotion you're trying to illicit from the player. MDK, EARTHWORM JIM, ABE'S ODDYSEE, and even SUPER MARIO 64 are all good examples of games with characters that you'd recognize in any dark alley simply by the shape of their heads.

ACCOMMODATING ANIMATION. Solid-skinned models have a tendency to collapse on themselves when animated. The resulting character can end up looking like an anorexic walking around in a baggy clown suit. To keep this from happening, you need to pay close attention to the joint areas; hips, knees, shoulders, elbows, and so forth. (See Stefan Henry-Biskup's article "Character Sheets" on page 44 of this issue for more on anatomically efficient modeling.) Again, this problem is less daunting than it was a few years ago thanks to the higher

Introducing Mel

Experience

I started making shareware games back when the 286×25 was considered the top-of-the-line machine, and my business cards have seen job titles ranging from 3D Artist to Art Director. I've had the fortune to work on projects at companies as large and corporate as Eidos Interactive and as small and close-knit as Surreal.

Goals

While 3D character animation is my one true love and area of expertise, I've had the opportunity to learn various artistic tricks and techniques from some of the most talented people in the industry. My goal with this column will be to disseminate some of that knowledge back into the industry as a whole, and hopefully help to raise the bar a bit when it comes to in-game art.

Topics

In the upcoming months, we'll look at as many aspects of art production and artist management as we can reasonably fit in the allotted space. The goal will be to keep this whole process timely and accurate, with a good balance of how-to topics, such as this month's column. I'm also open to your suggestions. I can be reached by e-mail at mel@surreal.com.

number of polygons available to us.

In Figure 1, the simple addition of an extra segment of vertices at the elbow allows the joint to flex correctly without collapsing the arm segment. In contrast, the simpler construction in the first model removes the fidelity of the animation, and the joint collapses on itself.

POLYGON COUNT. Just a few years ago, we had to do more with much less. As polygon-budgets increase, the trend is inevitably towards less efficient modeling tools because, hey, we artists can get away with it. Don't be lulled into a sense of gluttony here. As the hardware gets better, it will be easier to lose the sense of urgency associated with keeping your polygon counts to a minimum. But there's just no sense wasting polygons unnecessarily. Character polygons are often the most technically expensive to render, and most engines pay extra in terms of processing time for polygons that must be transformed and manipulated through animations. Shaving a couple hundred polygons off of a character often gives you two to three times that number for other objects in your scene, or lets you keep an additional creature onscreen while still maintaining your target polygon count.

Texture Mapping

After the concept sketch, good texture mapping is probably the most important part of a 3D artist's job. As artists, we need to be intimately familiar with the way colors and textures affect the look of our underlying geometry. A skilled texture artist can use textures to create the illusion of polygonal details not present in the model (such as using darkened areas to round off a corner). And while a picture may be worth a thousand words, a couple dozen pixels can be worth their weight in polygons. But it's easier to show than tell, so on to

The Cultist

Game: EXPERIENCE
Developer: The Whole Experience (WXP)
Publisher: TBD
Character statistics
Poly Count: 1,076
Vertex Count: 583
Modeling tool: PowerAnimator 8.5
Texturing tool: PowerAnimator 8.5
Character type: Solid-skin real-time 3D

FIGURE 1. *Joint construction.*

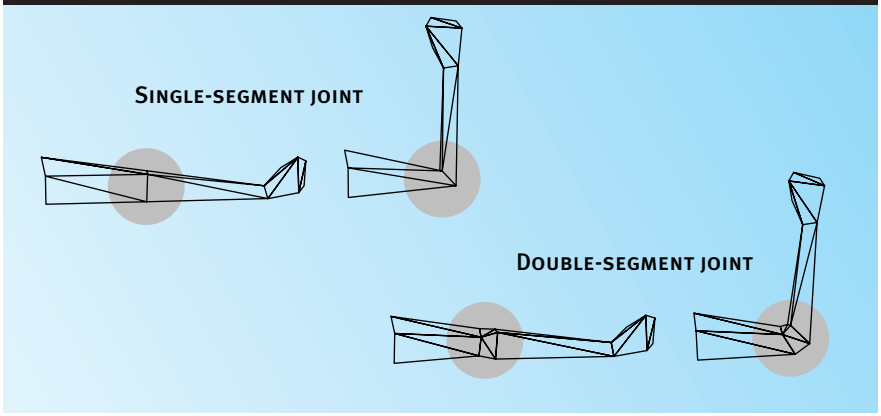


FIGURE 2. *The Cultist concept sketch.*

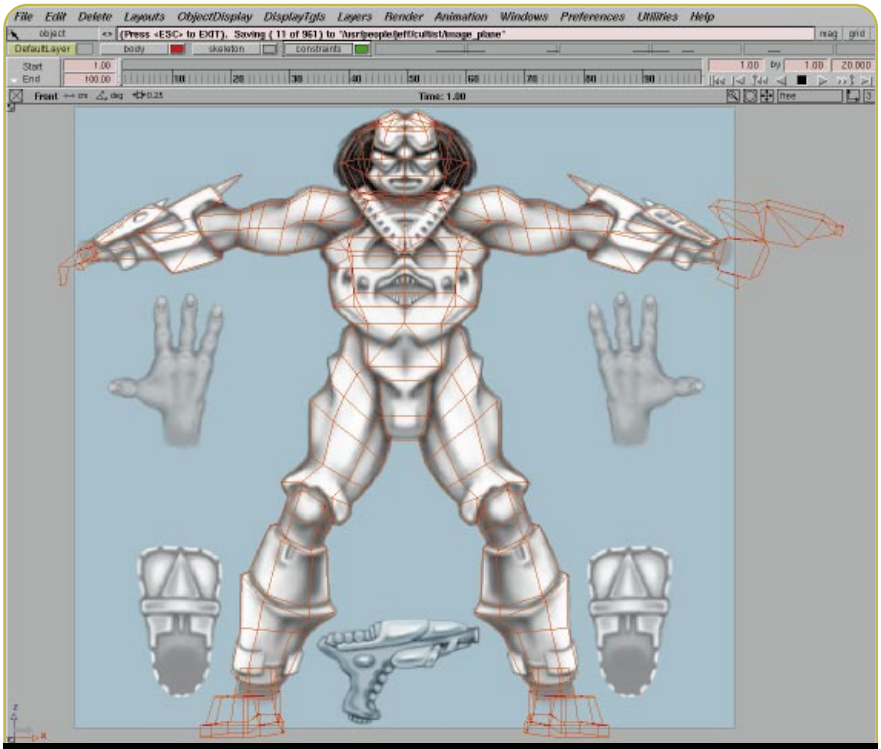


FIGURE 3. *Modeling in PowerAnimator using an image plane.*

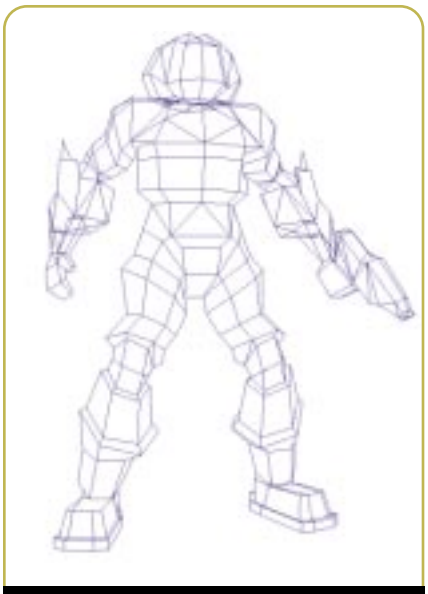


FIGURE 4. *Cultist wireframe.*

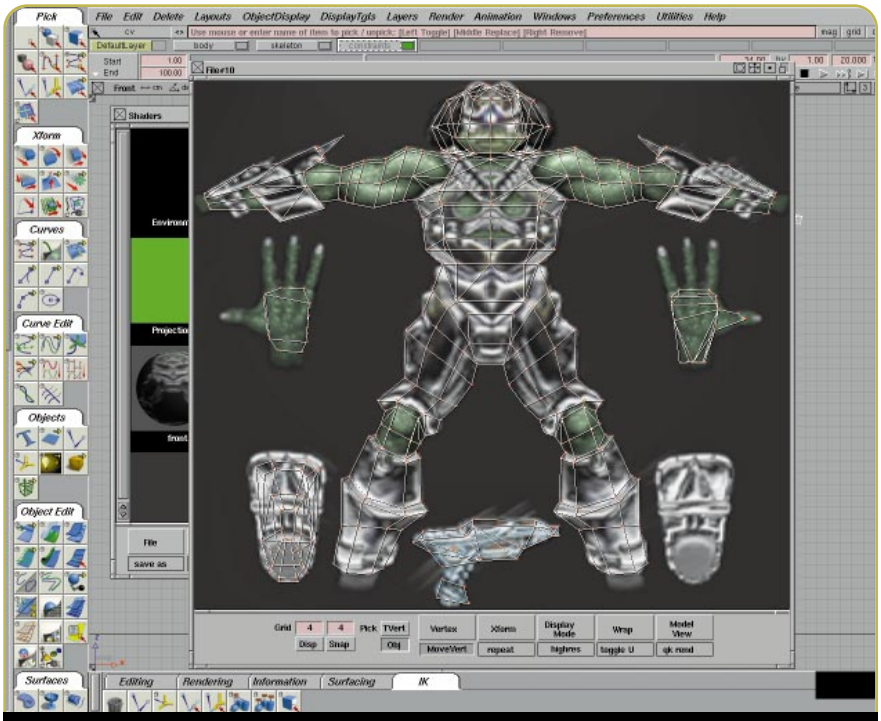


FIGURE 5. *Texture application.*

the dissassembly.

The boys over at The Whole Experience (WXP) in Seattle, Wash., are working on a real-time 3D title called EXPERIENCE. One of the main protagonists is this gun-toting bad boy (Figure 2), who you have to defeat in one of the later missions. The art direction on this product leans heavily on the neocartoonist style of Lyndon Sumner, WXP's concept artist.

Starting from a sketch on paper, WXP's character artist generates a place-holder texture map in black and white, which the modeler can use to build the character. The black and white texture is handed off to the modeler, and while the modeler fleshes out the character in 3D, the texture artist goes back and adds color and detail to the character texture.

The obvious advantage of using a place-holder texture is that it helps to prevent bottlenecking in the character generation pipeline. The character modelers can move on to animating the character without having to wait for the character to be fully textured, and the texture artist isn't as rushed in completing the textures for the character.

The place-holder texture is brought into PowerAnimator as an image plane (Figure 3), over which the modeler can model directly to get the correct silhouette for the character. If you look closely, you can see how accurate the polygon mesh is compared to the concept piece. Using PowerAnimator's bi-rail extrusion tool, the modelers at WXP were able to follow the basic outline of the original concept with a very high degree of fidelity.

The image plane technique is available in some form on most major modeling packages. Taking the process one step further by generating top- and side-views of the character can make the modeling process go even faster.

As you can see from the character wireframe in Figure 4, the character has a basic humanoid outline with just enough variance to make it stand out.

Now we get to the texture application (Figure 5). Because WXP's artists model their characters using the place-holder texture as a guide, the mapping coordinates are very easy to apply. But any of you who've done this know that mapping the sides of a leg or a torso inevitably produces a smearing effect, unless you use a cylindrical type of mapping coordinate. Furthermore, no single cylindrical mapping coordinate could map this character, and inevitably, you'd be required to adjust, by hand, the texture coordinates around the edges of the model.

WXP's technique solves this problem in a snap. The character is seamed together like two clamshells, with an edge running all the way around the model. The resulting model is texture

FIGURE 6. *The Cultist, fully textured.*

FIGURE 7. *The War Giant concept sketch.*

FIGURE 8. *Complete texture set for War Giant.*

mapped using planar mapping coordinates (except for the hands and feet, which are mapped separately). And because the artists built the model on top of the texture map, the mapping coordinates fit perfectly.

Keeping an edge at the outline of the character largely eliminates the problem of smeared textures over the arms, and using a single mapping coordinate guarantees a pretty uniform pixel density without creating any seams in the texture. The down side to this method is that it wastes quite a bit of texture space. But with texture buffers for the target platforms running around 6- to 8MB, the folks over at WXP don't seem too worried.

To sum up, the techniques WXP used to create this character produce good results without many of the headaches usually associated with creating work of this quality. Working from a solid concept piece, they were able to generate a character that retained all of the important qualities of the original design (Figure 6). The character model was recognizable in profile, and the texture maps were applied in a very short time frame, while maintaining a seamless, uniform pixel density. Overall production time from concept to finished, textured model ran about a day and a half.

Figure 7 shows a color concept for the War Giant, a two-story tall villain in Psygnosis's DRAKAN, being developed by Surreal Software in Seattle. DRAKAN's development team has been augmented by the talents of concept artist Hugh Jameson, whose unique fantasy style lends a credible realism to the DRAKAN cast of characters. Having an artist on staff that can crank out work of this quality is invaluable. The full-color concept piece eliminates the guesswork

The War Giant

Game: DRAKAN
Developer: Surreal Software
Publisher: Psygnosis
Character statistics
Poly count: 576
Vertex count: 270
Modeling tool: 3D Studio MAX R2
Texturing tool: Proprietary tools
Character type: Solid-skin real-time3D

FIGURE 9. *Texture mapping using Surreal's proprietary modeler.*

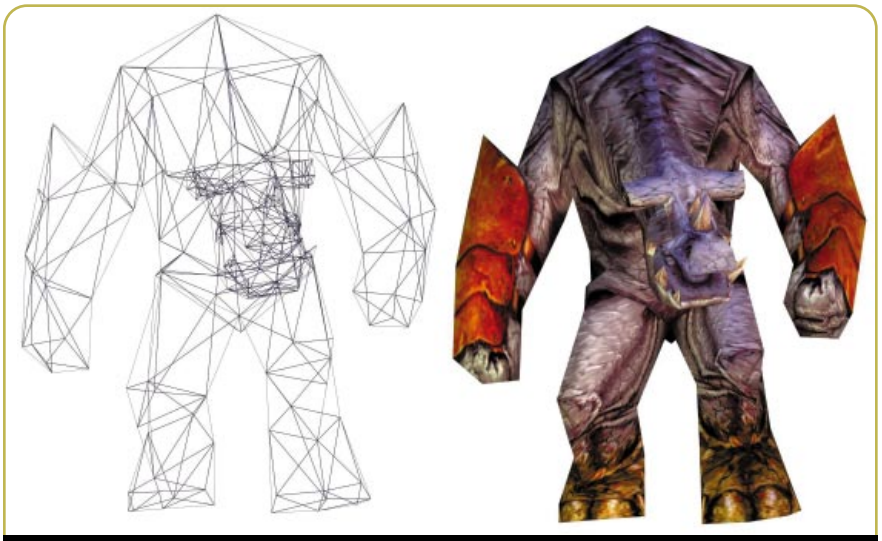


FIGURE 10. *War Giant wireframe and textured model..*

when it comes to generating the textures, and the time commitment for creating work of this quality is clearly worth the result.

Figure 8 shows the textures that Surreal's artists created using the initial color concept piece as a guide. Note the high degree of realism and intricate detail on the back, neck, and head of the creature. The War Giant character stands nearly 20-feet tall in the game. Consequently, the large polygons on the creature's legs will need to have an incredibly high pixel density, necessitating the use of large textures.

While this level of texture application requires a much larger time investment, the equitable use of texture space allows for a much higher local pixel density than in the previous method. Note that care has been taken to create areas of the texture that can be used more than once. The texture artist has taken advantage of the fact that the war giant is a biped, with symmetrical body composition. Only one arm and one leg are included in the texture, and only one side of the chest, back, and neck is modeled in complete detail.

While this process demonstrates excellent use of texture space, the technique involves some painstaking work on the part of the person applying the texture maps. It took a full four days just to generate and apply the texture maps for this character. The result, however, bears out the cost in man hours, and the resulting composite 512×512-pixel texture map provides for a very high pixel density on the model.

Using a proprietary tool generated in-house at Surreal, the texture artist is able to interactively texture map individual polygons with a high degree of accuracy. In this case, the texture maps were created during the mapping process; the artist generated textures as needed. This way, the texture is used in a very efficient manner. Since planar mapping wasn't used for this example, extreme care had to be taken to avoid any seams in the textures or discontinuities in pixel densities on the 3D mesh. Here, the additional time required to apply the texture maps to the model was recouped by the quality of the result.

Figure 10 shows the wireframe and fully textured model generated in 3D Studio MAX. Using the concept piece and several pencil sketches as a guide, the modeler generates a low-polygon version of the mesh using 3D Studio MAX. As in the previous example, the modeler applies mapping coordinates while generating the model. And while the polygon count is less by almost half, the basic shape of the character is recognizable. As you compare the wireframe and finished model, it's immediately apparent how the correct texture can make all the difference. ■

Lessons Learned

We've seen examples of two different techniques for modeling and texturing a solid-skinned character. Let's compare them side by side.

The Cultist

Modeled using an image plane backdrop generated from the original concept sketch, which was then used as the 256×256 texture map for the character.

Pros.

- Very rapid production time; approximately five days from blank paper to fully textured model.
- Guaranteed fidelity between concept and final result by using image plane method.
- Place-holder textures relieve the bottleneck between concept and modeling and animation tasks
- Planar texturing method assigns uniform pixel densities and results in seamless texture mapping.

Cons.

- Planar texturing technique is fast but wastes texture space.
- Lower overall pixel density due to use of a single texture map.

The War Giant

Modeled using classic modeling techniques and textured with four 256×256 texture maps. Mapping was applied by hand adjusting the UVs in a proprietary paint tool similar to those available in Softimage, 3D Studio MAX, and PowerAnimator.

Pros

- Final result is an extremely high-quality textured model.
- Higher pixel density available due to multiple high-resolution texture maps.
- Efficient use of texture space means lowest possible texture buffer.

Cons

- Time commitment is significant. The War Giant took over 10 days to go from concept to fully textured model.
- Bottleneck between concept and modeling and animation is significant.

Special Thanks

Jeff Connelly, Sky Kensok, Patrick Moynihan, Lyndon James Sumner II, Ron Levine, Jame Thrush, Stuart Denman, Hugh Jameson, Heron Prior, and Louise Smith.

Trends in the Display Industry

What's a game system without a screen or display? Nothing to look at, that's for sure. Therefore, it seems appropriate to look at some of the trends in display technology. The move to larger display areas will have a direct impact on gaming in the coming years. The addition of multiple displays will also have some effect, but in limited areas. Potentially, stereoscopic displays will be the most important developments, but I remain skeptical at present on that one.

The most immediate impact on the industry is going to come from the growth in 17-inch displays. Judging by the promotions for new system sales of multimedia PCs, the 17-inch screen has become the standard display, and as a result, a very affordable upgrade for new PC buyers. I was offered a pretty decent-looking 17-inch monitor by a local dealer for about $350. It's still the cost of a good-size television set, but by computer standards, it's almost a steal.

The increase in display area has been a long time coming when you consider how long Windows has been around. Windows and the 1,024×768-pixel resolution display go hand in hand, and on anything less than a 17-inch display the resulting image is a good recipe for squinting eyes and fatigue. Furthermore, it seems that graphics board vendors are eschewing the compute-intensive penalties of antialiasing in hardware in favor of 1,024×768 performance, which makes for as compelling a gaming experience as you could hope for with today's technology.

Nevertheless, the bulk of display sales still remain primarily 15-inch, which is in itself a vast improvement over the tyranny of the 13- and 14-inch screen. Indications are that more home users are going to turn to 17-inch screens in the coming years. Commercial, or business, users may be inclined towards parsimony when it comes to handing over the few extra dollars.

Unfortunately, display markets are subject to consumer intransigence for a number of reasons. While some consumers are quite happy to fork over a couple of thousand dollars for the latest in desktop horsepower every eighteen months, they are loathe to update their screens. Part of the blame for this has to lie with the physical space that a cathode ray tube (CRT) display takes up. Some stretch back as far as they stretch across and that means a single 17-inch screen can consume fifty percent of your desk space. There's no way of avoiding giving a CRT a prominent place on your desktop, unless you like to crane your neck at painful angles. If desk space was not an issue, then we'd probably see a jump in 21-inch display sales because the price differential justifies the results of having 1,280×1,024, or 1,600×1,200 Windows.

Of course, trends in the display market seem to be rather irrelevant when most games still default to 320×200 resolution or the good old 640×480 of VGA. The PC industry is definitely working on making 1,024×768 the standard default over the coming eighteen months, but legacy applications and VGA seem irreplaceable. Even Windows setup screens are still defaulting to the chunky pixel look of VGA, and the results are, to say the least, anachronistic. After all, I don't know of a single recently released graphics chipset that couldn't support 1,024×768 resolution as a default.

Despite all of this, monitor prices have not dropped as rapidly, or in pace, with PCs. If they had, a 21-inch monitor would probably cost less than $300 today. Nevertheless, as PCs have come down in price, and competition in the consumer market has exploded, the addition of a 17-inch monitor as part of a new multimedia home computer package has become a competitive advantage for those PC companies with the leverage to stand up to the display vendors. Yup, the PC vendors' bundling of displays is driving down the prices of displays from Mag, Sony, NEC, ViewSonic, Samsung, and other vendors of CRT displays. At the present rate, PC vendors are probably going to dominate the distribution of 17-inch screens, leaving other display sizes to the monitor vendors, many of whom sell directly to large corporate accounts. Of course, not all displays sold by monitor vendors are attached

	1997	1998	1999
15" monitor	29,408	32,231	35,325
17" monitor	14,068	16,783	20,022
20" monitor	2,143	2,298	2,463

FIGURE 1. *Unit growth of PC displays (in thousands of units). Various sources.*

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.

to a PC. There are ATMs, point-of-sale displays, and terminals to think of as well.

It's no wonder display manufacturers are shifting their focus to emerging technologies. The CRT dates back to the early research in television systems in the 1920s. It was only when portable computers showed up in the 1980s that the industry started to get a taste for alternatives to the cumbersome technology of the tube — alternatives such as liquid crystal displays (LCD). Plasma technologies today are still emerging technologies, but consumer electronics companies are investing billions of dollars in trying to make flat screens that hang on a wall like a mirror or painting — and the bigger the display, the better. These are screens targeting the world of television, still the biggest market for displays, but the computer industry is playing its part in the acceptance of flat-screen displays.

The alternative technologies to CRT displays are

- Thin-film electroluminescent (TFEL) displays
- LCDs: monochrome displays (Color LCDs continue to dominate the computer market, but are unsuitable for large flat displays.)
- TFT-LCD: color flat panel display (FPD)
- Field emission display (FED): electron tubes in which cathodes emit electrons. (FEDs are not a significant area of growth in comparison to other plasma screen technologies.)
- Plasma display panel (PDP): used in large panel displays for financial applications where desktop space and display area are equally important.

The emergence of FPD screens in the consumer market is going to take place in the consumer electronics market first. The costs of such devices in the computer industry make them prohibitively expensive for mass-market consumption, hence they are relegated to specialized markets — much as PDPs are finding favor in financial trading houses where brokers have multiple screens on their desktops.

The most comprehensive resource on display technologies and markets is Stanford Resources (http://www.stanfordresources.com) for anyone interested in delving deeper into these markets. In addition, Bob Raikes in the UK (http://www.meko.co.uk) provides a pretty good newsletter, *Display Monitor*, that covers display and graphics hardware.

Multiple Displays

It will be interesting to see how Microsoft's support of multiple displays in Windows 98 will impact the market as well. Multiple display support provides three unique perspectives that developers can address.

LARGE DESKTOP. The bounding area of all displays,which may vary in terms of both resolution and color depth, is the virtual desktop area. Some of the problems that this may cause are related to ways legacy applications center dialog boxes and access negative coordinates of the screen space. In multiple display situations, negative screen coordinates are also visible. Microsoft has taken these problems into consideration, and the programming fixes for applications are relatively simple. Obviously, with multiple display support, the opportunity will exist for these same applications to utilize more desktop space in unique new ways.

REMOTE DISPLAY. The display can be mirrored remotely, or as part of the same configuration. This means that a secondary display need not be a companion of the primary display.

INDEPENDENT DISPLAY. A high-resolution screen appears on a secondary display and is completely independent of the Windows interface. This means that images on the secondary screen in this type of configuration can work through Windows APIs without having to be a part of the virtual screen.

There is some thought that multiple-display support for gaming is only useful in Arcade PC applications, but hardcore games enthusiasts have shown themselves willing to spend money on hardware that enhances their experiences and heightens the pleasures of gaming. Multiple monitor support for games using the independent display model offers some intriguing possibilities, and implementing support in a game does not have to be a taxing technical burden. The code to support multiple displays in a Windows 98 application is pretty straightforward. But, this is all conjecture. There's a great idea on paper, and then there's reality.

Stereo and HMDs

Talking of great ideas on paper, what may prove to be an important trend in display technology for gaming is the emergence of stereo displays and head mounted displays (HMDs) which offer the kind of immersive experience that you would expect of virtual reality simulators.

After all, stereo is the way we see the world. To simulate the views that our eyes see in the real world, you need to produce two images of slightly different perspective, and then combine them in the mind's eye. In the real world, objects near and far are displaced relative to the left and right eye view by different distances. The closer the object, the greater the displacement. Stereo displays use the mind's ability to remember the way things work in real life to create the impression of depth in the virtual world.

There are a number of problems with stereo displays. They look great at exhibitions, but the novelty wears off when a user walks into a store and has to hand over cold, hard cash for a pair of glasses. Even when the price and performance of a product are right, it's difficult to engage potential users without a one-on-one demonstration. So, while stereo sounds great in theory, and can have immense appeal, it's a difficult sell. Then there's the issue of having to wear stereo and HMD stuff that looks like a brainwasher's torture kit, and makes one susceptible to an appearance on America's Funniest Home Videos. In short, stereo and HMD gear is about as goofy looking as you could ever expect from a computer peripheral, and unfortunately, you have to wear them to use them.

There is also the perception — stemming mostly from historical problems with stereo — that stereo and HMD devices cause eye strain, headaches, and seizures. Technology has also played a role in holding back the tide of stereo, and as a result, game developers have been reluctant to put any great effort or investment into creating stereo titles. However, that may all change.

The current technology for stereo display is LCD shutter glasses. They are lightweight, cheap enough for consumer use, and with graphics horsepower on the rise, they are becoming

technologically appealing too. With stereo, you can alternate between showing the left and right view images alternately and let the mind meld them together seamlessly, or show both images at high frame rates. LCD shutter glasses are triggered to shut one eye off to an image while the other one is displayed so it makes an interleaved display of stereo images the most suitable option.

The key difference to doing stereo today, as compared to even a year ago is that developers are making real-time, true 3D titles. This makes it a little more appealing for stereovision developers to implement depth in the user's experience. Also, an API such as OpenGL has support for stereo at the driver level, and Direct3D will eventually make it standard. So, good quality stereovision today doesn't necessarily mean that game developers have to create special versions of their titles, but it does mean that they have to do their 3D math right in their game engines.

There are a handful of stereo display companies with consumer-level products. At present, H3D (http://www.h3dentertainment.com) is engaged with the Wicked3D Board Company (http://www.wicked3d.com) in creating a Voodoo2-based stereo driver that will support all OpenGL, Glide, and Direct3D games. These two companies believe that by making stereo possible at the driver level, and by certifying products according to the quality of the stereo they provide, they will create demand for stereo gaming.

Unfortunately, with the exception of some high-end computer graphics applications, stereo in the consumer market remains on the fringes for now. The following companies are worth checking out for stereo display technology that could be applied to the gaming market. H3D, i-O Display Systems (http://www.vio.com), VRex (http://www.vrex.com), and Stereographics (http://www.stereographics.com) should be familiar to gamers from past forays into the consumer market.

- Chromatek has interesting technology (http://www.chromatek.com).
- General Reality also has interesting technology (http://www.genreality.com).
- H3D (in conjunction with the Wicked3D Board Company) is one of the better solutions for stereo.
- Interactive Imaging Systems (formerly Forte Technologies) has gone the retail route, but is looking at diverging market opportunities (http://www.iisvr.com).
- i-O Display Systems has a retail presence.
- Stereographics tried the retail stereo glasses business, but got caught in the hell that is retail sales.
- VRex has good retail presence.

I remain skeptical about any mass-market product that needs to be seen to be believed, but that's not the case just for stereo. It's actually true of all displays. What is certain is that 1,024×768 on a 17-inch screen is a mass market phenomenon that has taken six years to arrive. What is also certain is that displays are going to continue to go down in price, driven by cheaper pricing. Displays are going to get thinner, lighter, and more attractive. It's probably time to say goodbye to VGA, forever. ■

Back

GAME AI: THE STATE OF THE INDUSTRY

by Steve Woodcock

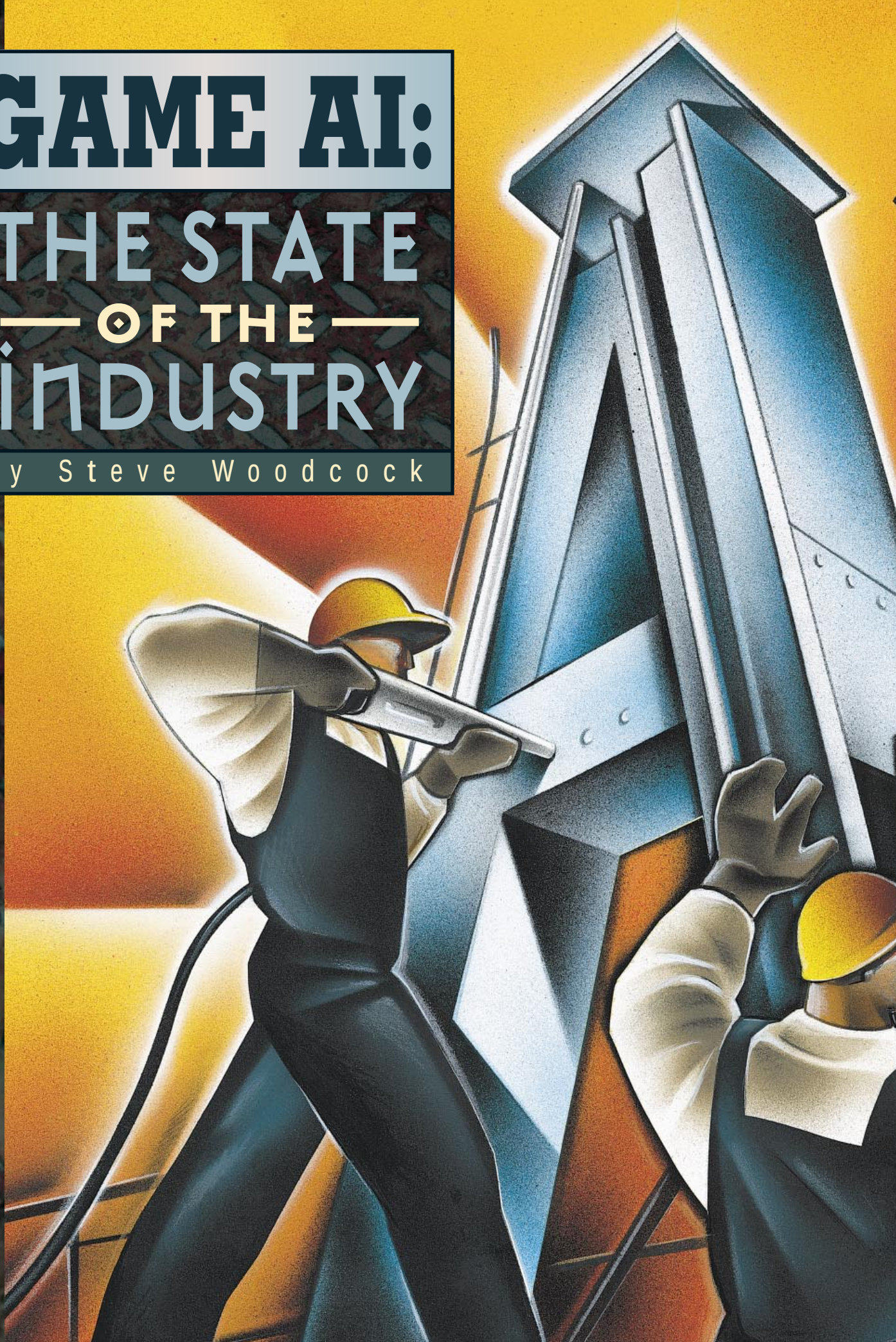
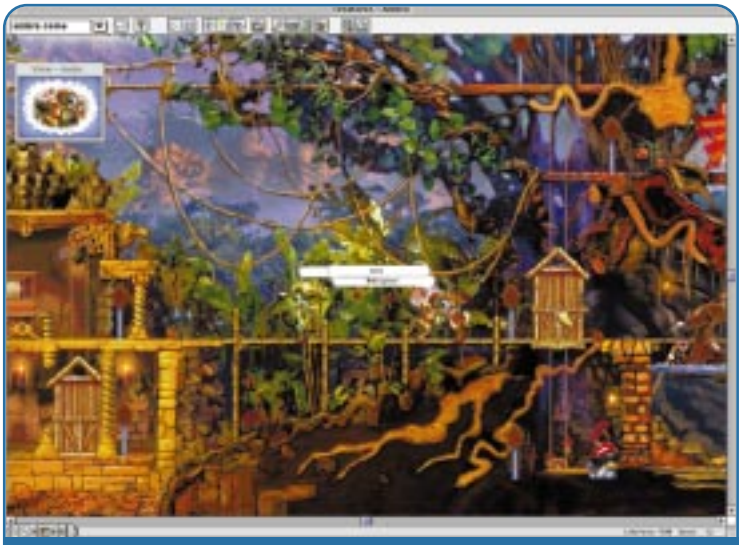


Illustration by Vadim Vahrameev

Since the birth of the videogame market, artificial intelligence (AI) has been a standard feature of games — especially with developers' emphasis on single-player games, which represented the vast bulk of released titles up until fairly recently. Although the recent support for multiplayer capabilities (especially via the Internet) and free online gaming services such as Battle.net and Microsoft's Internet Gaming Zone may have prompted some developers to downplay the need for good computer opponents, good AI is needed more than ever.

Steve's background in AI comes from a decade of SDI-related work building massive real-time distributed wargames for the Air Force. When he's not saving the world, he's busy doing AI development on a contract basis and target shooting when he gets the chance. Steve lives in gorgeous Colorado Springs, Colo., with a very understanding wife and an indeterminate number of ferrets. His e-mail address is swoodcoc@concentric.net.

At the 1996 Computer Game Developer's Conference (CGDC), I attended a number of excellent technical discussions on game AI. Everything from formal university techniques to better pathfinding was presented to standing-room-only crowds. As a result, developers Eric Dybsand, Neil Kirby, and I thought the time might be right to begin a series of CGDC roundtables on game AI, both as a forum for developers to exchange ideas and as a venue for gauging the industry's use of AI techniques. This article discusses what was discussed in those roundtables, details some industry trends, highlights some recent games that use interesting and innovative AI techniques, and examines how the impact of offloading 3D graphics processing onto hardware may affect AI. I'll wrap up with the thoughts of various game developers on the future of game AI.



CREATURES

Late to the Resource Dance

Historically, game AI has been one of the last maidens to join the dance during the development process. Emphasis was placed on other aspects of the game, from tuning 3D engines to integrating last minute sound effects. Design and coding the computer opponents was often deferred to the final phase of the project, when it was dumped on whichever hapless developer happened to be least busy at the moment. There was little planning beyond "Get it done by the shipping date," and little design beyond "Don't lower the frame rate."

In part, this can't be helped. For obvious reasons, most testing and tuning of computer opponents can't really be done until large portions of the game are already complete. Since this doesn't usually happen until close to the shipping date, it can be difficult if not impossible for developers to correct any major deficiencies in the AI if anything pops up at the last minute. That's one reason why patches are issued a mere week or so after a game hits the streets.

This state of affairs seems to be changing, however. More developers are becoming involved in the design and implementation of the AI at an earlier stage, and more resources are being dedicated to AI processing. Table 1 shows the results of two questions we asked all developers who attended the CGDC AI roundtables in 1997 and 1998. While these figures and others from the roundtables shouldn't be considered statistically accurate and cannot necessarily be applied across the entire industry, they do show some encouraging trends.

The number of projects with developers dedicated to game AI increased sharply from the roundtables of 1997 to 1998, from roughly 24 percent of all projects represented to nearly 50 percent. A handful of developers reported that there were "two or more" programmers dedicated to working AI, and one developer at the 1998 session reported a whopping three programmers wholly dedicated to game AI and scenario development. AI is still taken care of by a small subset of the programming staff, and these developers often have other duties as well, but this situation seems to be rapidly changing.

The average number of CPU clock cycles reportedly dedicated in some fashion to AI processing also jumped from the 1997 to 1998 roundtables, from an overall average of 5 percent to around 10 percent. When broken out between real-time action games and turn-based games, we noted that the biggest gains were made by developers who were responsible for the former. This jibes with the fast evolutionary pace of the real-time action game genre. On the other hand, gauging AI advances in turn-based games is necessarily tenuous: many of those present who were working on turn-based projects said that they basically received all of the CPU power they needed when it was the AI's turn.

Significantly, developers overall reported that AI was becoming a more important factor in game design than it had been in the past, often being assigned an equal priority with graphics and sound in the initial game design and allocation of resources. Those responsible for game AI (whether or not it is their sole task on the project) are getting involved earlier in the design and development cycle,

TABLE 1. *Resources dedicated to game AI development.*

Does your team include any full-time AI programmers?

	1997 CGDC	1998 CGDC
	50/207 (~24%)	87/190 (~46%)

What percentage of the CPU overall does your game's AI get for its processing?

	1997 CGDC	1998 CGDC
Overall	~5%	~10%
Real-time games	1% - 2%	5% - 10%
Turn-based games	5% - 25%	5% - 50%

smoothing the way for better integration between the AI and game engine. Fewer developers reported having the AI dumped on them in the last month of development, although everyone cited the crunch of the shipping date as a major factor in the tuning of any game AI.

A handful of developers in the roundtables noted that working on a sequel to a successful game, or building on a licensed 3D engine technology, often gave them greater freedom to pursue development of more capable computer opponents. In such instances, this growing emphasis on AI seems as much feature-driven as technology-driven, as companies look to differentiate products in the marketplace.

An interesting side effect of the rapid advances in the 3D hardware acceleration market is that the CPU has been freed up for more AI-related tasks. While some developers at the roundtables felt that the trend towards more glitz in games was stealing CPU cycles away from other portions of the game engine, most developers were of the opinion that offloading 3D processing to 3D hardware was gradually giving them CPU cycles that could be used for deeper AI processing. All the developers felt that offloading 3D processing was a positive trend, especially with today's powerful CPUs (such as the Pentium II and the AMD-K6-2 with 3DNow!) providing orders of magnitude more processing power than were available just a few years ago.

Another trend is the rise of the AI specialist — an external developer who focuses solely on creating the game AI. (Full disclosure requires me at this time to note that I'm just such an individual myself.) This can be a low-risk approach for companies that are interested in providing more challenging AIs in their games, but which lack the in-house experience to develop it. Several developers also noted that their companies are in the process of setting up their own internal teams to handle all AI development across multiple games, which is something that simply didn't happen just a couple of years ago.

S.W.A.T. 2

A handful of people from development studios stated that they were also attempting to find developers with knowledge of both AI and general writing/story-telling expertise. This trend was most evident among groups working on RPGs, especially online products, and likely reflects some of the lessons learned from Origin's ULTIMA ONLINE experiences. Developers working on these projects felt that by hiring AI developers experienced in writing and story telling, they could build more realistic worlds that would interact with players more consistently with what the games' producers had in mind.

Technologies in Use Today

Exploring the AI technologies used in games has been popular in the CGDC roundtables. As the importance of good game AI increases, developers have been turning to academia and

AI Technologies

FINITE STATE MACHINES. The FSM is probably the single most common AI technology in use, though many developers may be unaware of it. Simply put, an FSM is a logical hierarchy of rules and conditions that can only be in a fixed number of possible states. The inputs and outputs of each state are identical, and there's no choice of the sequence in which states are visited.

An example of an FSM might be the logic controlling the doors on the USS Enterprise. When one approaches them, they open; when one clears them, they close. In Red Alert conditions, only authorized personnel can use them. A few simple conditions (authorization and proximity) control a limited set of actions (opening and closing). Sequencing is maintained by the fact that a door can't be closed if it's not open, and vice versa.

FUZZY STATE MACHINES. The FuSM is much like its FSM brother, with a twist: rather than a given set of inputs mapping to a specific output, they map instead to a "membership" to a set of possible responses. These potential responses are each given a weight based on their membership. Developers then use a variety of methods to select a given response — the weights might be used as a simple probability, or they may be accumulated and matched against some threshold value to see which one is triggered. The result is a state machine that can generate a variety of different, yet plausible, responses to a given set of stimuli. Keeping with our example of the USS Enterprise, where the doors might be controlled by FSMs, we would probably want a Klingon warship controlled by an FuSM, so that it could react flexibly to changing battlefield conditions.

ARTIFICIAL LIFE. A-life is the name given to a particular discipline that studies natural life by attempting to recreate biological phenomena from scratch within computers and other artificial media. A-life emulates biological life by building systems that behave as living organisms. Depending on

Continued on p. 30.

defense research for information on more exotic technologies and better methods than the brute force approaches often used in the past. Here are some AI technologies that have been employed recently.

RULES-BASED AI. Rules-based AI, characterized by the heavy use of the almighty finite state machine (FSM) and its close cousin, the fuzzy state machine (FuSM), remain the technologies of choice for AI development. Every game on the market uses rules-based AI to some degree or another, and with good reason: these methods work. Rules-based AI remains the foundation of opponent intelligence in most games for three simple reasons:

1. Rules-based approaches are familiar, taking their principles from comfortable programming paradigms.
2. Rules-based designs are generally predictable and hence easy to test (although this, in turn, leads to one of the biggest complaints on the part of players — predictability of game AI).
3. Most developers lack any training in, or knowledge of, the more exotic AI technologies and thus don't turn to them when deadlines loom.

Valve Software's HALF-LIFE (http://www.sierrastudios.com/games/half-life) provides some excellent examples of what can be accomplished using FSMs. The AI in that game uses what their developers call a "schedule-driven state machine," which is another way of saying it's state-dependent and response-driven. By layering FSMs, the developers were able to achieve some remarkable complexity in the way the AI behaves — monsters can panic and run away when under fire, move in formation when attacking the player, and fetch reinforcements if they see they're losing a battle.

Yosemite Entertainment's S.W.A.T. 2 (http://www.swat2.com), on the other hand, demonstrates some of the capabilities of FuSMs, which were heavily used for the tactical AI in that game. This approach was driven by the game's design: the developers wanted set scenarios that could play out in any number of plausible, yet different ways. Similar games often use scripting to lay out a scenario, but this can lead to predictability and limits its replayability. S.W.A.T. 2 gets around this by building FuSMs. Each possible decision is run through a bit of code they call a "fuzzy decider," which weights the possible responses of any given unit based on its health, personality, and so on, and then chooses one of these randomly. The result is a series of actions that are perfectly believable and realistic, but which vary from any previous game so the player is never quite sure what will happen. FuSMs also provide more of an "analog" feel to the AI, allowing a spectrum of realistic behaviors that might not be expected by the player.

One nice side effect demonstrating the intrinsic power of such a straightforward approach to AI can be seen in something called "emergent behavior." A developer faced with coding a com-

UNREAL

AI Technologies (Cont.)

the design, one can use hard-coded rules, genetic algorithms, or a variety of other methods for this emulation.

GENETIC ALGORITHMS. GAs are an approach to machine learning that derive their behavior from the processes of natural evolution. This is done by creating within a machine a population of individuals represented by chromosomes — in essence a set of character strings that are analogous to the chromosomes that we have in our own DNA. The individuals in the population then go through a process of evolution in which they are tested against some fitness criteria. Those that fail are discarded, while those that score highest are allowed to breed — essentially, creating a new individual using a combination of two or more individuals' chromosomes. Mutation is often allowed to prevent things from settling into a steady state. The result is a population of individuals that gradually adapt themselves to the constraints of their digital environments, in effect evolving over time.

Why Do So Many AI Technologies Seem Similar?

The astute reader might notice that a lot of AI technologies seem as though they're variations on a theme. Genetic algorithms, neural networks, and even FSMs can all modify themselves; GAs, neural networks, and A-life are all biologically-based, and so on. One might well wonder, therefore, why AI experts bother to make distinctions between them.

It's a good question, and unfortunately a good answer might take more space than allowed by this article. Suffice it to say that various AI technologies have a lot in common with each other, and can be viewed as variations on a theme. (For example, a neural network that is non-learning is basically just a big set of rules and can be reduced to a series of cascaded FSMs.) Adding to the confusion is the fact that there are different names for some AI technologies depending on what field of engineering is using them.

plex AI can instead break it down into smaller pieces. Rather than having to code thousands of rules for every conceivable situation that could arise in a game, lower-level behaviors are coded individually and then linked in a decision-making hierarchy. The interaction between these low-level behaviors can cause higher-level, more "intelligent" behaviors to emerge without any explicit programming. This is the basis behind flocking, an AI technology that seeks to emulate the behavior of large groups of animals, such as flocks of birds or schools of fish. No one member of a flock actually knows anything about the motion of the flock, and yet through individual actions their motions are coordinated and fluid. Flocking has found its way into a number of first-person shooters and RPGs as the basis of sort of artificial life, another form of AI technology seeing more use in games.

GENETIC ALGORITHMS. A number of game developers have started investigating the use genetic algorithms (GAs) in their games. Genetic algorithms, which you can read about in Swen Vincke's article on page 36 of this issue, are something of a rules-based approach to AI, but focus on using digital analogs of biological DNA that allow an AI, in effect, to rewrite itself in response to its environment.

Cyberlife's CREATURES (http://www.creatures.mindscape.com) makes heavy use of a combination of chemistry-based GAs and neural networks to control virtual pets (called Norns), which learn how to speak, feed themselves, and interact with the player in a variety of ways. These creatures live in a virtual world filled with unexplored areas, beginning the game in nearly total ignorance of their environment. Every aspect of an individual Norn is encoded in a form of digital DNA and can be passed down to descendants over time, allowing players to selectively breed Norns with desired characteristics. A sophisticated mutation engine forces new characteristics to pop up from time to time. Norn DNA can be exchanged online with other players, adding to the gaming experience, and the game's creators have a web site dedicated to assisting in these exchanges.

This ability to create new characteristics has proven to be one of the most popular and interesting parts of the game, and has led to developments in the Norns that have surprised even the programmers at Cyberlife. At one of the 1997 CGDC roundtables, CREATURES producer Toby Simpson reported that they had received a DNA strand from a player whose Norn seemed "smarter" than the others in the player's world. Examination of its "brain" showed that it had indeed mutated, spontaneously developing additional "lobes" (processing centers) for its AI to work with.

HALF-LIFE

Oidian System's CLOAK, DAGGER, AND DNA is another game making use of GAs, using them to produce smarter opponents. A Risk-like game of world conquest, the heart of CDDNA uses GAs to guide the computer opponent's decisions. The game comes with four different AI opponents, each of which has a strand of digital DNA containing rules governing its strategy in the game. As each AI plays, a record is kept on how well it did in every battle. Between battles, the user can pit the AIs against each other in a series of tournaments; the winners survive to provide stronger and more capable computer opponents over time. The player can edit a given AI's DNA if so desired, ultimately building a library of opponents that are theoretically adapted to each player's unique playing style.

It's worth noting that the natural adaptive abilities of technologies such as GAs are giving developers a tool that can help them tune an AI during development. Several developers in our CGDC roundtables either mentioned that they had used genetic algorithms to help tune their AIs, or were considering it.

AI tuning is always somewhat problematic; by the time a game is near enough to completion that tuning can be done meaningfully, there can be hundreds of parameters that can affect the AI's style of play. Testing every combination is an impossible task, especially given the often short amount of time that, by that point, remains in the development cycle. Using GAs to tune an AI involves making hundreds of runs of a game using various parameters for the computer opponents, and then using genetic algorithms to tweak those parameters from run to run to favor those AIs that perform better. Over time, this method can test out many more AI variations than an individual developer ever could, allowing him or her to focus on particular configurations that work poorly and to select a more capable suite of AIs for the final product.

The drawback to using GAs and neural networks in games is that they require a considerable amount of time to converge to any significant degree. Both CREATURES and CDDNA demonstrate this. Each requires dozens or hundreds of AI generations of evolution before the player begins to see significant behavioral changes. CDDNA solves the problem by providing the player with an offline testbed in which AI strains can wage war against each other in a batch-processing mode. Players can interrupt the process at

any point and move the current champions into a stable of computer opponents. CREATURES makes the convergence and training process part of the game itself, involving the player in the breeding, raising, and education of the Norns. In this way, the enormous load that a neural network places on the CPU can be handled without adversely impacting graphics or game play. (Cyberlife states that each Norn requires roughly five percent of a CPU's cycles.)

ARTIFICIAL LIFE (A-LIFE). A-life is more a description of a desired result — a realistic emulation of biological behavior — than of a technology, per se. A-life is nothing new. The classic showcase in this field is the well-known LIFE computer program, but only recently have developers begun looking to A-life to build better game AI. The aforementioned flocking behavior is an example of A-life that was implemented using a simple rules-based approach, while CREATURES uses genetic algorithms to achieve its organic behavior.

REBEL MOON RISING

A-life technologies offer a way to build a better and more believable environment for games. They're particularly powerful when used to control the actions of NPCs in online role-playing games. For instance, very few of ULTIMA ONLINE's paying customers want to play a village blacksmith when they could choose to be powerful knights and sorcerers, and yet village blacksmiths are an essential part of the gaming experience. ULTIMA ONLINE also uses A-life to control the spawning, migration, and activities of wandering monsters and other wildlife. Parameterized characteristics allow different species to behave differently, so wolves have a predisposition to hunt in packs in forest regions while dragons tend to be solitary and dwell around mountains. This ecological system leads to a relatively realistic environment in which players can adventure freely. (To some extent, this realism was later compromised in the interests of game play, but that's a topic for another article.)

Flocking, mentioned earlier as an A-life technology, has found its way into some of the more recent game releases. HALF-LIFE and GT Interactive's UNREAL (http://www.gtinteractive.com/unreal) both use flocking to control the movement of groups of fish, birds, and monsters to create a more realistic and natural environment. Players, in return, have noticed and commented on the fact that these games don't feel as canned as previous first-person shooters. Players feel as though they're interlopers in a living, dynamic environment, rather than stalkers going from room to room slaughtering monsters.

EXTENSIBLE AIS. While many games over the years have allowed users to modify various game-related parameters, only recently have developers begun to let players alter and extend their computer opponents in a more direct fashion through scripts and code plug-ins. The most common example of this is, of course, the QUAKE's Bot phenomenon. When id introduced the world to the QUAKE scripting language, Quake C, it radically changed the level of interactivity expected in games. Players were able not only to build their own levels and fill them with monsters, but actually to specify how the monsters would act in some situations. If players didn't like the monsters that came with the game, they could build their own — and they did. A whole subculture of Bot builders sprang up on the Web, and players have developed a surprising variety of highly capable computer opponents.

More recently, UNREAL was released with Unreal Script, a robust programming environment that allows extensive customization of the game by the players. Developed by Tim Sweeney, Unreal Script provides a higher-level interface to the API controlling the computer opponent. This obviates the need for detailed programming statements in favor of more general directives such as, "Run forward two seconds, fire a weapon, find a new attack spot, run to it and evaluate, and create a new attack strategy," Players, in turn, have been using this capability to develop a wide variety of Bots, from deadly Terminator NPCs to companions who assist the player in various ways.

Grimmware's WISDOM OF KINGS — THE TWILIGHT WAR (http://www.grimmware.com/Games/WisdomofKings) take this concept one step further. Due to be released towards the end of this year, WOK (a real-time strategy game in the AGE OF EMPIRES vein) will break out the AI for each unit type into stand-alone .DLL files that players can rewrite or replace. The developers intend to expose the API completely to the player, providing the source code for the AI shipped with the game so would-be AI developers will have the option to start with the factory-built opponents or build their own from scratch. Grimmware will provide documentation and support via the company's web site. Activision plans a similar approach to its upcoming title CIVILIZATION: CALL TO POWER (http://www4.activision.com/ games/civilization).

Providing hooks in a game to support an extensible AI isn't easy, however, and while the trend is growing, developers at our roundtables agreed it's not likely to become widespread. The groups were split on whether extensible AI is even worth the effort, and with good reason. Significant technical challenges are involved (Do

we provide hooks through raw code or a scripting interface? How much control do we give players? How do we prevent hackers from writing AI module viruses that can wipe out a hard drive?), which require careful design from the beginning of the game's development cycle. The development costs of such an AI engine can quickly approach the level of the 3D engine, with arguably less significant impact on sales. Building in this level of interactivity and extensibility also presents developers with another dilemma: they risk exposing proprietary technologies to the scrutiny of competitors.

The Future of Game AI

What technologies are on the horizon that might be useful in games? Will good AI continue to increase in importance, or will multiplayer eventually render it moot for all but the most disconnected players? Do consumers really care about good game AI, or are they more interested in the latest 3D special effects? Would dedicated hardware (à la 3D hardware accelerators) be useful?

As might be expected, these topics elicit heated debates. Developers at the roundtables weren't sure where the next big advance in game AI might come from, though they did offer their opinions. Most people felt that they would slowly move away from a hard-coded, rules-based approach toward more flexible AI engines based on fuzzy logic and neural networks. While these technologies are presently prohibitively expensive for most kinds of games, widespread opinion was that learning from and adapting to the players' style would become a more important feature over the next few years.

One technology that many developers agreed would be a significant advancement is speech recognition. Until recently, speech recognition was largely the subject of academic and military biometrics research, but lately speech recognition APIs have begun to trickle down to PCs and are now attracting the attention of game developers. Already in limited use in a couple of flight simulators, speech recognition will be an option in a handful of first- and third-person shooters and real-time strategy games in development for late 1998.

One of these is Fenris Wolf's up-coming REBEL MOON REVOLUTION, a third-person real-time action game in which the player is the leader of a squad of space-suited marines. A key element of its AI technology will be the use of speech recognition as an interface between the players and the other members of their fireteams.

Robotics research was also cited as an interesting area of technology that might have applicability to game AI. Researchers at NASA and Carnegie-Mellon University have been working with autonomous robot designs for a number of years now, finding ways to integrate navigation, learning, and limited decision making for probes intended for planetary exploration. Because the distances are too great to permit real-time control by technicians on Earth, researchers have had to develop techniques that allow these robots to make their own decisions. There are some striking parallels between this work and what could be accomplished with unit AIs in real-time strategy games, NPCs in massive multiplayer RPGs, and so on.

In general, AI developers as a group felt that game AI would continue to see incremental and slow evolutionary advances, rather than revolutionary changes. We will continue to be CPU- and design-bound for the foreseeable future. ■

RESOURCES

The following is a *very* short list of resources for developers interested in digging more deeply into game AI technologies and research.

Books and Periodicals

Precious few books really discuss AI from a gaming perspective. Most are more academic-oriented texts that go into theory more than practice. Probably the best single comprehensive reference at present is *Artificial Intelligence: A Modern Approach* by Stuart J. Russell and Peter Norvig (Prentice Hall, 1995).

Additionally, author Bryan Stout ("Smart Moves: Intelligent Pathfinding," *Game Developer*, October/November 1996) is working on a book dedicated to game AI due out in early 2000. It's tentatively titled *Adding Intelligence to Computer Games* and will be published by Morgan Kaufmann.

Newsgroups

Several Usenet newsgroups focus on artificial intelligence in general and game AI in particular. A few of the better ones in terms of noise-to-content ratio are comp.ai.games, comp.ai, and rec.games.programmer.

Web Sites

There's probably no better resource for researching AI technology than the Web. Some sites I recommend include:
http://www.gamaustra.com — Gamasutra (sister site to *Game Developer*) maintains an online roundtable of game AI that grew out of the CGDC roundtables. Highly recommended.
http://www.concentric.net/~swoodcoc/ai.html — The author's page, dedicated to all things related to game AI. It includes links to other AI resources, archives of various Usenet threads, and a page focusing on games making use of particularly interesting AI technologies.
http://hmt.com/cwr/boids.html — Craig Reynolds is the Father of Flocking. His page is the best place on the Web to start research into the theory and technology behind flocking and similar A-Life technologies.
http://www.aracnet.com/~wwir/j&p.html — Nova Genetica is one of the best online references for all things related to genetic algorithms and genetic programming.
http://ai.iit.nrc.ca/ai_point.html — The Artificial Intelligence Resources page is maintained by the NRC/CNRC Institute for Information Technology; it's an excellent starting point for AI research.
http://www-cs-students.stanford.edu/~amitp/gameprog.html — Amit's Game Programming Page is crammed with information on path-finding algorithms and pointers to other AI resources.

THE ORC PROBLEM

BY SWEN VINCKE

Once upon a time there was a fierce orc named Kroxy whose goal was to retrieve all the gold the humans had taken from him. He couldn't really remember the actual theft, but he was sure that the gold was his. Being a master at orc politics, Kroxy soon convinced all the other orcs that the humans had plundered their villages, and they readied themselves for war. They didn't seem to be bothered by the fact that the humans didn't even know the orcs existed. Orcish logic allowed for that. So Kroxy led his army of orcs towards the land of the humans and eventually ran into a human army. This was a bit of a shock

Swen Vincke is still working on THE LADY, THE MAGE, AND THE KNIGHT *(LMK) and expects to be still working on it when he writes another article for* Game Developer *magazine. You can contact him at lar@larian.com*

SET 1.

$$\alpha = \left\{ \exists j, S_j \in S_{all} \mid D(S_j, m, n) > D(S_{optimal}, m, n) \ \wedge \ H(S_j, m, n) < H(S_{optimal}, m, n) \right\}$$

as there were clearly more humans than there were orcs, and Kroxy ordered his troops to assume the grand orcish defense position. Much to his dismay, the humans didn't attack. Instead they made camp and looked at the orcs through funny tubes. Kroxy called for the high shaman and demanded to know what the humans were doing. After consulting with the orcish gods, the shaman told Kroxy that the humans were probably counting the orcs. Upon hearing this Kroxy burst out in rage and yelled: "Well, in that case, we'll count them, too!" While the humans were entrenching themselves, the orcs undertook the great task of counting the humans, and eventually, Kroxy learned that there were many humans and not so many orcs. This worried Kroxy and he retreated to his tent, where he performed the ritual of Grand Orcish Insight. When he left his tent again, great magic had happened. Floating above every orc and human in sight was a little bar, indicating how strong each individual was. Now Kroxy demanded that the shaman provide insight as to which orc should attack which human, so that the battle may be followed by a great orcish feast, rather than a great orcish burial. The shaman sighed, and started thinking...

Brute Force Doesn't Work

Perhaps I got carried away with this orc story, having played WARCRAFT II again, but the problem challenging Kroxy is quite an interesting one, especially for a game developer. Let's state Kroxy's problem a little more formally: if you have m friendly units fighting n enemy units, how do you align the friendly units in such a way that the damage they inflict is maximal, while trying to minimize the damage they receive? If you can calculate the answer to this question quickly enough, then you've solved an important part of computer opponent artificial intelligence.

Unfortunately, this problem is not easy to solve. Using a brute-force approach, you would try all the fighting combinations, qualify them according to some function, and select the best out of the set. Because we want to imbue our AI with a sense of strategy, it might be wise in some cases to gang-up two units on one opposing warrior. Therefore, for each of the m friendly units, there are n attack possibilities to choose from. This means that there are n^m possibilities — an exponential function. Even a modest fight, for instance eight against eight, would entail evaluating 16,777,216 combinations! Obviously, a better way of solving this problem is needed.

Defining the Problem Mathematically

This paragraph could get confusing, so bear with me — I'm going to represent this problem mathematically. What we're trying to do is minimize one function while maximizing another function. Say $D(x_i)$ stands for the total damage a unit i inflicts on a unit x_i, and $H(x_i)$ stands for the total amount of damage a unit i receives from a unit x_i. The actual damage can depend on multiple factors such dexterity, strength, armor, and so forth. Let's define the triplet (S,m,n) as the representation of a deployment situation. (S,m,n) represents the set $\{ x_i \}$ where i=0,1,2,3..,m-1, and where the total amount of enemy units is n($0 \le x_i < n$). Such a set $\{ x_i \}$ contains all the information we need to know which friendly unit needs to attack which enemy unit. We can then define $D(S,m,n)$ to be $\sum_{I=0,1..m-1} D(x_i)$ which is the total damage inflicted under a particular situation S. We can do the same for $H(S,m,n)$ so that $H(S,m,n)$ represents the total amount of damage received under a particular situation S. Be careful though. It's possible that we are not attacking all of the enemy's units, and those units definitely won't stand idly around. So we need to account for the possibility that, if we attack only a subset of the enemy unit, the remaining enemy units will also inflict damage upon us. We can say that this extra damage can be represented by some function $ED(S,m,n)$. Therefore, the definition of $H(S,m,n)$ is $\sum_{I=0,1..m-1} D(x_i)$ + $ED(S,m,n)$. The actual definition of $ED(S,m,n)$ depends on the game. It could be something like the sum of all the remaining damage points the enemy has minus the amount of remaining units multiplied by the average armor of the friendly units. Because there are maximally n^m situations, the set of all situations is $S_{all}=\{S_I\}$, I=0,..., n^{m-1}. We are now looking for a situation $S_{optimal} \in S_{all}$ so that Set 1 is empty.

Clearly, many situations could be optimal. Some situations might have the property that they inflict huge amounts of damage while receiving huge amounts of return fire, where others might inflict a low amount of damage while receiving an equally low amount of damage in return. The first scenario, for instance, could happen if you put your strongest units against the enemy's weakest units, and your weakest units against the enemy's strongest units. This introduces another problem, because your strongest units might finish off the weakest enemy units quite quickly. Your strongest units should then come and help your weaker units. This sort of redeployment might give you an edge, but it also might be a bad idea. So a situation that looks optimal at first might leave you with too many casualties, and in subsequent battles, the tide may turn. We can express this formally by introducing a function such as $B(S_{original},m_{original},n_{original})$, which yields a new triplet $(S_{new},m_{new},n_{new})$, expressing the situation after one round of fighting. A fight $F(S_0,m_0,n_0)$ can then be represented as the set $\{(S_0,m_0,n_0), (S_1,m_1,n_1),..., (S_{k-1},m_{k-1},n_{k-1})\}$ where $(S_{k+1},m_{k+1},n_{k+1})=B(S_k,m_k,n_k)$. It's possible that some of the enemy units have died after one round, in which case some opposing units have become free. To keep matters simple, we will say that these free units will help the

SET 2.

$$\alpha = \left\{ \exists j, S_j \in S_{all} \mid D_{fight}\left(F\left(S_j, m, n\right)\right) > D_{fight}\left(F\left(S_{optimal}, m, n\right)\right) \wedge H_{fight}\left(F\left(S_j, m, n\right)\right) < H_{fight}\left(F\left(S_{optimal}, m, n\right)\right) \right\}$$

friendly units most in need. We do this by introducing the function FD(S,*m*,*n*), just as we introduced ED(S,*m*,*n*) for the H(S,*m*,*n*) function. If we define $D_{fight}(F)$ as $\Sigma_{I=0,1..k-1}$ $(D(S_i,m_i,n_i) + FD(S_i,m_i,n_i))$ and do the same for $H_{fight}(F)$ (taking ED() into account), then we can say that our goal is to find a situation $S_{optimal}$ so that Set 2 is empty. In other words, the total damage inflicted over a fight has to be maximal, and the total amount of damage received has to be minimal. This definition doesn't say anything about winning or losing, because in some cases it's impossible to win. Still, at least we try to hurt the enemy as much as possible. Another thing to notice about the goal definition is that it is assumed that if we attack a unit, it will defend itself. While this assumption is definitely not always valid, at least the AI will have a way of assessing the situation. However, saying that our goal is to make α empty is not handy, so let's reformulate the function. If we define FITNESS(S,*m*,*n*) to be $H_{fight}(F(S,m,n))$-$D_{fight}(F(S,m,n))$, then we can state that the situation $S_{optimal}$ for which FITNESS($S_{optimal}$,*m*,*n*) is minimized is the situation that will make α empty. It's possible that FITNESS(S,*m*,*n*) could become negative, but that's not really a problem. We actually want FITNESS(S,*m*,*n*) to be negative because that means we're dealing out more damage than we're receiving.

The Orc's Grand Algorithm

So, how do we find $S_{optimal}$? As you can guess, the search space is quite complex. When dealing with large and complex search spaces (say 120 units attacking 140 enemy units), I often turn to approximation algorithms. In this article, I'll look at how we can use a genetic algorithm as an approximation algorithm for solving Kroxy's problem.

As the name implies, genetic algorithms (GAs) closely follow the biological concepts of evolution, but luckily not too closely. Nature takes millions of years to solve problems — viewed in this context, its methods would be of little practical use to us. You should be aware that there is no guarantee that GAs will find the optimal solution in a finite number of steps. As I said, these algorithms are an approximation technique — often, a good and fast approximation is better than a slow and accurate solution, especially when it comes to games.

Like most things in nature, the concept underlying GAs is surprisingly simple. The general idea is that you code instances of your search space into chromosomes. (For the purpose of understanding GAs, you can regard chromosomes as sets of genes.) You then let the chromosomes evolve just as nature does. Hopefully, evolution will form new chromosomes that will contain encoding of the solution to the problem.

More specifically, you start with a population of random chromosomes and let them evolve by interbreeding. Every chromosome contains a set of genes (the genotype) describing the attributes of the chromosome (the phenotype). When two chromosomes breed with each other, they generate new chromosomes. These new chromosomes contain a mix of the genes of their parents. Because of mutation, the children don't always contain exact replicas of the parents' genotype. By replacing the parents with the offspring, evolution occurs. Some chromosomes aren't very successful at surviving in the environment; their genotype dies off, and they cease breeding. Others are quite successful, and the most successful parts of their genotype is shared more and more by all the chromosomes in the population. At a certain point, you get a chromosome that is so successful that it can't be improved anymore. This is the chromosome for which evolution was looking. It's also the chromosome that contains our solution.

WHAT'S IN A GENE? The first step in creating a genetic algorithm is deciding what information the genes are going to contain, or what the phenotype of each gene is going to be. In nature, the phenotype determines an individual's traits. In our orc problem, it's going to determine the optimal troop deployment. By letting every gene correspond with a friendly unit, and letting the actual value of that gene being the enemy unit we're going to attack, we have an adequate representation that fits into the formalism that we set up earlier. A typical chromosome's genotype could look like (1,3,1,4). Thus, its phenotype says that friendly unit 1 is attacking enemy unit 1, friendly unit 2 is attacking enemy unit 3, and so forth. In terms of the previously introduced notation, we have S={1,3,1,4}.

INITIALIZING THE POPULATION. The next step is generating a population. Every individual in the population has a corresponding chromosome, and each chromosome has its own genotype. The initial population must contain a sufficient number of different genotypes. If a particular genotype is over represented, it might take a longer time to get to a solution, especially if this genotype is of a bad quality (we'll get to the reason for this later on). We will generate our initial population randomly, but in some applications, the candidates for the initial population are carefully selected. The idea is to embed some knowledge in the genotypes before applying the genetic algorithm. While this sounds intuitively correct, research has shown that for most cases, it's better to select a random population.

THE FITNESS FUNCTION. When evaluating the quality of a chromosome, we need some kind of function that tells us how "good" a chromosome is. This function is called the fitness function. In problems of minimization, one chromosome is better than another one when its fitness is lower. For our problem, we've already determined such a fitness function. The previously introduced function, FITNESS(S,*m*,*n*),

FIGURE 1. *Selection mechanisms.*

$$\alpha = \left\{ \exists j, S_j \in S_{all} \mid D\left(S_j, m, n\right) > D\left(S_{optimal}, m, n\right) \ \wedge \ H\left(S_j, m, n\right) < H\left(S_{optimal}, m, n\right) \right\}$$

has the property that its minimum represents the best solution, so we can use it to qualify a chromosome's fitness. It's important to realize that the quality and speed of a GA often depends on the quality of the fitness function. Sometimes, the fitness function is quite obvious, but sometimes it's nonintuitive and you have to perform some analysis.

SELECTION OR "DATING." To start the GA, we need to select which two parents are going to breed with each other. This is a crucial step, because parents that don't breed stand little chance of distributing their genotype to subsequent generations. Our main objective is to select two parents that will generate good children. Since we don't know what good children are, we have to make educated guesses. We can't just say that we'll take the best chromosomes we have at this point, because it's possible that we're converging toward a local minimum. (If this doesn't mean a lot to you, look up my article on the A* algorithm, "Real-Time Pathfinding for Multiple Objects," in the June 1997 issue of *Game Developer*. In it, I discuss the problem of local minima.) On the other hand, we could be converging toward the global minimum, so we have to carefully balance our selection so that the GA has a higher chance of selecting fit chromosomes while still retaining the option of selecting weak chromosomes. Numerous selection mechanisms exist, and I'll discuss three of the more popular mechanisms here.

The first selection mechanism is the so-called roulette selection (Figure 1A). Its name denotes the fact that it basically performs a linear search through a roulette wheel. Every slot in the wheel contains the fitness value of a chromosome. The GA sets a random target value that is a random proportion of the sum of the fitness values in the population. So, if we define the total fitness of the population as $T=\sum_{l=0,1..n-1} \text{FITNESS}(C_i)$ where n is the number of chromosomes in the population and C_i is the ith chromosome, then our target value could be αT, where α is a random value between 0 and 1. The selection mechanism steps through the population until it reaches the target value; the last chromosome in the summation is then selected as a parent. With roulette selection, fit chromosomes aren't guaranteed to be selected. Still, they do have a greater chance of being selected, because fit individuals will contribute more to the sum than less fit individuals. However, because ours is a minimization problem, this distinction is contrary to our goal: in our problem, the best chromosomes will contribute little, while the worst will contribute more. So we invert the summation. We sum 1/fitness, and our target value becomes $1/\alpha T$. Because you could be dealing with very large fitness values, it's sometimes a good idea to multiply everything in the summation by some large value, such as the sum of all the fitness values. This doesn't affect the search and avoids getting values for 1/fitness or $1/\alpha T$ that are practically zero.

The second selection method is simply random selection (Figure 1B). You select a random chromosome to be the parent. This method is said to be "genetically disruptive" because it doesn't take the parents' fitness into account. Still, you shouldn't discard this method as useless. In several cases, I've had much better results with random selection than with roulette selection, especially with crossover operators that suffer from premature converging to a local minimum (which I'll talk about shortly).

The third selection method is called fit-fit selection (Figure 1C). It selects the next fittest chromosome in the list of chromosomes. Because the population changes throughout the algorithm, not every chromosome will get a chance to breed; the fitness of the chromosomes relative to each other changes constantly. When you use fit-fit selection, the GA will have a tendency to converge rapidly toward a solution. The problem with this method is that one of the main advantages of

FIGURE 2. *The crossover operation.*

$$\alpha = \left\{ \exists j, S_j \in S_{all} \mid D_{fight}\left(F\left(S_j, m, n\right)\right) > D_{fight}\left(F\left(S_{optimal}, m, n\right)\right) \ \wedge \ H_{fight}\left(F\left(S_j, m, n\right)\right) \right.$$

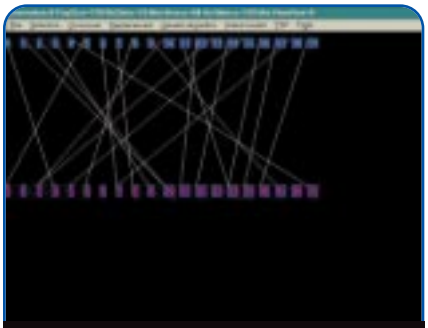


FIGURE 3A. *The initial situation for the predefined test problem with optimal fitness 0. The lines represent which unit should attack which unit.*

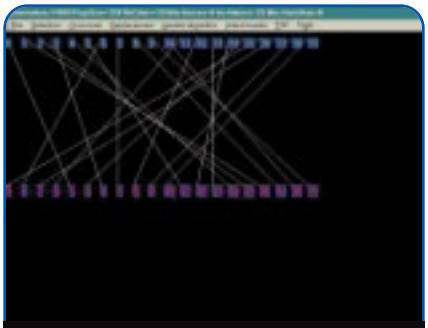


FIGURE 3B. *This is the optimal situation for the predefined test problem that was reached after 15,000 iterations of the GA.*

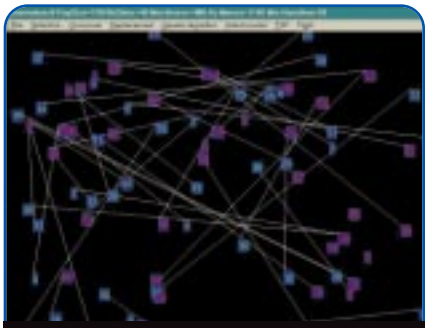


FIGURE 3C. *This is the initial situation for a hard random 40 units vs. 40 units problem. Note that the fitness value indicates that we're loosing.*

GAs (using the genetic diversity among the population to explore different areas of the search space) isn't exploited as much as you would want.

ON THE NATURE OF BREEDING. After the GA has selected two parents for breeding, the parents need to get down to it. This brings us to the subject of crossover operators. Crossover operators are one of the most actively studied aspects of GAs. The crossover operator's job is to determine how two parents are going to generate their offspring, usually two children. The operator that we'll use here works as follows. First, you define two arbitrary crossover points, a and b, on the parent chromosomes. For child 1, all the genes that lie outside the interval [a,b] are copied from the first parent, and all the genes that lie inside the interval [a,b] are copied from the second parent. For the second child, the same thing happens, but the genes inside the interval [a,b] are copied from the first parent. Thus, two new children are created with a genotype that inherits from both parents (Figure 2).

Ideally, if a crossover operator made all the right decisions, a GA would yield a perfect solution in one generation. However, the amount of time that this would take would be based on an exponential function, which isn't what we want. Still, the search can be helped by embedding some knowledge into the crossover operator. This has to be done with care, because as with the selection procedure, we don't want to guide the search into a local minimum. Unfortunately, I haven't found a good general heuristic for this specific problem yet. This has a lot to do with the B(S,*m*,*n*) function, which is very game-specific. Later on in this article, I'll present an example of embedding knowledge into a crossover operator for another problem that has an easier fitness function. I can provide a general guideline though. Inserting local knowledge into a crossover operator usually boils down to applying some function to the current chromosome. This function yields a child that has a fitness value that is at least as good, but preferably better than, the parent chromosome. I also advise having some measure of randomness, even if knowledge is added. As we'll see later on, techniques such as a controlled random search usually work well. If somebody finds a good general crossover operator for Kroxy's problem, let me know.

After a crossover operator has been applied to generate the two children, mutation can happen. Chances are relatively low (one to three percent is typical) that mutation will occur, but you will need to include mutation because it inserts new genotypes into the population. Especially in cases where the genetic algorithm is converging around a local minimum, mutation can be the thing that saves the day. One way to mutate chromosomes is to choose two genes randomly, and then exchange them with each other. This is known as an exchange mutation. In the crossover operator described in the previous paragraph, you apply an exchange mutation after the children have been generated.

FAMILY RESPECT. Once you've selected two parents to breed and you've obtained their children, you must determine how these children are going to be integrated into the population. This is the issue of replacement. Replacement decides which chromosomes to kick out of the population, a matter not to be taken lightly. Your choice of replacement technique should depend, to a certain degree, on your choice of selection technique. Following are three ways of implementing replacement.

In what's known as weakest parent replacement, the child will only replace one of its parents if it's stronger than the parent. When used in conjunction with a selection method that selects both fit and weak parents, this form of replacement will continue to improve the overall quality of the population. If the selection mechanism discriminates weak parents, they will never be replaced. So the part of the population that matters decreases in size.

Both parent replacement is a replacement technique that simply replaces both parents. Using this replacement method means that every chromosome will only breed once. If your selection

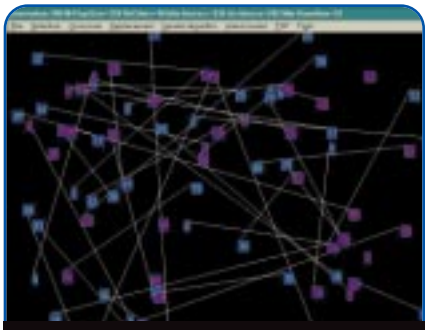


FIGURE 3D. *This is the 40 vs. 40 situation after a little more than 10,000 generations. The definite defeat has been turned into a definite victory.*

method favors fit candidates, this replacement technique might be killing your best chromosomes.

Weakest chromosome replacement simply replaces the two weakest chromosomes in the population, as long as the children are not weaker than the weakest chromosomes. This method will assure that there is rapid convergence. By now, you should understand that rapid convergence might quickly get you to a local minimum; getting out of it under this scheme is rather difficult. Weakest chromosome replacement works well in large populations.

KNOWING WHEN TO STOP. As stupid as it may sound, knowing when to stop a GA can be very difficult. Ultimately, we want the GA to stop when we've found the best solution. But of course, we don't know what that best solution is. What you can do is look for the theoretical lower bound, and then stop the GA when it approaches that bound. Alternatively, you can just stop the GA when your game is running out of time. This is one thing that's very nice about GAs: you can stop one whenever you want, and just use the best result that it's come up with thus far. That's what I always do.

SOME RESULTS. I've developed a small test program that generates random fight situations and runs the GA over it. You can see some of its output of it in Figure 3. If anything, the results from the test program are quite satisfactory. For a predefined test situation of 20 units vs. 20 units, all of which have the same amount of hit points, armor, and attack values, the GA took about 15,000 generations to find the optimal solution, which is fitness 0. For a more difficult 40 vs. 40 problem with units that have random attack and hit point values, the GA took only 10,000 generations to transform the situation from FITNESS=408 to FITNESS=–938, turning a definite loss into a definite victory. For these and all subsequent test results, I used roulette selection in combination with weakest parent replacement, a mutation rate of three percent, and an initial population size of 120 (Figures 3A-G).

EXTENDING THE MODEL. Until now, we've evaluated the orc fighting problem with a relatively simple fitness function. I've made many assumptions about the capabilities of the units, some of which are obscenely naïve. The most obvious fault in the model is the fact that position wasn't taken into account. When units actually need time to get somewhere, $S_{optimal}$ is likely to be very different from $S_{optimal}$ for units that can teleport themselves. You can see this very clearly in Figures 3F-3G, in which some units are traversing huge distances. Luckily, all we have to do is adapt the fitness function a bit so that it takes position into account. The GA will then solve the problem while taking position into account. In the implementation of the battle transformation function $B(S,m,n)$, it's possible to add up all the distances between the units that attack each other. If we define this distance as $DIST(S,m,n)$, then we can redefine the fitness function as $\omega_1 DIST(S,m,n) + \omega_2(H_{fight}(F(S,m,n))-D_{fight}(F(S,m,n)))$. ω_1 and ω_2 can be anything and depend on how important distance and the exchange of damage are. The results of running the demonstration program under this model can be seen in Figure 3E-G. You'll notice that the number of generations necessary to come to good results is a little higher than before, which reflects the increased complexity of our search space.

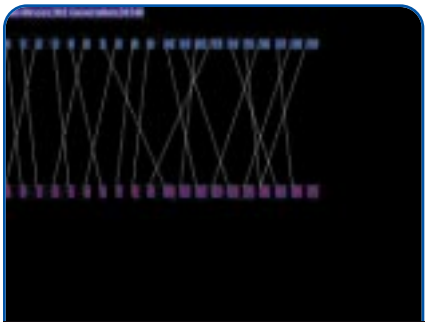


FIGURE 3E. *This is an intermediate result for the predefined test problem using the extended model in which position matters.*

FIGURE 3F. *This is the optimal situation for the predefined test problem. Note the number of generations necessary to reach this result. It's often better to stop searching when you're just in the right neighborhood when using an approximation algorithm such as a GA, as figure 3E shows.*

And Now for Something Completely Different

Clearly, using GAs to solve Kroxy's problem is a good idea. We can interrupt the GA whenever we want, and even after a modest number of generations, the results are pretty good. I'm using this approach in THE LADY, THE MAGE, AND THE KNIGHT (LMK), our upcoming role-playing game, and can tell you that the AI has definitely become better because of it. I also use GAs in LMK's character behavior AI. At some point during the development of LMK, we decided that it would be a nice touch to have the characters turn out the lights in their houses before they went to sleep at night. This was easily done, but for characters who had large houses, a simple algorithm directing them to go to the closest light and turn it out showed a definite lack of intelligence. The characters ran around mindlessly, turning out the lights in seemingly random fashion. Unfortunately, getting them to follow an intelligent path to turn out the lights meant that we had to solve the travelling salesman problem, in real time. A GA

FIGURE 3G. *This is random 40 vs. 40 situation under the extended model. As you can see, here too the definite loss was transformed into a definite victory, while the troop movements make sense.*

engine running in the background turned out to be just the thing to solve our problem.

THE TSP. The travelling salesman problem (TSP) is a classic permutation problem. In the TSP, a salesman has to find a route that visits each of *N* cities once and only once, and that minimizes the total distance traveled. It's easily stated, but it's extremely hard to solve. To determine its complexity using a brute force approach, represent each city by a number, and write the route as a sequence of numbers. The problem then boils down to finding the number sequence with the shortest total distance. The total distance is the sum of all the distances between every pair of cities, and is the distance between the last and the first cities in the sequence. There are *N*-1 possibilities for choosing the first city (assuming a starting city value of 0), because the starting point is arbitrary. When you select one of them, you get *N*-2 possibilities, and so forth. The total number of possibilities is therefore (*N*-1)*(*N*-2)*…*2*1 or (*N*-1)!. So, the brute force approach has an algorithmic complexity of O(*N*!), which is exponential. An enormous amount of research has been dedicated to the TSP, and the best algorithm known so far has a complexity of $O(2^n n^2)$, which is still exponential and not very good even for small *n* (for *n*=20, you're already at over 400 million steps).

To solve the TSP with genetic algorithms, we only have to modify a few aspects of the GA that we've developed so far. We can represent every city by its coordinates in space and assign an index to it, starting with 0 for city 0 and continuing until we are at city *N*. We consider the index of the city as the value that we put into a gene, and the coordinates of the city as the gene's phenotype. If we have five cities, all chromosomes will contain five genes (as each chromosome represents a possible path), its phenotype will document the route of the salesman (starts at city 0, proceed to city 2, and so on), and a sample chromosome might have the value (0,2,4,3,1).

We again generate the population randomly, but we do have to make sure that every chromosome has the same number of genes and that all of the genes are different. This is trivial to do, but it can take some extra processing time, especially for large populations. The fitness function can be the summation of the total Euclidean distance between every pair of cities. As in Kroxy's problem, we want to minimize the value of the fitness function. Because the fitness function is accessed a lot, it's better to precompute all the distances between every pair of cities. You can store this in a matrix where the indices of both rows and columns refer to the city index. The value of each cell is then the distance between the cities represented by row and column. As a bonus, if the direction doesn't matter, you only have to compute half of the total distances represented in the matrix.

The selection and replacement methods don't need to be changed, but we have to use a different crossover operator. The problem with the crossover operator that we used in Kroxy's problem is that it didn't care if two genes had the same value. One crossover operator that was specially invented for solving the TSP is the partially mapped crossover (PMX) operator. PMX uses two crossover points, x and y. All the genes in the two parents between x and y are swapped. Because all genes have to be unique in the TSP (that is, no city can be represented twice in a chromosome), we have to apply the following procedure:

If a gene inside the crossover interval [x,y] lies in [0,x[or]y,n] (where n equals the number of genes), copy from its direct parent the gene lying in the same position to the faulty gene outside the crossover interval in the child chromosome (the "direct parent" is the parent from which the child copies all the genes that are not affected by the crossover operator). Repeat this process as long as there are equal genes in the child chromosome.

The Edge-Recombination Operator

When applied to the TSP, the PMX provides satisfactory, but not optimal, results. PMX can be improved by embedding local knowl-

FIGURE 4A. *This is the initial situation for a random 40-city problem. The red dots are the cities, and the lines show the path through the cities.*

TABLE 1.

Suppose we have two genotypes p1= (2,3,5,4,1,0) and p2=(2,1,3,0,5,4).
Then their edgemaps look like

Gene value	Neighbors	Gene Value	Neighbors
0	1,2	0	3,5
1	0,4	1	2,3
2	0,3	2	1,4
3	2,5	3	0,1
4	1,5	4	2,5
5	3,4	5	0,4

The edge map tells us what the neighbors of a particular gene are (or adjacent towns in the case of TSP).
The combined edgemap for p1 and p2 then looks like

Gene Value	Neighbors
0	1,2,3,5
1	0,2,3,4
2	0,1,3,4
3	0,1,2,5
4	1,2,5
5	0,3,4

The combined edgemap contains the combined edge relationships from both parents.

LISTING 1. *Edge Recombination Operator algorithm.*

Give two genotypes p1 and p2 of length *N*, and their combined edge map M. To obtain c1, the child genotype, do the following:

1. Copy the gene g from first position in p1 to the gene in the first position in c1. Remove this gene value *g* from the edge map
2. Let I be the first gene value in genotype c1
3. If all *N* gene values have been added to the child c1 then exit and return c1.
4. If for the *i*th gene there are no more entries in the edge map, then randomly choose a gene value *g* to be added to c1, where *g* has not already been added to c1. Add *g* to the next consecutive gene location in c1, remove *g* from the combined edge map M, set I to *g* and go to Step 3.
5. Choose the *j*th element from the *i*th entry in the combined edge map where the *j*th entry in the combined edge map contains the minimum number of elements. If several elements have this property, choose one of them randomly. Once we have chosen *j*, we then add *j* to c1 at the next consecutive gene location. We then remove *j* from the combined edge map, assign *j* to I and go to Step 3.

edge in the algorithm. If two cities lie very close to each other, in most cases it would be a bad idea to replace one of these cities with one that is lying far away. The edge-recombination operator algorithm (ERO) addresses this issue. It's known to have solved several moderate to large instances of the travelling salesman problem.

The idea behind the ERO algorithm is to build a child genotype based on the combined edge relationships from both parent genotypes. The advantage to this approach is that the parents might already know something useful. The ERO algorithm makes extensive use of something called an "edge map." The edge map tells us what the neighboring genes are (Table 1). Note that the same gene necessarily appears at multiple entries in the edge map. The final edge map combines the edge maps of the two parent chromosome. The actual ERO algorithm is described in Listing 1. You'll immediately see that the algorithm only generates one child, but we can easily obtain a second child by exchanging the two parents.

The important part of the ERO algorithm is Step 5. There, we visit those towns with fewer adjacent edges earlier than those with more edges, because a town with fewer adjacent edges is likely to cause difficulties if it's chosen later on in the town selection process. When we're near the end of the algorithm, there's a high probability that there are no more adjacent tows in the edge map. This causes the algorithm to perform mutation by resolving to Step 4.

The ERO algorithm does what is called "controlled random exploration." When the chromosome is long enough, a reasonable number of random choices are made. These choices represent different ways of combining the parents without introducing new foreign edge relationships into the child. This combination causes the algorithm to converge quite rapidly towards a solution. Be aware however, that this strong point of ERO is also a weak point, as we might fall into a local minimum. One way to get better results is to switch to PMX when you're in a minimum.

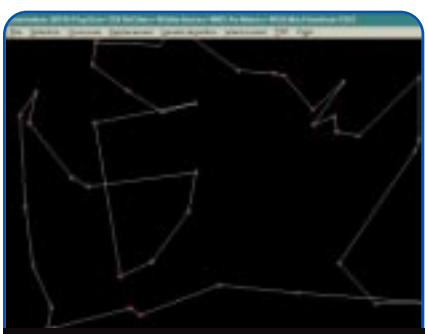


FIGURE 4B. *This is the situation after 30,000 generations using the ERO operator.*

FIGURE 4C. *This is the situation after 30,000 generations using the PMX operator. You can clearly see that the ERO operator outperforms it.*

Some of the results of applying the GA to the TSP are listed in Figure 4. As you can see, it does pretty well. Indeed, in LMK you get the impression that some designer has told the characters how they should turn out the lights. I could go on and give you more examples of how to solve hard search problems using GAs, but by now I hope you understand their potential. In many problems, GAs are my algorithms of choice. They are very easy to implement, they can be stopped at any time, and often still give adequate results — quite an important feature for a game. Generally, if you have a difficult search problem that you don't know how to solve, you can usually find a way of tackling it with a GA.

The only problem with GAs is their speed. Do they get results fast enough for use in real time? The answer depends on the game, and just for what exactly you're trying to search. In LMK, we don't use GAs for split second decisions, but we do use them for decisions whose impact will stay in place for some time. Such decisions can usually be calculated in the background. If you can run a GA engine in the background, I really don't see any reason why you couldn't implement them in your game, too. Of course, if you don't have to solve problems like TSP, don't mess with GAs. But if you are facing these kinds of problems, consider GAs. Thirty-thousand iterations vs. 2^{40} times 40^2 isn't that bad. ■

FOR FURTHER INFO

For those of you who are further interested in GAs, there's a wealth of information out there. Try out the comp.ai.genetic newsgroup or The Hitch-Hiker's Guide to Evolutionary Computation, which can be found at http://www.cs.purdue.edu/coast/archive/clife/FAQ/www/. Alternatively, just do a search on genetic algorithms. You'll find that there is enough to keep you busy for quite some time.

Anatomically Correct Character Modeling

by Stefan Henry-Biskup

A good character model has to fulfill two requirements: it has to look great and it has to animate well. The aesthetic quality and technical functionality of your model are deeply intertwined. Aesthetically, you want a model that really captures the details of the original design. Technically, you want to maintain a model's character traits as it animates.

As character modelers, we're carrying on a figurative, sculptural tradition that is as old as art itself. And computer graphic art represents some stunning figurative sculpture — just look at the beautiful, high-resolution versions of characters from TEKKEN 3 and MORTAL KOMBAT 4. While we digital sculptors seem to be creating better-looking characters, creating characters that animate well is a new challenge that can make or break a game.

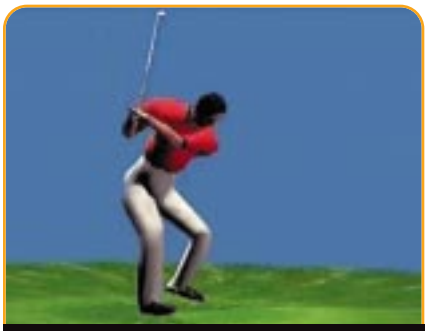


FIGURE 1. *This golfer's thighs are too long and his shoulders bulge.*

Creating a model that can look good in the da Vinci pose and perform all of the exotic movements that today's games require is a daunting task. Fortunately, we can take cues from human anatomy to solve the puzzle.

A character model and skeleton is a large and complex hierarchy. As such, problems that exist at or near a model's base (root) will propagate throughout the entire figure. Conversely, changes to the root can propagate improvements as well.

Case Study: JACK NICHOLAS 5

When Accolade began working with Eclipse Entertainment to create JACK NICHOLAS 5, we encountered some major problems with the golfer's animated appearance. The model was attached to an animated skeleton, which was driven by motion capture data. However, when when the model animated, problems cropped up. The golfer's posture was very stiff and unnatural, the shoulders were ballooned out, and the thighs looked too long (Figure 1). A great deal of time was spent tweaking the vertex attachments, applying bulge angles, and editing link parameters in an effort to fix the model's animated appearance. We tried to improve the character's posture by adjusting the bones of the back, but that threw off the rest of the motion, causing the golfer's club go into the ground during a swing (the hierarchy effect). Most of our fixes related to the attachment of the mesh to the skeleton — essentially changes to the surface portions of the model. But the source of the problems was actually inside, in the skeleton itself.

The problem lay in the structure of the skeleton and its positioning within the mesh, not in the vertex assignments. The joints of the hips and lower back were coplanar in the z axis, forming a flat horizontal line (Figure 2). As such, the lower back rotation occurred in the hip area, creating a stiff posture that lacked the natural arc of the spine. This configuration placed the skeleton's root too low in the mesh, and because the rest of the joints were children of this root, the problem was propagated on to them.

Stefan Henry-Biskup has been in the game business for six action-packed years. He is currently a senior artist at Accolade working on SLAVE ZERO.

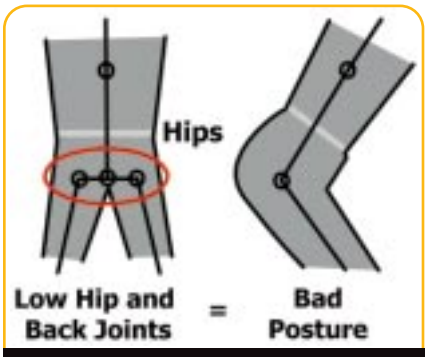


FIGURE 2. *Coplanar joints in the hip area causes unnatural movement.*

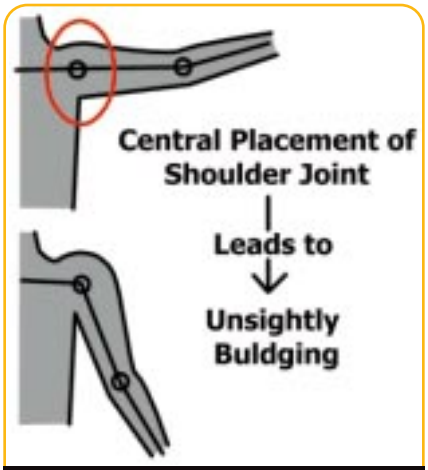


FIGURE 3. *The root of this limb is placed too low.*

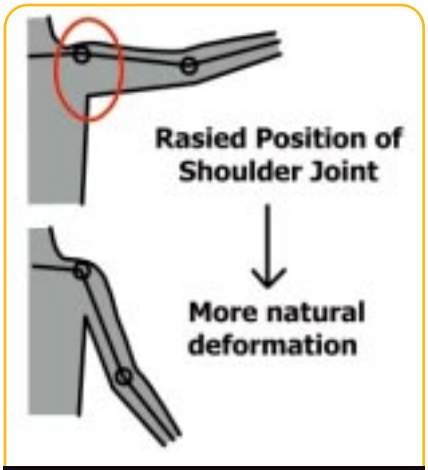


FIGURE 4. *Correct placement of the shoulder joint.*

The low placement of the root turned out to be the cause of our knee and shoulder problems (Figure 3), and in fact affected all joints to some degree. To solve it, we reprocessed the motion capture data for a more appropriate skeleton, and then modified the model with respect to the new skeleton (Figures 4 and 5). With a new skeleton and model in place, we greatly reduced the amount time needed to tweak the vertex attachments, and the appearance of the animated model improved greatly (Figure 6). (An .AVI file of the golfer before and after are available at http://www.gdmag.com.)

The problem with the golfer in JACK NICHOLAS 5 illustrates the importance of getting skeletal positions correct in the beginning. To achieve this goal, I suggest turning the usual production sequence upside down, building the skeleton before you build the mesh. You can then build the model's surface by aligning your geometry to the appropriate bones of the skeleton as you go. This technique is similar to the way in which a sculptor uses an armature when creating a figure in clay. It lets you concentrate on orienting the surface contours of the body to the bones as you build them, and then create appendages along those naturally posed bones. So the first order of business is to get the skeleton into position, and this is where character sheets come in handy.

What's a Character Sheet?

The character sheet was born in the 2D animation industry as a reference document to help animators draw characters. A character sheet is a guide to human proportions, and it can save 3D modelers a lot of time in creating characters. You've probably seen a character sheet before; it usually shows at least the front- and side-views of the character as well as any number of expressions and costume details that the designer wants to note. I scan my character sheets, and then map the scanned images onto polygons in my 3D package and work directly over that texture in the orthographic viewports.

This is the beauty of digital technology: if you're just working from a pinned up character sheet next to your computer, you're essentially redrawing the character as you build it. With the sheet displayed in your viewport, you can place the geometry quickly and reduce the time spent tweaking your character model's position and scale. Mapping a scanned image to a set of polygons is much more effective than simply displaying the image as a viewport backdrop. If you apply an image to an object in the world, you can zoom in on details of the map and your geometry will scale up along with it. Finally, after you've used the sheet to position and build the skeletal structure for the character, it will be an aid in the construction of the final surfaces.

Figure 7 shows Anahani, a character designed by artist Heather Capelli for an Internet-based multiplayer game. We scanned the drawings into the computer and then cropped them in Photoshop. Setting the canvas size to a nice round number such as 300 pixels wide by 600 pixels high is a good idea, too; it makes it easy to match the aspect ratios of the source image and the polygons. The two polygons onto which you've mapped your character sheets should be positioned at right angles to one another, so that the rectangle containing the front-view lies on the z,x plane and the side-view image is mapped onto a rectangle in the z,y plane. You may have to slide the rectangles up and down until the head and feet of each image are aligned. Positioning the images so

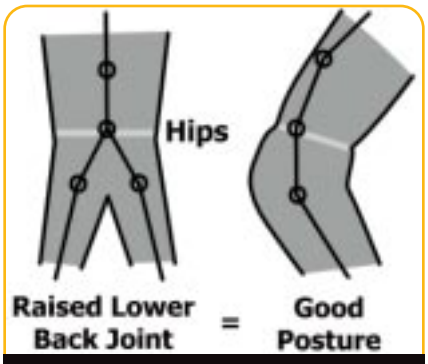


FIGURE 5. *Correct placement of the hip joint.*

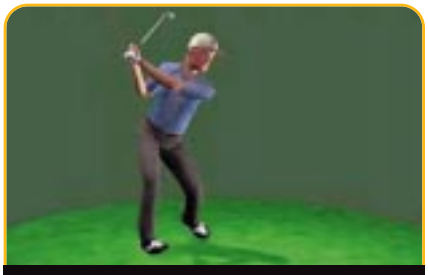


FIGURE 6. *This golfer is modeled correctly and moves much more naturally.*

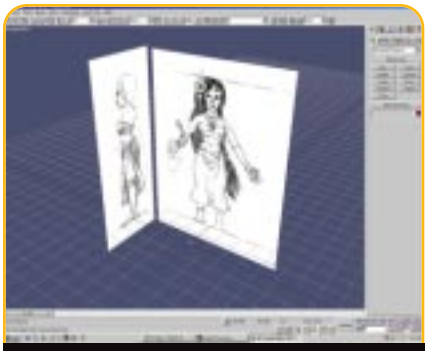


FIGURE 7. *The Anahani character sheet mapped to rectangular polygons in MAX.*

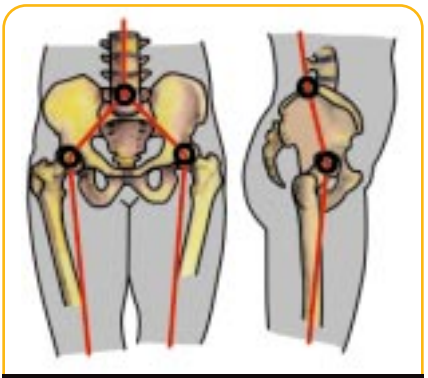


FIGURE 8. *The pelvis.*

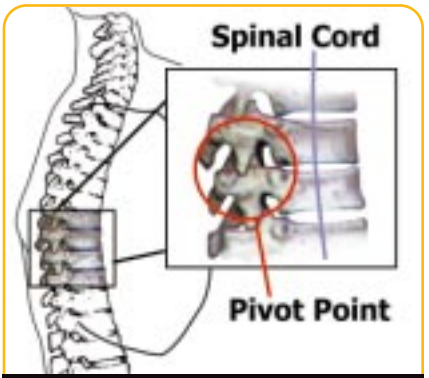


FIGURE 9. *The spine.*

that the planes don't intersect is important — that way, as you begin creating your model, your skeleton won't intersect the image-mapped polygons and confuse you.

Deconstructing the Body

Now let's look at where the bones should go. Move the bones to align them to the character sheet images using the front and side orthographic views. While most manuals teach you to place the bones in roughly the center of the geometry that you intend to attach, this is often not where bones are placed in the human body. As we saw in the opening examples, you can achieve some dramatic improvements in the realism of your model's animations with the application of some basic anatomical observations. I'll show some specific examples of how anatomical reference was used to guide the positioning of Anahani's bones in several areas.

The following tutorial is based on 3D Studio MAX and Character Studio, but many concepts can be applied to other modeling and animation tools. I use the basic Character Studio skeleton, so my skeleton is prebuilt and prelimited for me already. Another benefit of the Character Studio bones is that their axes are automatically aligned to the bone, which is not the case with standard 3D Studio MAX bones. Because you'll use the axis to build geometry onto the skeleton later, it's important, if you're using a different system, to build your skeleton with the axes aligned to the bones. Finally, note that our digital bones are really just straight lines between the axes, and are not the natural shape of human bones. Often, as we will see, the form of a human bone itself can be misleading, so it's important to closely examine the bone and accurately locate the rotational axis of a joint.

CREATING THE PELVIS. In the root, the pelvis, there are three points that form an upright triangle tilted slightly back at the top (Figure 8). The two hip joints are the bottom corners and the lower back joint is the top point. The hip joints are up about one-third of the distance between the bottom and the top of the pelvis. From the side, they're slightly forward of the skeleton's midpoint. When positioning the hip joint horizontally from the front, don't be fooled by the upside-down L shape of the top of the femur. The bones of the thigh naturally tilt quite a bit inward. But, with digital bones, your thigh bone will be much closer to vertical. If you get this wrong, you'll place the hip bones too far apart or the knees too close together. The lower back joint is centered horizontally within the body; it lies at the same height as the belly button, and as we'll see shortly, it's located to the rear of the body mass.

THE SPINE. The spine is a misleading bone structure. It actually rotates around a vertical axis that runs through the little wing-like structures at the back of each vertebrae (Figure 9). This rotational axis lies just behind the spinal cord, so it's farther back than might you think. An accurate representation of the spine is very important; the rib cage of your model will swing very differently if the figure's spinal bones are placed in the center of the torso mass rather than at the back as Figure 10 shows. In our model, the spine comprises three bones, which lends flexibility to the rib cage while keeping the attachment between the torso mesh and the skeleton simple.

THE SHOULDERS. The shoulder joint is very close to the top of the shoulder mass (Figure 11). Correct placement of this joint is crucial. Otherwise, the shoulder balloons when animated (as happened with the golfer in JACK NICHOLAS 5). From the front view, the shoulder bone appears roughly aligned with the sides of the rib cage. From the side, you can see that it's a little closer towards the figure's back than the front. The shoulder is probably the single hardest area from which to get a full range of motion without ugly skin crimping or surface distortion. Some animators build abnormally wide shoulders to compensate for such distortion.

THE ELBOWS. Note that the elbow sits to the rear of the arm mass (Figure 12B). As with the shoulder, this close proximity of the joint to the surface is essential to keeping the elbow from looking too soft or bulging unrealistically. The elbow is a little tricky, and its location in the arm is deceiving. I used to think it was located above the bulge of the forearm, like the relation of the knee to the calf. But in fact, the bulge of the forearm encompasses the joint. Note that the joint actually represents the

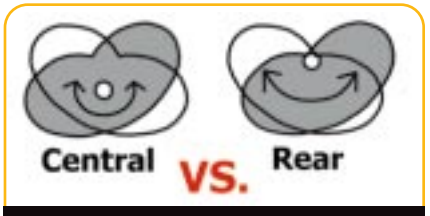


FIGURE 10. *Spinal position seen from above.*

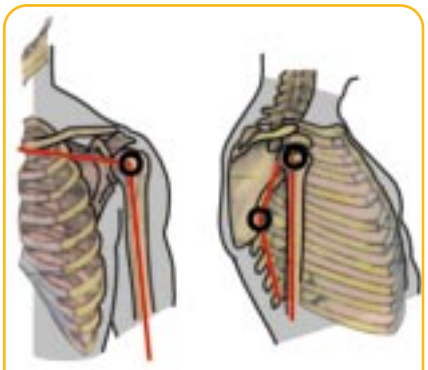


FIGURE 11. *The shoulders.*

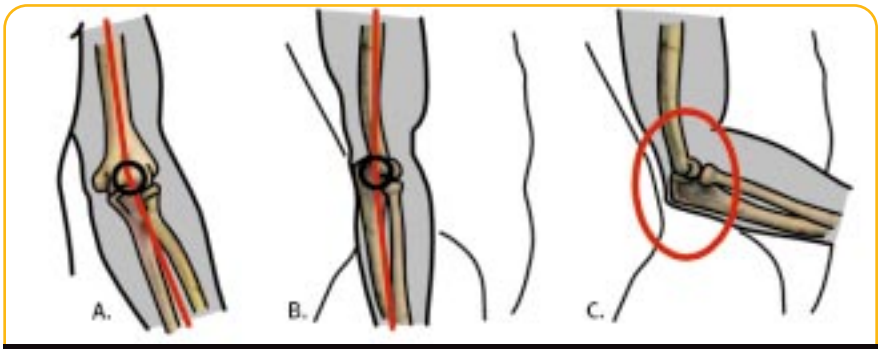


FIGURE 12. *The elbows.*

meeting of three bones, not two. At the elbow, the ulna is the primary bone of the forearm pair to which to pay attention. The characteristic point that you see at the outside of your elbow when it's bent, caused by the meeting of the humerus and ulna, is actually not the respective ends of those bones. In fact, the humerus is connected to the ulna just below the end of the latter bone, causing the ulna to cantilever outward when bent and create this point (Figure 12C). In your model, this is a matter of defining the attachments correctly.

THE KNEE. The knee took me some time to understand (Figure 13). The key to realistic movement from this joint lies in keeping the mass of the knee fairly constant as the leg bends. The bones of the lower leg, the fibula and tibia, actually slide around the lower end of the femur (Figure 13C). To achieve this movement, place the knee's axis of rotation a little above the vertical meeting point of the two bones (above the end of the femur), not where the two bone ends meet. As seen from the side and front, the axis should be centered horizontally within the knee's mass. By placing the knee joint away from the surface of the geometry, we get a bulging effect, which (unlike the shoulders) we want in this case.

The rest of the skeleton is positioned by aligning the remaining bones to the character sheet as well. Note that bone joints in one part of the body behave similarly in other locations. For instance, the joints of the finger and toe bones work just like the knee: lower bones orbit about a point partway up the higher bone. The meeting of the ankle and the foot is similar to the elbow: there is a protrusion by the heel that is akin to the point of the ulna. Finally, as noted before, the bones of the neck meet the head towards the rear of the skull, just as the rest of the spine relates to the rib cage.

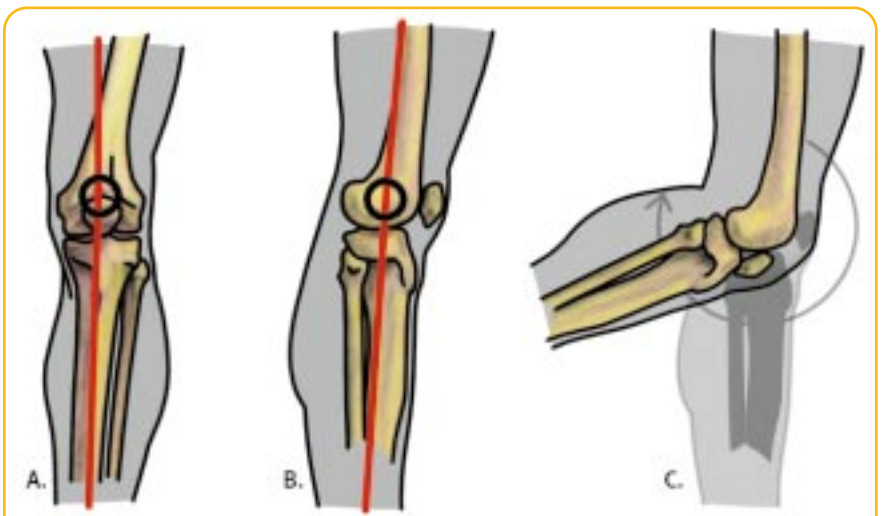


FIGURE 13. *The knees.*

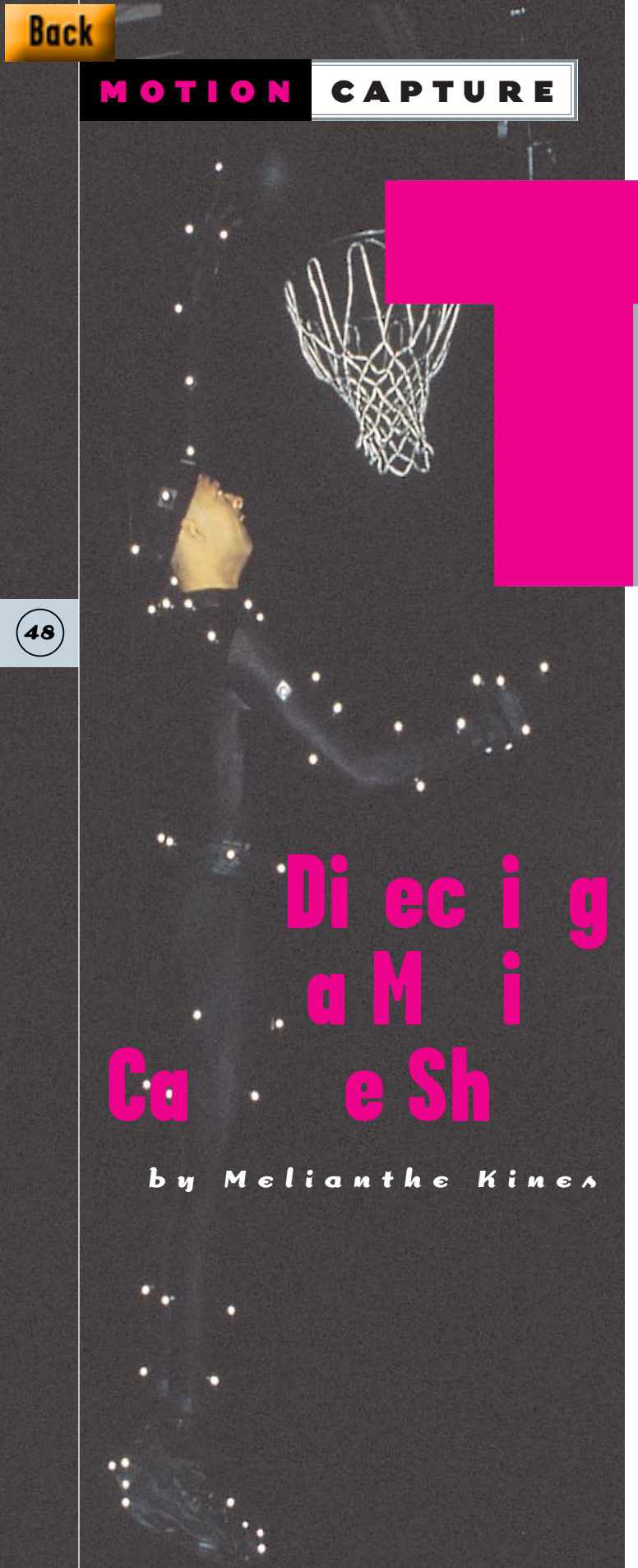
As Tools Evolves, Concepts Remain Valid

Modeling tools are constantly reinventing themselves, which makes our lives as game developers easier. For instance, the newest version of Physique, the skin attachment program in Kinetix's Character Studio, uses a new method of attaching vertices to bones. The tool now supports true weighted vertex assignments, using envelopes. This method has serious ramifications for how spline configurations at the joints will be built in the future, as it allows you more easily create a good looking vertex attachment.

However, no matter what features future software versions support, these core concepts that I've outlined will remain true:

- Character sheets are one of the most powerful tools you have for nailing the detail and proportion of your model, and using them within your modeling environment leverages their strengths even more.
- The quality of skeletal joint positions (particularly those at the base of the hierarchical tree) will continue dominate the geometry built upon it.
- Building the skeleton first and then properly posing it will help your modeling and save you significant time and effort.

By studying human anatomy, a modeler can learn where to put the bones of the skeleton so that it deforms properly, how the human body's joints behave, how much freedom of rotation joints have, and how muscles wrap around and connect to bones. By creating structures that closely mimic a real human body, your model will move and deform more naturally. ■



Directing a Motion Capture Shoot

by Melianthe Kines

The director of a motion capture project has two key responsibilities: giving direction to the talent and running the studio session. Directing the talent means clearly explaining the required movements to the performer, and then giving constructive and decisive feedback on each take. Running the session involves not only calling "action" and "cut," but also controlling the pace of the shoot — by deciding when to continue to the next move, allowing breaks, and ending the shoot day.

Thorough planning will make both of these jobs significantly easier. As I discussed in my previous article ("Planning a Motion Capture Shoot," *Game Developer*, September 1998), you should go into your motion capture session armed with a very specific shot list, a flowchart for each character, and a day-by-day shooting schedule. If you've gone over these documents in agonizing detail with the game's key team members and the motion capture studio personnel, you'll have a very clear idea of what has to be accomplished and how. The more control you have over the production process, the more you can focus on creating great motion-captured animation that will work perfectly in your game.

Procedures During the Shoot

KEEPING TRACK OF YOUR SESSION. You'll want to be able to review each take on video during the shoot and also afterwards to select which ones to use in the game. The easiest way to do this is to set up a regular video camera as a "slate camera" to tape the session. The slate videotapes you record should have time code that matches that of the motion data videotapes. Hold up a slate board for each take, noting the file name, move name, and take number.

Watching for continuity is critical. It's especially important to make sure your talent starts and ends in the correct rest frame position and hits floor and height marks. To doc-

Melianthe Kines is a freelance interactive director and producer. She has directed motion capture and Ultimatte shoots for Acclaim Entertainment and Electronic Arts. Her past motion capture projects include NBA JAM EXTREME, NBA JAM '99, THE CROW: CITY OF ANGELS, and FIFA: ROAD TO WORLD CUP '98. Her Ultimatte production credits include WWF: IN YOUR HOUSE and FRANK THOMAS BIG HURT BASEBALL. She can be contacted via e-mail at mkines@escape.com.

ument rest frame positions, take Polaroids at several angles and trace outlines of the performer on acetate taped to your slate camera video monitor. Put pieces of black tape on the studio floor to indicate where the performer's feet should go. Use tape or other marks on a nonreflective studio stand to show the height of other characters and objects in the game consistently. (Make sure you jot down the measurements that you use so the marks can be recreated in later sessions.)

Take extensive notes on your shot list, and highlight the takes that you like best during the shoot. Record the time code numbers of each take, if possible. Additional technical considerations to keep track of include timing, speed, distances, and direction. Some of these things you'll have to judge by eye, so watch closely and feel free to ask for video playback of current and earlier takes.

If you're lucky enough to have a good assistant, put this person in charge of continuity and keeping track of take numbers and notes. The assistant should slate each take and jot down your comments about each take. Having an assistant is invaluable because it allows you to focus on the performer and watch the action without distraction. A capable assistant can also take care of the performer's requests and will coordinate information between the studio and the team.

SET RULES FOR THE SHOOT. Have a meeting before the shoot to discuss how you'll communicate with other team members in the studio. Above all, establish how your team will make decisions. Ideally, a few key people (the lead programmer, animator, and/or producer) will be present during the shoot, schedule permitting. (Alternatively, you can schedule meetings with the team after the shoot each day to review the "dailies," the video you recorded on the slate camera.) You'll want input from them as well as from the motion capture studio personnel during the session. However, explain that the director will control the session's progress and will also be the only one to communicate with the talent on the set. Try to avoid arguments during the shoot about whether the move you just captured was good enough. This will waste time and make the talent confused and uncomfortable.

Suppose you feel that other team members understand the game better than you do (which would mean that you didn't spend enough time in preparation). You decide to let the animator and programmer critique the talent's performance and provide feedback. These team members may or may not be able to clearly express what they want; furthermore, they may contradict each other. All the performer will end up hearing is "Could we do just one more?" over and over again. Or suppose that you've handed control of the shoot over to the sports or martial arts experts whom you've hired for the shoot. If you don't watch out, your expert may confuse the talent by telling him or her that a move you've specifically asked for isn't "authentic." Sometimes, for the sake of good game play, you have to compromise between realistic sports moves and what will work in the game. Such distinctions should be the director's call, not the performer's or the expert's. If you're planning to have an authority on the set, try to involve them in the preproduction meetings so these issues are resolved before the shoot.

Having a few people in the studio is one thing, but don't allow a crowd. Don't permit unnecessary personnel to hang out and watch. Make it clear that you have a lot of work to accomplish and do not want distractions in the studio. Besides, the performer may feel self-conscious with a big audience.If the talent is a celebrity, it's especially inappropriate to allow a bunch of visitors to the studio.

The author directs Washington Wizards small forward Juwan Howard during a capture session for Acclaim's NBA JAM EXTREME at the company's Glen Cove, N.Y., motion capture studio.

Directing the Talent

Imagine that you've been hired as a motion capture performer for the first time by some game company. You walk out of the dressing room wearing a skintight motion capture suit with sensors attached everywhere. Even if you'd been warned about this, if you're human, you're going to feel a bit awkward. That's why the director's first task is to put the talent at ease. Joke about the wacky suit and make small talk to break the ice. Allow the performer some time to warm up and get used to the motion capture suit and any props that will be used. Play some music in the studio — in fact, ask the talent to bring some of his or her favorite CDs. Familiar music helps keep the performer's energy level up, and everyone else's too (especially if they don't despise the performer's taste in music).

If the performer is new to motion capture and to games, try not to condescend when explaining what you want done. Even if he or she has trouble grasping the more complex concepts, maintain a respectful tone with the performer. Start with the basics — explain how he or she will have to hold a scale position at the beginning and end of each move. Talk about the importance of hitting rest frames and marks. Discuss the game character and the shot list; you can even welcome and make use of the talent's suggestions to some extent, if they don't impact your motion capture and game requirements.

PROVIDE CUES. Help the performer hit his or her marks and maintain timing by providing cues. You can do this verbally, by calling out when the performer has reached a certain position. However, be careful not to yell out instructions at the performer like a football coach. If you're hollering, you'll probably make it hard for the performer to concentrate on the action at hand. The performer might also turn his or her

head to look at you when you say something; that would ruin the move.

You can also cue performers physically — you or or one of your team members can stand outside of the capture space and play an opposing character. For example, during the NBA JAM EXTREME shoot, we needed Juwan Howard to run at full speed and then come to a full stop right inside the capture space border. I didn't want him to have to look down at black tape on the floor while he was running, so I stood in his path, just outside the capture space line. He was forced to stop or knock me over. (Fortunately, he stopped.) This kind of trick is a good way to keep the motion realistic and dynamic — the performer should feel as though he or she is in a real situation, reacting to other people.

WORKING WITH TALENT. As I mentioned, you'll be watching for continuity and a whole slew of technical considerations. But try not to get bogged down to the point where you're treating the performer like a robot. Remember to keep an overall sense of the game's character and help the talent bring the role to life. Throw in some humorous idle moves or wild improvisations if you can spare the studio time. The performer will probably move quite naturally, being relieved of hitting his or her marks. Often, those last-minute ideas work well to add some spice to the game characters. Just play the talent's favorite music CD, roll the cameras, and joke around with the performer. The very worst thing you can do is put your performer on the spot by saying, "OK, ready? Now, act natural!"

You'll usually get the best performance when the talent is relaxed and you've got a good pace going. You want to build some momentum by moving from one shot to the next as quickly as you can. Also, make sure you compliment the talent frequently. If you're not happy with a take, never tell the performer it was "all wrong." Instead, try saying, "That was really good, but I want you to do it again a lot faster and with more power." Or you can even say, "Great! But you didn't hit your marks, so let's do it again." If you've got a capable performer, after a while you'll probably start getting what you want on the first or second take. Don't get too bold, though — always do at least one extra take of the move as a backup. It's okay to tell the talent it's a "safety take." You never know — the data on the other take could be corrupted, or the studio might have forgotten to hit record. Then again, you and the team might realize something after the shoot that makes the first "perfect" take useless. You may save the day by having a backup take that happens to solve the problem.

Let's face it, being a motion capture performer is not an easy thing to do. The sensors fall off constantly. You build up a sweat and you're wearing a skin-tight lycra suit. And you're being watched — no, studied very closely — by a bunch of strangers.

As a matter of fact, you and the studio crew will have to watch closely to make sure the sensors and the suit don't shift around too much during the shoot. If the sensors move on the performer's body, the studio crew will have to rescale the suit and adjust for any displacement. You'll have to rescale before and after lunch as well. Still, you can't let the performer take off the suit during lunch. Find out from the studio what can be safely done to make the talent comfortable during breaks — at least let him or her put on a robe or a loose sweatshirt over the suit.

DIRECTING CELEBRITIES. The main difference between directing celebrity and noncelebrity talent is that you usually meet celebrities for the first time at the shoot. You don't really know until then whether you'll be dealing with a prima donna or a complete professional. You'll never be able to force a celebrity to give you a good performance, so try to establish a good rapport and talk about the great video game you're creating. If the performer complains about the difficulty of wearing the suit and using sensored props, appeal to the ego. Say something like, "We knew this would be hard — that's why we needed you!" You can also call the performer's agent if you have to — you know, the person who agreed to the contract, gets a cut, and actually has the power to threaten the talent.

Many celebrities — Juwan Howard, for example — are extremely cooperative as well as talented. But I've also worked with performers who whined, made extraordinary demands, and seemed incredibly lazy. Usually, if I compromised on the requests — and kept my word — the complainer went on to give a great performance, just to get it over with. Professional athletes can really amaze you in the studio with what they can do, if they're motivated to get the shot right on the first take.

Getting the Moves You Want

Sure, it's nice to make the talent happy. But what about making the programmers and animators happy? What about making the person who buys the game happy?

Well, the only reason to please the talent is to establish an atmosphere in which you can get the moves you need. You have to know what you want in order to be able to explain it clearly. Trust your instincts — if you've done your planning properly, you'll know a good move when you see it.

What's a "good" move? For most videogames, the best kind of motion is fast and decisive. When a move is applied to a game character, it should be easily recognizable, look cool, and be quick. Sometimes, you come up with a really impressive move in the studio, but in the game it's confusing and silly. If you're going to use a move that takes even a few seconds in game time, it had better be worth watching. Don't forget that your motion data has to be converted to the target number of frames. If the move is too long, it's going to be choppy when it's cut down. And of course, if it's not cut down, it may exceed the amount of memory that's been allocated for the motion data.

Of course, there are plenty of things you can do to fix the moves "in post" — that is, with software. You can speed up moves, change the distance a character moves, or turn the character upside down if you want. Just remember that each time you alter the motion data, it will look less and less like realistic human motion. If the whole point of using motion capture is to get realistic animation, you should reshoot the moves until the performer can do them quickly, smoothly, and naturally. That's one reason that the choice of a rest frame is critical.

The rest frame. The rest frame is the single most important move you will capture, and it's the easiest to screw up. Why? There's a tendency to rush through the capture of this move. It probably seems as though all the performer has to do is stand there. Also, it's usually the first move of the shoot, and everyone's anxious to finish it and start capturing the cool stuff. Don't make this mistake — redoing the shot or compensating for an undesired rest frame is a major problem.

Often, it's impossible to redo the rest frame once you've captured the hundreds of moves that branch off of it (unless your programmers devise a fancy algorithm to modify every existing move in the game). Still, you may wonder, how could you possibly get a rest frame wrong?

First, the position of the feet and the angle of the body are very important. In most fighting games, for example, you're likely to have a "fighting stance," where the person stands at an angle with one foot in front of the other. If you plan ahead, your performer will be able to transition easily to a walk from this position. If you don't plan ahead, it will be impossible for your performer to go anywhere from this position. But you might want the character to have a more natural-looking stance — maybe facing straight forward, arms at the sides. In that case, how far apart should the performer's feet be placed? Will he or she be able to transition to a walk or a run easily? If the arms are at the sides, how long will it take to throw a punch or lunge for a ball, and then return to the stance?

Another problem is that the position might seem comfortable to the talent at the beginning of the shoot, but later on, he or she might find it hard to hold steady or return to precisely. By then, it's too late. This is especially true of those famous "crouching" positions that just about every fighting game includes. The only solution is to let your talent stretch out frequently, and watch closely to make sure the rest frame is hit precisely (unless that brilliant programming team of yours has blending tools to compensate for imperfect transitions).

Sometimes, you don't realize that something is wrong with the performer's stance until you see it in the game. It might look ridiculous when applied to the character model and used in the game environment. This is another good reason for a test shoot to try out the motion data in the game. You can experiment with several different rest frames and find one that works best for each character. Just be sure you can recreate the ones you like in the real shoot, possibly with a different performer.

Contact between characters. Whether you're creating a fighting game or a sports game, you have to figure out a way to match up any contact moves between game characters. First, you need to know the distance between one character and another when any interaction is likely to occur. For example, a fighting character might have a short-range and a long-range attack that can be used on opponents. You want to make sure that when your hero throws a punch, his hand appears

to knock the enemy's head back, not go through it. In a 3D environment, you've got to account for the direction and the angle of the attack as well; is it an upper cut from a shorter character on the defender's right side? You need to account for the way the characters stand when they are in close proximity to one another as well, unless you want them to be stepping on each other's feet. Your characters are probably of different heights, too — remember that when you're capturing a "knee to the stomach" move. And of course, get precise measurements of any objects as they will appear in the game. Suppose the "magic staff" prop that you use in the studio is two feet long, and the performer playing the wizard likes to gesture with it wildly. If the staff object in the game is relatively four feet long, whenever your all-powerful wizard character waves it, he'll poke his own eye out.

Studio Props and Sets

STUDIO PROPS. Your wizard would be safe from embarrassment if you had built the correct prop for your performer to use. When you plan your shot list, you should include sketches and exact dimensions of all objects with which game characters may interact. You'll need to give the studio some time to build props for your shoot. Motion capture props have very specific requirements. They must be completely nonreflective and are usually painted black. If the prop is going to have sensors on it, the studio and the team will need extensive preparation to be sure the prop will work. On each of the NBA Jam projects and on the FIFA soccer shoot, we successfully used a sensored ball to simulate its spinning motion. Bear in mind that the sensors on the ball make the performer's job much harder.

Also, with optical motion capture systems, the prop shouldn't block the cameras' view of the performer. If you obscure the sensors, the computer won't be able to track the performer's motion completely. For example, if your character if supposed to lift a large crate, you can't use a solid box four feet square. Instead, you could use a box frame of the same dimensions. In either case, the performer is going to have to do some acting to express the weight of the heavy box, thanks to the accuracy of motion capture.

You don't need a prop for every single object in the game, as long as you know the dimensions of each object and how it's supposed to be held by the character. Sometimes, it's easier for the performer to mimic picking something up because you don't have to worry about occluding the sensors on the performer's suit.

The motion capture suit should be designed to simulate the game character's costume. A football player can wear a suit with shoulder pads and a helmet. A femme fatale can wear high heels. On the FIFA shoot, the soccer players wore cleated soccer shoes. The motion will be very different depending on the kind of shoes the performer is wearing. It also depends on the kind of surface on which the performer is walking or running.

STUDIO SETS. As you can imagine, you can't have performers wearing shoes with cleats in the studio unless you have a grass floor. For the FIFA shoot, the studio constructed a mini-soccer field made of boxes several feet high, filled with dirt, and covered with grass. The soccer players were able to run, kick, and dive realistically, and they were able to control the soccer ball in an authentic way, too. To allow the performers to accelerate into and decelerate from a fast run, the studio built grass-covered ramps in and out of the capture space.

Someone running at top speed will usually make it through the average capture space in one stride, left foot and right foot. For any game where a character is supposed to run, you need room in the studio for the talent to run into and out of the capture space. Of course, it's easier if you're working with a normal floor. More and more these days, though, motion capture shoots are being done on special surfaces that simulate the game's terrain. Hockey and figure skating games have been captured at ice rinks, for example. The more complicated your shoot, the better your advance planning has to be.

Special Set-Ups and Stunts

Of course, you can create some pretty complicated set-ups in a studio with a plain floor. The studio can build you some steps, ladders, or a steeply angled floor to correspond with your game's environment. If you were to use the basic walk cycle for going up and down stairs or traversing hilly terrain, it wouldn't look right and it probably wouldn't fit together properly.

Most games wouldn't be complete without some spectacular jumps and falls. You could simply let your performer crumple gently onto a mat, but what fun would that be? Bring in a stunt coordinator if you're going to put your performer at any risk of injury. If the shoot requires forceful contact between the performer and an object, another performer, or the floor, it's best to have a safety expert on the set. Your legal department will be much happier.

Stunt coordinators usually work on film or theatre projects and can help make "injuries" look more dramatic. With motion capture and good software, you can exaggerate these moves ten times over. For example, the performer can stand on a medium-sized box and fall off onto a big soft mat next to it. Tweak this motion and put it into the game, and your game character can appear to fall off of a twenty-foot ladder or a basketball rim. Use the box and no mat if you want the character to jump off instead of fall. Make

sure that the performer really exaggerates the bending of the knees and the swinging of the arms, on both the takeoff and landing of the jump.

If the game calls for the character to jump off of a tall building instead, consider using a flying rig. Not everyone can use this easily — it's particularly awkward in a motion capture suit. It's hard to control the swinging of the cables while trying to perform a move smoothly, all while starting and ending in a rest frame. Still, if you simply require the character to flail his or her arms in the air while hurtling towards the ground, ask your stunt coordinator to set up a flying rig. If you need a performer who can control his or her movements enough to look like a flying superhero, hire a gymnast or stunt person with experience using this contraption.

A ratchet is another helpful stunt setup that propels the performer violently backwards in a harness to simulate getting blown away by powerful weapons or explosions. You must have an experienced stunt coordinator to use it, of course, and even then it's a little nerve-wracking. Plan on capturing only one take for each ratchet move and schedule it for the end of the day, because it really knocks the wind out of the performer. The talent should be paid a little bit extra as hazard pay if you're going to put him or her in a ratchet or in any potentially dangerous situation.

You can also use rigs to simulate nonhuman motion. Using a flying harness is obviously one way to make your characters look supernatural, but there are plenty of other ways to capture strange-looking motion. Put the performer on stilts or outfit the performer with extra motion capture arms and legs. Test an actor's ability by making him or her crawl across the studio on hands and knees; ask what animal he or she had to play in acting class. For that matter, you can look into bringing a real animal into the motion capture studio. I know that some people have experimented with this, but I can't really give you any advice on how to direct Fido. I guess you should apply the same rules you would with humans — yell, plead, and bring lots of treats.

Wrapping the Shoot

As the director, you get to say, "That's a wrap!" (Others may plead with you to let them utter those words, but don't give in.) Collect your notes and slate videotapes after the shoot, and settle into a comfortable chair to review what you've captured. After you've selected your picks for each move, provide the take number and time code in and out points to the motion capture processing team. When the animators receive the data, discuss it with them to make sure they understand and like what's been provided. Review the moves as they're put into the game, and ask the team for feedback while time is still available for reshoots.

Celebrate the end of your last reshoot with a wrap party for your cast and crew. Thank everyone for their hard work, and see if you can get them copies of the finished game. After all, they'll want to see the spectacular character animation made possible by your successful motion capture sessions. Good shooting! ■

POSTMORTEM

Monolith SHOGO: MOBILE ARMORED DIVISION

by John Jack

Monolith was founded by some software industry veterans in late 1994. Then, in early 1996, this core group of developers pitched an original idea to Microsoft. When Microsoft agreed, work began on SHOGO: MOBILE ARMOR DIVISION and its underlying engine, LithTech. According to the terms of our agreement with Microsoft,

LithTech would be a new stand-alone 3D game engine that could be leveraged across multiple products, as well as licensed to third-party developers. SHOGO, an anime-inspired first-person shooter starring 40-foot tall transforming

As the first project manager hired by Monolith, John has been in the middle of it from the beginning, witnessing the successes and failures generated by an incredibly talented and energetic company. When he's not riding his sport bike, John can be found in his office answering e-mail, yelling on the phone, waving his hands, and ordering pizza.

robots, would be the showcase product for LithTech. LithTech and SHOGO were to be developed simultaneously, with each project pushing the other in terms of feature requirements, capabilities, and design.

When I joined the company as a Project Manager in April of 1996, we had just finished negotiations on the LithTech/SHOGO prototype, and we had begun to ramp up for production of both the engine prototype and the game. I was given responsibility for project managing both the game and the engine, as well as acting as the liaison between Monolith and Microsoft.

While the entire project involved a number of lofty design goals, at the core we had a simple, straightforward premise — create a cutting-edge 3D engine that would allow us to make SHOGO a great 3D action game. We were gunning to produce a product that would rival, if not stomp, any other 3D action game currently in development. Multiplayer capabilities via a client/server architecture, complete support for DirectX, high-speed action, and full Direct3D support were all basic requirements for the engine. In fact, we would focus primarily on 3D hardware acceleration first, and worry about software rendering later, which was considered highly risky in early 1996. The LithTech/SHOGO project was by far the biggest project Monolith had ever undertaken, and it would shape, for better or worse, the future viability of the company and our technology.

A Look at the Good...

Even though we hit more than our fair share of bumps in the road during the development process, there were a few things that went right (or nearly right) from the very beginning. These successful endeavors made dealing with the constant challenges of developing both a game and an engine seem possible.

1. **DIRECT3D.** The decision to go with Direct3D as our only supported 3D API turned out to be one of the best decisions we made during the development of LithTech. When we began pitching the idea of LithTech and SHOGO, one of the selling points for Microsoft was that both the engine and the game would focus on Direct3D. Monolith had quite a bit of experience working with early versions of Direct3D (we wrote a few of the Direct3D test applications, including the ultimate space-bar test, ROCKEM), and it was this experience, combined with our knowledge of DirectX in general (we also did the first two DirectX game sampler CDs for Microsoft) that helped us land the deal.

Although the idea to go with Direct3D was great on paper, it didn't work out so well in practice. When we began working on LithTech and SHOGO, Direct3D was in its earliest stages, which proved frustrating. Many features that were available under Glide weren't available under Direct3D, and because the only 3D hardware worth anything at that time was 3Dfx-based, we questioned the logic of supporting both Direct3D and Glide. So we decided to pursue separate Direct3D and Glide versions simultaneously. Pursuing an independent Direct3D version let us support any new cards that were in development from other manufacturers, and supporting Glide meant that we could get the most out of the 3Dfx card.

Mike Dussault (LithTech), Scott Schlegel (LithTech), Bill Brooks (SHOGO), Wes Saulsberry (SHOGO), Scott Pultz (LithTech, in back), Brad Pendleton (LithTech, in front), Kevin Stephens (SHOGO), Matt Allen (SHOGO), Nathan Hendrickson (SHOGO), and John Jack (SHOGO/LithTech).

To support both APIs, LithTech originally had an abstraction layer of rendering code contained in the engine itself. This abstraction layer, which sat beneath any API-specific rendering code, covered most of the basic 3D setup issues. We then had separate, external .DLLs (D3D.DLL and GLIDE.DLL) to render for each API we supported. In theory, we could plug in any new rendering .DLL if and when we decided to support another 3D API. Each of the rendering .DLLs contained all of the API-specific instructions that couldn't be included in the abstraction layer. Sharing some of the same basic rendering code made maintaining each .DLL much easier during development, and worked well for a while.

However, as game scenes and models became more complicated, rendering performance became more of an issue. The abstraction layer approach was a likely culprit — it was too slow to get the best performance out of either API. But removing the abstraction layer in the engine would mean duplicating a ton of code, causing thorny code maintenance issues. Every time a feature went into the engine, it would have to be implemented in both Direct3D and Glide. At about this time, we got an early version of DirectX 5, and it forced us to make a decision.

DirectX 5 looked pretty solid, and had most of the fea-

SHOGO: MOBILE ARMOR DIVISION and LithTech

Monolith Studios
Kirkland, Wash.
(425) 739-1500
http://www.lith.com/
Team Size: 15
Release date: October/November 1998
Time in development: 26 months
Critical hardware: Intergraph NT workstations, Motion Analysis motion capture system
Critical Software: Microsoft Developers' Studio, Softimage, Dedit (internal), LithTech (internal)
Target Platforms: Windows 95/98 P200 w/4MB 3D Card

LithTech supports several different types of environments, including outdoor terrain areas. Check out the ships in the cloud layer, as well as the drop ship off in the distance.

A city level while piloting a mecha. D3D gives us all the features we need for special effects. Note the halos around the missiles.

Special effects abound. Note the full-brights in the upper right corner.

tures that we needed for SHOGO. So, Mike Dussault, the lead engineer on the LithTech project, decided to move all of the rendering code into the D3D.DLL to determine what speed benefits we'd get. This was a pretty big undertaking — we estimated that it would take a minimum of two weeks of work — so it was risky. Fortunately, the results were much better than expected, and as a result, we decided to focus solely on Direct3D. Mike was able to get all of the features that we needed out of that version of Direct3D, such as colored dynamic lighting, lightmapping, and fullbrights. As an added benefit, by only supporting one API, Mike could focus his work on optimizing the engine's performance under Direct3D as much as possible. The decision to support only Direct3D proved to be right on. With the upcoming release of DirectX 6, LithTech is now in an excellent position to support just about any card out there without resorting to proprietary APIs.

2. THE PHYSICS OF BOXES. A big issue that the engine team had to tackle was how to define an object's bounding box, and how to use this box to calculate hit and collision detection. The engine team realized early on that the fastest way to do our physics calculations in LithTech would be to have axis-aligned bounding boxes. This meant that an object's dimensions would remain aligned with the world axes (x, y, z). So the height (y) of the bounding box could vary, but the width (x) and depth (z) needed to remain equal. Unfortunately, this system became problematic with some of our more complex character animations, such as death animations, which moved the character away from the center of its bounding box, or ducking animations where the game needed the character's bounding box to shrink in the y dimension. Model geometry that moved outside the bounding box (such as arms or legs) would clip through walls or other objects, which was totally unacceptable.

The solution that the engine and game teams came up with was to allow the bounding box, and the model's translation within that bounding box, to change per animation. This solution allowed us to set the bounding box and translation in the actual model file, which meant that we could see, in each looping animation, whether or not any part of the animation clipped outside the object's bounding box, and alter the bounding box size and model translation if necessary. Although the final solution involved rectangular, axis-aligned bounding boxes, the additional flexibility of changing the bounding box sizes and model translations provided realistic, usable results. This solution was a great example of compromise between speed for the engine and flexibility for the SHOGO team.

3. ANIMATION AND TEXTURING. When we began work on LithTech and SHOGO, we were an entirely Softimage-based development studio. Back in those "early days" (circa 1996), we really didn't have a clear idea of what we wanted from the LithTech animation and modeling toolset, other than the lofty goal, "Let's make a great 3D action game." We did know that all of our models, characters, and objects would be created in Softimage 3D, so at minimum we needed to develop a good method for exporting or converting Softimage models to our engine's model format.

A big design goal for SHOGO was to allow enemies to use (and more importantly, to display on screen) the same weapons that the main characters used. So we needed a system that allowed us to place separate weapon models in the hands of enemies and characters. The LithTech engine's model format was in part influenced by this game requirement, so the team pursued what is commonly referred to as a bone-based or skeletal-based model format. In a skeletal-based system, model geometry is attached to and parented by a hierarchical node system, rather than represented by a continuous mesh of geometry. The hierarchical system would ultimately give our game development teams access to individual nodes on a model, which would (hopefully) add additional flexibility to the animation and modeling system.

As the hierarchical system in LithTech grew, we gained the ability to attach additional models, sprites, and lights to separate nodes on models. Because the system also retained rotation and translation information for any parented nodes, it allowed us to pull off some great special effects, which was extremely important to SHOGO. For example, under this system, we were able to attach headlight sprites and lights to headlight nodes on a truck, as well as attach muzzle flash sprites and particles to the end of an enemy's rifle. An unexpected benefit of the system was that we could also have characters or enemies that were made up of separate models and attached together. The tanks in SHOGO, for example, are actually made up of two

separate models: a base and a turret. The animation system allows the tank turret to turn independently of the base, even though the turret remains attached to the base as the tank moves through the world. Because the tank was made up of separate models, the tank turret could be blown off the base of the tank when destroyed, which, when combined with an explosion, made for a much better special effect.

In retrospect, using Softimage was probably overkill for our modeling needs, considering that most of our animation data came from motion-captured sources. However, using Softimage did allow us to use UV texture mapping for our models, giving us more realistic texture mapping, as well as more flexibility in terms of texture layout and size. UV mapping allowed us to map the same texture onto different models in different ways, which saved texture memory and reduced the necessary size of most model texture maps. The down side to UV mapping is that at first glance, our texture maps don't really have clearly defined regions (head, shoulders, legs, and so on), which might make end-user texture map modifications a little more difficult.

4. Motion capture. In the early stages of development on LithTech and SHOGO, we did some initial experimentation with motion capture to find out whether or not it would work for our needs. We got a few test animations from several external studios and had mixed but mostly positive results. The real problem was access to the equipment and lead time for that access.

Brian Waite, who evaluates all of our new 3D hardware and technology, and I (being the hand-waving producer-type whose project would actually use the equipment) ventured to Northern California to meet with the folks from Motion Analysis, which at that time (again, 1996) offered a highly accurate optical motion capture system. An optical (vs. magnetic) system had the advantage of no wires and no mess, which we felt would be better for our needs, because many of the animations we'd need would involve relatively complex combat. After evaluating the Motion Analysis system and getting some hands-on experience and training, we decided to go ahead and purchase the system.

We believed that having our own Motion Analysis system in-house could really shave time off the animating process for our 3D artists, as well as give us more human-like motion and movement for our in-game characters. Softimage worked well with the Motion Analysis data, so everything seemed to be a good match. However, purchasing a large piece of expensive equipment proved to be a major factor in the future of LithTech tools development and the model creation processes in general.

After getting the system back to Monolith, we soon came to realize that motion capture is a double-edged sword. On the plus side, motion capture let us get multiple animations into the game quickly, with very realistic results (if you have good motion capture actors) and data that we could use across multiple characters, all of which saved us time. On the negative side, the system itself takes up a lot of physical space, which meant that we needed a big room with tall ceilings for best results. Also, and this is the big one, we ended up with a huge amount of data after a motion capture session, which meant that our 3D artists and motion capture technicians spent lots of time working with data that turned out to be extraneous or unnecessary.

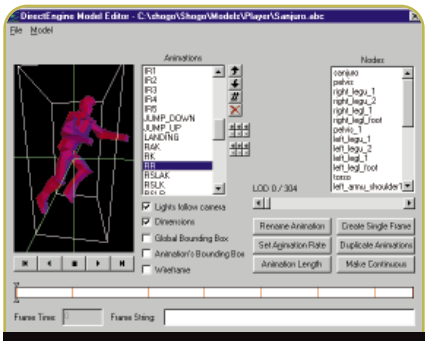
All in all, however, the results we got from the system were great. We were able to many animations in a short period of time, we could get realistic movement and animation into the game in a hurry, and because the system was in a building next to the development teams, we could schedule motion capture sessions pretty much any time we needed a new animation. The early motion capture sessions were mostly experimentation, but after a few times, we had nailed down the process. Along the way, we were fortunate enough to hire Simon Wong, who had worked extensively with Motion Analysis systems in the past, to head up our motion capture studio. Without an expert in this position, our motion capture experiences might not have turned out so well.

Enemies firing the same weapon you're currently holding. Note the translucent water off to the right. If this were a movie instead of a screenshot, you could see that the water surface is actually moving.

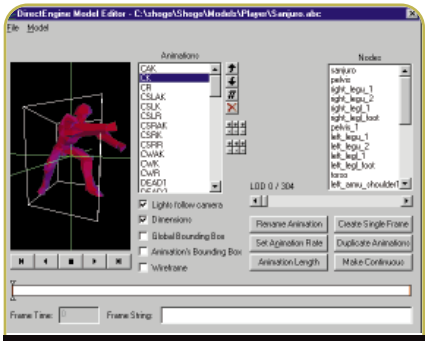
5. The development tool that almost wasn't. Our motion capture system can capture up to 120 frames per second, although we get better accuracy if we take that down to 30 frames per second (which is still more than enough for in-game animations). Once we'd get that data from our motion capture guys (the cleaning/exporting process takes about 20 minutes per animation), our 3D artists brought it into Softimage and applied the motion capture to our in-game characters. Each animation

An on-foot level from SHOGO. Note the soft lighting on the left, compliments of single-pass multitexture lightmapping available under DirectX 6.

Another on-foot level from SHOGO. Check out the blood particle effects, used extensively throughout SHOGO.



A character model shown in ModelEdit — note the size of the bounding box.



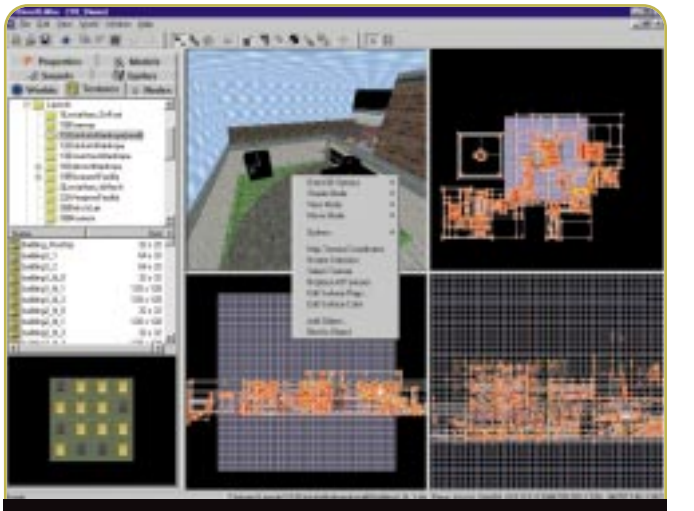
The same character model in a duck animation — LithTech allows character bounding boxes to change size per animation.

that the artists applied took some time, but in the end we'd get a character that had 20-60 different realistic and useful animations in about half the time it would take a 3D artist to animate them by hand. By shaving time off the animation process, our 3D artists could focus on creating models and textures, rather than toiling over creating realistic, hand-plotted movement. Once the animation data was applied, the model was exported to the LithTech format (.ABC file) using a custom plug-in that the engine team wrote for Softimage. The game team could then use this .ABC file and call and play back specific animations through code or game events.

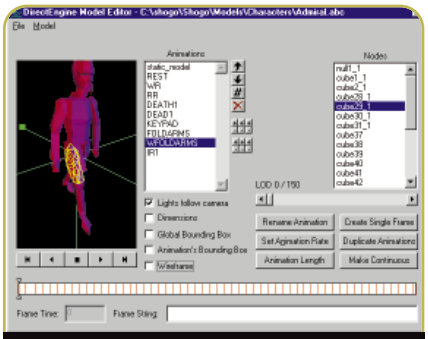
Our motion capture/animation process was all fine and dandy until we realized that many of the models were memory pigs. The size of our models were dictated by (geometry data) + (animation data). The geometry data was relatively small (approximately 30-50K depending on the number of polygons and vertices in the model), but the animation data (which is actually the number of nodes * number of keyframes) was huge. In some cases, our animation data was 6-8MB per model! This realization nearly gave the SHOGO team a collective heart attack, but the engine team quickly came up with a solution. The huge size of the models was mostly due to our 30 keyframes per second data rate for animation, which turned out to be tremendous overkill. Because LithTech had supported interpolation between keyframes from almost the very beginning of the project, there was no need for the in-game animation to contain 30 keyframes per second. In fact, we found that 7-10 key frames per second was usually more than enough when coupled with the engine's ability to interpolate between keyframes. The problem that we faced was in removing all the extra keyframes. The data capture rate of our motion capture system couldn't go below 30 FPS and still be accurate, so reducing the capture rate wasn't an option. We could cut down the frames in Softimage, but this process was going to be difficult due to the way that Softimage deals with animation data from the motion capture system. Even if we did pursue cutting down the keyframes in Softimage, we couldn't actually see how the engine's interpolation system would deal with the data until the models were exported and put into the game. So, we came up with a different solution, and ModelEdit was born.

ModelEdit started simply enough — the tool let our artists view animations for a particular .ABC file and trim down the number of keyframes in each animation based on what looked good. ModelEdit supported the same interpolation that the engine used, so what you saw in ModelEdit was what you'd get in the game. However, once we had this "post-process" tool, a flurry of features were quickly implemented when we suddenly realized, "Wow, how did we ever get along without these features?" Next came the ability to tag keyframes to trigger game events, such as character footstep sounds or firing information for weapon models. Then came the ability to copy, paste, and import new animations into a model from another model, which made the process of exporting new animations or models much easier for our 3D artists. Mike Dussault and Brad Pendleton, the two engineers heading up LithTech development, spent some serious time working on ModelEdit, but every hour they spent saved countless hours for our artists, designers, and game engineers.

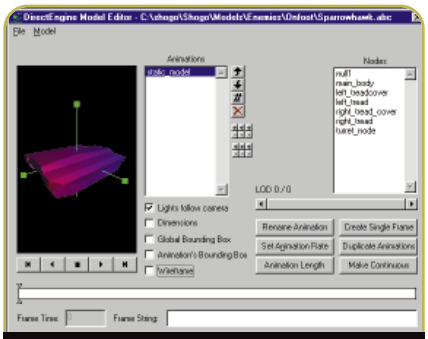
In fact, very recently a feature was added to ModelEdit that saved our



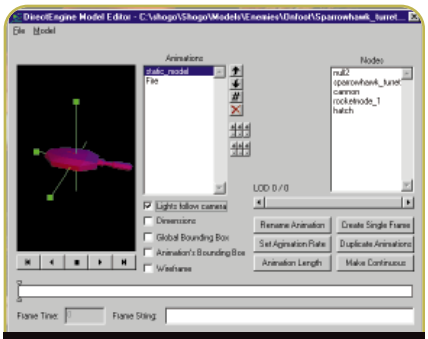
Present day Dedit supports multiple shading modes. Early versions weren't capable of creating complex worlds such as the world file seen here.



A character model shown with a leg node highlighted. Being able to identify and rename individual nodes saved our asses late in the game.



The base of a tank model in SHOGO.



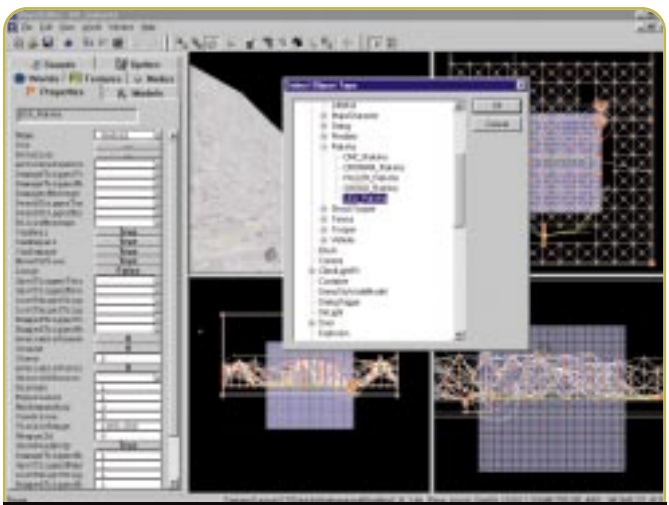
The tank turret model, which gets attached to the base via game code.

project. About three months ago, our main Softimage network drive crashed, and one of our artists lost about two weeks of work. We had the .ABC files, but not the original Softimage databases for those models, so we couldn't go back and change animations or geometry or rename nodes. The engine team added a feature into ModelEdit that allowed us to identify and rename nodes in any .ABC file, which was necessary for our hit detection system to work correctly. Without this feature in ModelEdit, the artist would have had to recreate the models from scratch — a time-consuming process that we couldn't afford so late in development. This recent feature is a perfect example of how two hours of an engineer's time wound up saving more than 80 hours of work for an artist.

...And the Bad and the Ugly

Even though a lot of things went right, we faced more than our fair share of stumbling blocks during the development of SHOGO and LithTech. In retrospect, most of our problems could have been avoided if we'd taken the time to think through the project clearly, but we certainly learned from every mistake we made along the way. On the bright side, both development teams are much wiser, stronger, and more seasoned having gone through the entire process.

1. LEVEL EDITOR DIFFICULTIES. Our early neglect of developing a strong, feature-rich, and stable level editor was one of the biggest mistakes we made during development of LithTech and SHOGO. Considering that Monolith had a very good understanding of the importance of solid level design tools — we had previously developed our own level editor for BLOOD, as well as an entire toolset for WAP (the engine used to make CLAW, GET MEDIEVAL, and GRUNTZ) — the lack of engineering time spent on our editor, Dedit, is an amazing oversight in retrospect.

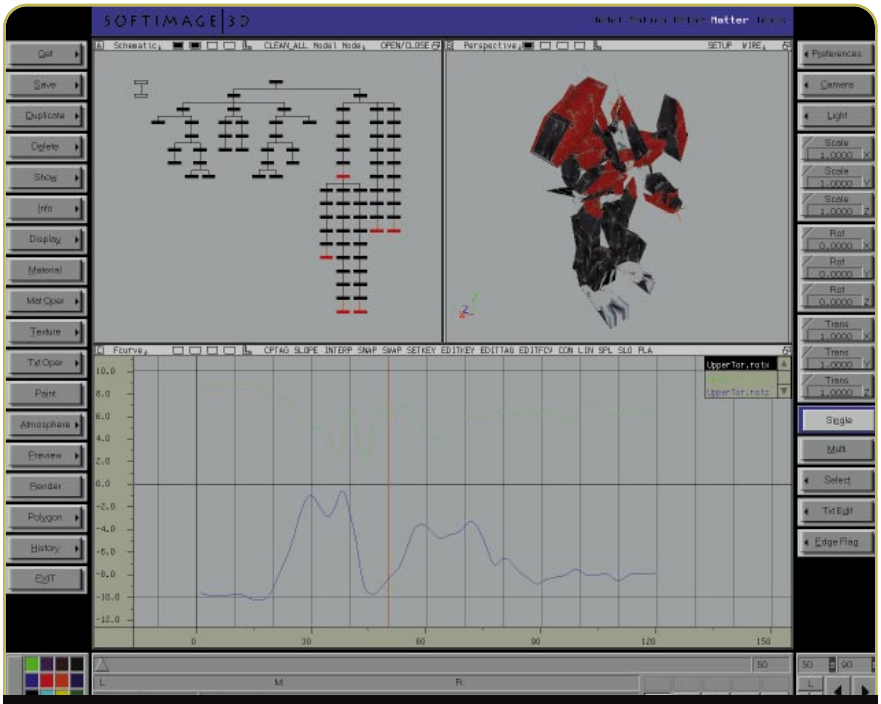


Present day Dedit makes adding objects easy — just right click and select the object you want to add to the world. Early versions required multiple, painstaking steps to add objects.

There were a few reasons why Dedit wasn't a big focus for the engine team from the beginning. Mike Dussault and Brad Pendleton were hard at work on the core design and architecture of the engine, and many of their decisions would affect a level's overall file structure. In addition, because the engine team didn't have access to seasoned, full-time level designers when the project began, some incorrect assumptions were made about how the level design tools would be used. As a result, early versions of Dedit were more than capable of creating test "box" levels, but lacked some of the more advanced features for creating complex geometry and placing objects and models. As we reached the point in the development cycle where we needed to start showing a progression of game features, the lack of advanced features became painfully evident.

Enter Craig Hubbard, the lead game designer and level designer on SHOGO (who had recently finished working on a level pack for BLOOD), and two additional full-time level designers, Nathan Hendrickson and Todd Clineschmidt. Before Craig began working on SHOGO, the game was essentially designed by committee, which caused (among other things) a severe lack of focus on tools development. Dedit contained the necessary core features at the point Craig joined our team, but certain crucial features weren't very intuitive, and many were far from usable. The addition of these designers was really the turning point for Dedit development and usability. Their combined input, suggestions, and demands made Dedit a much better tool.

So what lessons did we learn? First, if we had put more emphasis on level design tools, and if the engine team had had access to experienced level designers on the project from the beginning, Dedit would have been in much better



Using Softimage for modeling, texturing, and applying motion capture data made our lives much easier. You can see the fcurves at the bottom from the motion capture data.

shape from its inception. Early attention to Dedit's usability could have literally shaved months off the level design schedule. That said, Dedit is now a very flexible and usable level design tool. Who knows? If it hadn't hit rock bottom, Dedit may never have become the excellent level design tool it is today.

2. DSCRIPT. The biggest mistake we nearly made was sticking with Dscript, our custom programming language, until the bitter end. The idea behind Dscript wasn't a bad one. Written by the engine team, Dscript was to handle specific game events and object handling. In mid-1996, custom scripting languages seemed to be the future of 3D engine development — QUAKE had QuakeC, UNREAL had UnrealScript, and LithTech had Dscript. The Dscript model sounded good on paper — specific functions to deal with game events, garbage collection, and quick compile times.

But it posed a number of problems as well. As an engineer, you had to learn the language. It was similar to Java or C++, but it wasn't either, which meant a learning curve. Then there was the speed issue. An interpreted language would never be as fast as executable code, which didn't seem to be a big deal at first, but as we later found out, the speed difference became very apparent once many scripts were running. Also, with a client-server architecture, every script had to be duplicated on the server and the client, which if nothing else complicated file structures. The big whammy turned out to be debugging tools. In June of 1997, Kevin Stephens (SHOGO's lead engineer) and a few other game engineers at Monolith began to have serious doubts about whether or not we could make the games we wanted to if we continued to use Dscript. With an unfinished engine running Dscript, it was very difficult to debug game-related errors. Was it a core engine function that was crashing, or was it the game scripts? More than a few game engineers feared that because of Dscript, they would be tempted to take the easy and safe route when it came to feature implementation — being too ambitious could lead to hours of debugging time, which engineers simply couldn't afford.

Fortunately, the engine team agreed with the game engineers, and although we had spent the better part of four months writing Dscript and setting up the engine to deal with it, we decided to scrap it and go back to using Win32 .DLLs. So not only did we waste time writing and implementing Dscript, we also spent more than six weeks porting everything over to executable code, which added even more time to the development schedule. This additional delay was especially painful for our design team and our publisher, because nearly two months went by with few (if any) new engineering-related features implemented.

Scrapping Dscript, regardless of the effect it had on the schedule, was an essential ingredient to the success and usability of LithTech. If we had stuck with Dscript, our development timeline would probably have increased anyway, and the quality of the engine and the game would have suffered dramatically. Unfortunately, we could have avoided the entire exercise if we had closely evaluated the alternatives, although this may not have been possible without first trying it out for ourselves. The irony here is that when we began working on Dscript, many of the third-party developers who were interested in LithTech seemed to be very excited about the prospect of using a custom scripting language. Then, just as we began to make the switch from Dscript to Win32 .DLLs, the tides turned, and very soon we started to receive negative feedback from developers who were already working with other engines that relied on a scripting language. As it turns out though, the switch proved to be a good move in terms of internal development as well as external licensing.

3. NOT ENOUGH TIME. During the LithTech/SHOGO development cycle, we constantly struggled to develop an accurate schedule for both the game and the engine. When we originally signed the deal, we signed up to create a game engine and game in 18 months — an aggressive schedule to say the least. Unfortunately, we grossly underestimated just how long it would take to produce both entities. The really frustrating thing was that even with our extensive background in creating games and engines, we were still way off on our estimates.

4. MONOLITH GROWTH. One of the biggest challenges during the development of LithTech was dealing with the enormous growth of our company. When we finished work on a prototype for Microsoft in June of

1996, Monolith employed about 20 people. By the end of 1996, we were up to nearly 80 people, with several different projects under development. The original team that began working on the prototype moved on or up to other projects, and the LithTech team was essentially replaced with new faces in short order. Huge growth made designing the game and the engine very difficult, as plans for the engine and the game changed as different people moved into or out of the project. Fortunately, the engine and game teams stabilized by the beginning of 1997, and we ended up with a very talented group of developers.

5. A GAME *and* AN ENGINE? One of the worst (and I mean worst) ideas that Monolith ever had was to think that we could develop both a game (SHOGO) and a game engine (LithTech) at the same time. All I can say is that it will never happen again. When we began work on LithTech and SHOGO in 1996, we didn't quite realize what we'd bitten off. We ramped up our game production staff too quickly and expected the game team to keep pace with constantly changing tools and technology.

While any engine team needs access to experienced level designers, game engineers, and artists when designing the architecture for a new engine, expecting these game designers, engineers, and artists to be productive with anything less than finished tools was a mistake. All in all, I don't think that it affected the overall development schedule of the project (we still came in at around two years for the whole ball of wax), but we certainly spent a lot of money paying frustrated people who would have been better off working on other projects until the technology and the tools matured. We were very fortunate to have outstanding engine and game teams, without which the SHOGO/LithTech process would not have been possible.

What we did learn was that future versions of LithTech will be developed by the engine team with frequent input from game teams, but without the pressure of having to implement game-related features during the early engine R&D phases. This new process should remove the pressure from both the engine team and the game teams, and make everyone's lives easier.

Closing

In May of this year, we purchased back all rights to LithTech and SHOGO from Microsoft — a major decision that solidified our future for internal game and technology development. Over two years and a bunch of money later, LithTech is without a doubt one of the most flexible 3D gaming engines currently available. Through the good, the bad, and the ugly, the SHOGO/LithTech project was a learning experience for everyone involved, but the end result was an engine that's capable of producing several different kinds of games — a testament to the engine's elegant design and flexible architecture. LithTech is now being used by three internal game development teams at Monolith — Shogo, BLOOD2: THE CHOSEN, and MINDBENDER — and we expect to have licensed LithTech to several external developers by the end of 1998. ■

The Intel Object Architecture Profile

by Ron Fosner

I was very happy when Intel shipped the Intel740 graphics accelerator chip. Having Intel focus (even a little) on 3D means that it is interested in bringing 3D into the mainstream. It's fairly apparent why Intel dove into this market. In order to sell faster computers, there has to be a compelling reason for consumers to buy them. And while you and I wouldn't hesitate to buy the latest 666MHz screamer, average consumers need pretty compelling content to upgrade their systems. Unfortunately, the content isn't quite there yet. However, to help drive the content creators forward, Intel has developed its new Observation Architecture toolset and made it freely available.

One of the disadvantages of using architecture such as OpenGL or Direct3D is that once you pass off a command to the API, it's difficult to tell what's happening from that point onward. Drivers have a nasty habit of queuing up things, such as state changes and rendering commands. You may not see the difference between three calls to render three triangles and one call to render the triangle strip of three triangles. As a result, you may logically decide that converting your rendering engine to use triangle strips would not provide your game with any performance boost. But what you may not know is that the driver is optimized to process triangle strips of eight triangles or more. Any fewer and it breaks them down into individual triangles. Wouldn't it be nice to be able to monitor the driver and see how it's spending its time processing the commands you send it? Well, this is exactly what Intel has done with the Intel740.

Like most modern CPUs, the Intel740 chipset has built-in profiling counters. When you use the Observation Architecture driver, you not only gain access to the hardware counters in the chip, but also to those in the driver and the Pentium II processor. Installation isn't much different from installing any other video card. I'm working with the Diamond Stealth II G460, but any of the Intel740 cards should work. Once the card is installed, you need to get your hands on the Intel740 Graphics Accelerator Performance SDK, which contains everything you need to learn in order

Ron Fosner works on fast 3D rendering code at Data Visualization, a consulting company for those in need of speed, and is the author of OpenGL Programming for Windows 95 and Windows NT *from Addison-Wesley. He teaches a course on graphics code tuning at the (Computer) Game Developers Conference and at the WinDev conferences. Contact him at ron@directx.com.*

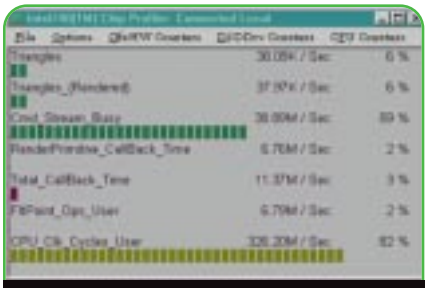


FIGURE 1. *A typical profiling session in OA.*

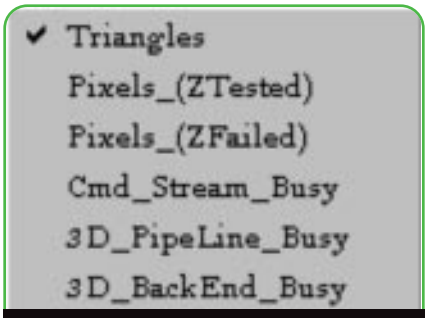


FIGURE 2. *A selection of the chip's counters that you can watch through OA.*

FIGURE 3. *A partial list of the driver's counters that you can watch through OA.*

to use the Observation Architecture (OA). You can then install the profiling drivers, reboot, run a program to set the Intel740 base address, reboot, and you're all set. The best part is that the SDK is available free from Intel.

Once it's installed, you launch the OA Profiler, which is simply a window containing counters of the things you've elected to watch. The counters run in real time, but there is an option to send everything out to a log file. Once the profiler is running, you can either select to profile locally or remotely. This is a nice touch — you only need one networked machine with an Intel740 board in it in order to give everyone the ability to test his or her software on it remotely. After a connection is made to an Intel740 chip, the profiler starts displaying statistics. You have options to display the chip's counters, the driver counters, and/or the CPU counters.

The OA profiler is meant to be used like any other profiler in that you identify a bottleneck globally and then drill down to the subsystem that is taking the most time. Figure 1 shows a typical profiling snapshot. The first three lines are from the Intel740 counters, the next two are from the driver, and the last two are from the CPU. The very last line is labeled CPU_Clk_Cycles_User and shows 82 percent or 326.20 million counts per second. This means that on my 400MHz machine, the User (as opposed to the Kernel) is taking 82 percent of the CPU cycles. In other words, any nonoperating system programs that are running are consuming 82 percent of my CPU's cycles. In this case, I was running multiple copies of Direct3D-based applications.

The next line up shows that these programs are performing nearly 7 million floating-point operations per second — not much. Not much is happening in the driver itself, as evidenced by the small bars. But looking at the chip's statistics, we can see from the Command_Stream_Busy value that it's busy nearly 60 percent of the time (the higher the percentage, the better) and that it's rendering nearly 38K triangles per second, with very little in the way of triangle culling going on: only 120 culled triangles per second. This gives you an idea of the OA's power. One of the hardest aspects of tuning a game is keeping the graphics card busy in parallel with the CPU. Using the OA, it's finally possible to measure how busy your game is keeping the graphics chip *and* the CPU.

Once you get an idea of how the CPU and the graphics chip are spending their time, you can try to narrow down where cycles are being spent by selecting various counters. Figure 2 shows the selection that's available on the actual graphics chip, while Figure 3 shows a partial list of the options that are available in the driver. (Note that these options, especially for the driver, can change from release to release.) For example, you might find out that there's an unexpected amount of AGP throughput, or that the CPU is stalled inside the driver because the driver is waiting for some event. You'd never know about these events because previously, there was no way of knowing about them. The OA Profiler is, hopefully, the first of many interfaces directly to the graphics chip that allow you to tap into the heart of the rendering engine. As more and more functionality gets embedded into the hardware, the ability to access this information is going to become more critical.

Why bother with just the Intel740? Frankly, right now it's the only game in town. Even if you don't think that the Intel740 will build up significant market share (just wait till it starts showing up on motherboards), the fact is that getting these kinds of statistics at all is quite difficult. The real reason you might want to take a look (other than simply being interested in raw speed) is to familiarize yourself with how graphics operations affect the rendering pipeline. Even if you don't need to speed up your program (but who doesn't?), it's instructive to see how well your rendering architecture is

keeping the graphics chip busy. If you're eating up 80 percent of the CPU, but only keeping the graphics accelerator busy 10 percent of the time, you might want to consider breaking up your rendering operations in a more serialized fashion and thereby keeping more data in the graphics pipeline. After all, graphics chips are just specialized CPUs, and the more work you can offload onto them, the more cycles you free up on the general CPU for your game logic. This is the kind of information you get with the OA that previously wasn't available.

There are two caveats. First of all, as of this writing (August 1998), there is no "OpenGL driver" menu item — currently, you can only profile Direct3D drivers. Intel is scheduled to provide an interface to the OpenGL driver with the next release of the SDK. Secondly, tuning games for the Intel740 means doing just that — your results may vary on other chips. However, in most cases when you tune for the Intel740, you'll see improvements in other chips. Intel claims that there's no penalty for changing textures on the Intel740, for example, while most other cards have to dump the pipeline setup and reconfigure. So if you change textures frequently, you'll be penalized more by other graphics chips, especially those on PCI cards, than by the Intel740. But these are small complaints for an otherwise excellent product. If you are at all interested in graphics performance, this is one tool that should be on your desktop. ■

Intel's Observation Architecture for the Intel740 Chip

Rating (out of five stars): ✪✪✪✪☾

Intel Corp.
Santa Clara, Calif.
(408) 765-8080
http://developer.intel.com/design/graphics/740/index.htm

Price: Free from Intel's web site.

Software Requirements: Microsoft Windows 95 OSR2 or Windows 98.

Hardware Requirements: AGP-equipped Pentium or better recommended plus one of the Intel740-based graphics cards. Figure on spending around $100 for the card if you don't already own one.

Technical Support: Web-based

Pros:

1. The OA is a snap to install and easy to use, even on a shared network machine.
2. The OA is an excellent source of low-level graphics information that you can't get anywhere else unless you've got access to a microprobe.
3. Most of the optimizations you'd perform for the Intel740 are applicable to other graphics chips.

Cons:

1. Designed only for Intel740 processors. Remember that the Intel740 actually has a smaller penalty for changing textures than most other cards, so if you're not aware of this, your program may run fine on an Intel740 but behave quite differently on another chip or a PCI card.
2. The utility is running all the time, and it would be nice to perform event-triggered sampling.
3. The tutorial is good, but could easily be ten times larger with more examples, especially some tailored to novice driver/chip spelunkers.

A TRIBUTE TO DANI BUNTEN BERRY

BY BRIAN MORIARTY

As we reported in the September 1998 issue, veteran game designer Dani Bunten Berry passed away on July 3rd of this year following a long illness. She was 49. Dani was presented with the Computer Game Developers Association's Lifetime Achievement Award at the CGDC in Long Beach, Calif., last May. Dani's longtime friend and fellow game designer Brian Moriarty honored her at that ceremony with a laudatory speech, which we have excerpted here.
We'll miss you, Dani. — The Editors

It's fair to characterize tonight's honoree as an old-timer. Her first published title, WHEELER DEALERS, was released back in 1978. This Apple II cassette was rather unusual. It came in a cardboard box instead of a ziplock bag. It sold for thirty-five bucks at a time when most games sold for ten or fifteen. Strangest of all, it wasn't designed for a single user. An array of push-buttons included in the box allowed up to four people to join in a real-time stock market simulation. WHEELER DEALERS sold only around 50 copies. But it marked the beginning of a preoccupation with a design issue 20 years ahead of its time: multiplayer gaming.

COMPUTER QUARTERBACK, published in 1979, was originally designed to support exactly two players. It was ported to Apple BASIC from a mini-computer simulation written in Fortran. In an amusing reversal of recent industry practice, it was the single-player mode that was reluctantly added at the last minute at the request of the publisher, Strategic Simulations.

Nineteen eighty-one saw the release of a second Apple title for SSI, CARTELS AND CUTTHROATS. An economic simulation designed for up to six simultaneous players, the box copy promised that the game was "so much fun you may overlook its use as a superb educational tool." One of the early admirers of CARTELS AND CUTTHROATS was a game theorist fresh out of Harvard with the curious nickname Trip.

The theme of war makes its first appearance in 1982, with CYTRON MASTERS for SSI's RapidFire label. This two-player design offered a curious conjunction of strategy and real-time action in a game that pushed the Apple II hardware to its limits.

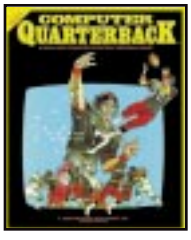
Just after CYTRON MASTERS was released, the aforementioned Harvard graduate expressed a desire to obtain the publishing rights to CARTELS AND CUTTHROATS for a new game company he was launching. When SSI refused to let it go, the original designer gamely offered to produce a superior knock-off.

Nine months later, Electronic Arts bamboozled the industry with a flattering new vision of what computer gaming was all about. Their slick and glamorous promotional campaign turned publishers into record labels, developers into movie studios, and game designers into rock stars. For a few short months, the prospect of fame, wealth, and a matching wardrobe inspired game designers to new heights of personal ambition and creativity; an ideal atmosphere for creating a masterpiece.

M.U.L.E. was multiplayer from the ground up. It used the joystick array of the Atari 800 to connect four people in an unprecedented example of computer-moderated parlor gaming. By combining the resource management of CARTELS AND

Photos (from top): Ozark Softscape and the "We see farther" image courtesy of EA. Images of Dani celebrating a victory and her portrait are courtesy of Editor Johnny Wilson from Computer Gaming World.

Right: Dani's first four published games.



CUTTHROATS, the auctioneering of WHEELER DEALERS and the futuristic setting of CYTRON MASTERS, M.U.L.E. sustained an exquisite play balance of teamwork and rivalry, bitter cooperation, and delicious treachery. Although the original version sold only 30,000 copies, M.U.L.E. developed a base of passionate fans that remains active even today. It is required study for anyone interested in the design of multiplayer computer games.

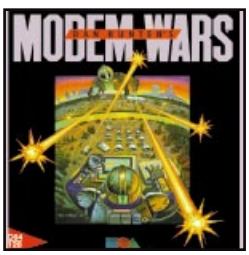
M.U.L.E. was the first title attributed to Ozark Softscape, an Arkansas design collective marketed by Electronic Arts as a hip back-country boutique, computer gaming's answer to the Allman Brothers. Expectations were high after the induction of M.U.L.E. into *Computer Gaming World*'s Hall of Fame. Astonishingly, their next EA release actually lived up to the hype.

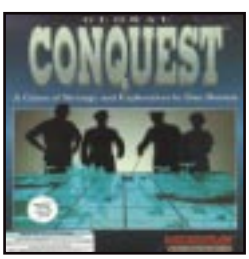
SEVEN CITIES OF GOLD was a solid commercial triumph. It brought together real-time action, strategy, and exploration in a historical adventure with a genuine smudge of educational value. In fact, the much-despised term "edutainment" was originally coined to describe this game. With sales of 150,000 copies across several platforms and numerous design awards, SEVEN CITIES catapulted Ozark into the ranks of the elite developers; and nobody complained about the fact that it was designed for only a single player.

Ozark wanted to follow up SEVEN CITIES with a computerized edition of one of the classic Avalon Hill board games, but Electronic Arts had other ideas. Some executive arm-twisting and a substantial cash bribe resulted in a sequel, HEART OF AFRICA for the Commodore 64, which continued the formula of action and strategy, exploration and history. It achieved less than half the sales of its predecessor. A few years later, another designer tried his hand at that old Avalon Hill game, CIVILIZATION.

HEART OF AFRICA was to be the last product Ozark ever designed for a single user. In fact, their next design took the multiplayer option to a provocative new extreme. Not only did ROBOT RASCALS have no single-player mode, it actually required the participation of no less than four human players. Daringly billed as a "family game," this peculiar fusion of turn-based action and strategy, augmented by a deck of real playing cards, received a polite but puzzled critical reception, and was carefully ignored by everybody else.

A final title for Electronic Arts broke even more new ground. 1988's MODEM WARS was the first game released by a major publisher to support modem-to-modem multiplay. A futuristic synthesis of toy soldiering and football, MODEM WARS was a technical tour de force, offering a surprisingly brisk interactive experience within the severe constraints of 1200-baud modems. Many of the latency and synchronization challenges faced by today's network game engineers were solved first by MODEM WARS.





MICROPROSE took up the cause of modem-based wargaming in a big way with the 1990 release COMMAND HQ, which boasted a simple, clean user interface that made historical strategy more accessible than ever, and racked up impressive sales.

Its successor, 1992's GLOBAL CONQUEST, was the first four-player network game released by a major publisher. Its absorbing mix of real-time action and resource development was the design prototype for an entire generation of combat simulations, including DUNE II, WARCRAFT, and COMMAND AND CONQUER.

The constellation of classic games you see here is just one dimension of a professional career in which the joy of communication has played a central role. Her long list of publishing credits includes columns and articles for virtually all of the leading industry journals. She delivered the first keynote address at the legendary 1988 Game Developer's Conference in Milpitas, and hosted a series of highly-regarded lectures, seminars, and roundtables at most of the subsequent conferences.

In an industry where many celebrity designers have become remote and unapproachable, she has never failed to remain near the center of social activities, freely sharing her company and expertise with the shakers and the shaken.

In the early 90s, this beer-guzzling Arkansas code wrangler undertook a transformation which dramatically exemplified the gamelike nature of social reality. The broadened perspective gained by her friends and business associates as a result of this transformation has been one of her most precious contributions to the industry.

It is no exaggeration to characterize tonight's honoree as the world's foremost authority on multiplayer computer games. Nobody has worked harder to demonstrate how technology can be used to realize one of the noblest of human endeavors, bringing people together. Historians of electronic gaming will find in these eleven boxes the prototypes of the defining art form of the 21st century.

On behalf of the community of game developers and game players worldwide, it is my great pleasure to present this Lifetime Achievement Award to one of the pioneers of interactive entertainment, my courageous teacher and fascinating friend, Dani Bunten Berry. ■

Center photos, (from top): cover shots of M.U.L.E., SEVEN CITIES OF GOLD, HEART OF AFRICA, MODEM WARS, COMMAND H.Q., AND GLOBAL CONQUEST.