

gd

GAME DEVELOPER MAGAZINE

SEPTEMBER 2002





GAME PLAN

LETTER FROM THE EDITOR

Decisions, Decisions

The topic of ethics in games has been coming up more and more recently. There has been a lot of coverage in a variety of media on game violence in general and the case of GRAND THEFT AUTO 3 and its astronomical popularity in particular. One particularly thoughtful example was Ren Reynolds' article that ran on the IGDA web site called "Playing a 'Good' Game: A Philosophical Approach to Understanding the Morality of Games," which used GTA3 as a basis for a discussion on emerging ethical questions in games (www.igda.org/Endeavors/Articles/rreynolds_intro.htm).

Morality? Ethics? In games? Talk like this may be where a serious cleft emerges between those game developers who think they are creating fun playthings and those who think they are the vanguard of an art and entertainment revolution for the 21st century.

Most mainstream critics discussing morality in game design today don't realize that it's not increasing violence or realism that's provoking such discussions, as it may seem on the surface, rather it's an increase in the spectrum of player choices in increasingly open-ended game designs. One of the first things a player of BLACK & WHITE needs to do is decide whether to be a "good" or "evil" deity. In fact, games with sandbox-style gameplay that do not necessarily specify a victory condition, such as BLACK & WHITE or THE SIMS, might not even be properly called games. And it's the clever combination of open-ended pure fun with interesting, linear action that has made GTA3 the critical and commercial darling it is.

Violence and killing have always played a big role in games, whether representational yet clearly fictitious (SPACE INVADERS) or strictly symbolic but based on reality (chess, for example, as a representation of war and complex political machinations). Yet whereas the gameplay possibilities offered by earlier technology usually centered around killing as a means to self-preservation as an end, newer, emergent forms of gameplay leave

motive to be determined by the player. Thus, the open-ended fun of THE SIMS or GTA3 which makes them so popular also forces players to grapple with some unexpected ethical questions.

I felt genuine sadness the first time a Sim — whom I had been toiling for hours to make happy — died while trying to cheer himself up with a cookout. Yet another time, when a Sim to whom I didn't feel much attachment to accidentally married a Sim for whom I had other plans, well it was into the ladderless pool with her, for a protracted, painful death by drowning from exhaustion. I felt a bit bad about it, but I did it all the same, because it was expedient and because the game let me. I even left her grave untended, punishing her Sim soul well into the Sim hereafter for what was, of course, my own slip of the mouse click that got her in trouble in the first place. I'm no philosopher (although I did spend an entire semester in college slogging through the fractured Greek of Aristotle's *Nicomachean Ethics*, so it's possible something might have sunk in), but from a purely ethical standpoint, that's just as bad as flattening innocents with your Yakuza Stinger, it just doesn't carry the same visceral zing that gets pop-culture pundits in a panic.

So while the media pillories games like GTA3 for their realistically depicted and seemingly gratuitous violence, game developers need to start thinking about one of the real issues to have come out of that game's success: the future of emergent gameplay. Developers mustn't focus singly on the task of devising the design tools they'll need to implement emergent forms of gameplay. Those who want to go where no game has gone before should also consider the ethical albatross they're hanging around players' necks by handing them new moral quandaries to face, when linear self-defense is no longer the name of the game.

Jennifer Olsen
Editor-In-Chief

GameDeveloper

600 Harrison Street, San Francisco, CA 94107 t: 415.947.6000 f: 415.947.6090

Publisher

Jennifer Pahlka jpahlka@cmp.com

EDITORIAL

Editor-In-Chief

Jennifer Olsen jolsen@cmp.com

Managing Editor

Everard Strong estrong@cmp.com

Production Editor

Olga Zundel ozundel@cmp.com

Product Review Editor

Daniel Huebner dan@gamasutra.com

Art Director

Elizabeth von Büdingen evonbudingen@cmp.com

Editor-At-Large

Chris Hecker checker@d6.com

Contributing Editors

Jonathan Blow jon@bolt-action.com

Hayden Duvall hayden@confounding-factor.com

Noah Falstein noah@theinspiracy.com

Advisory Board

Hal Barwood LucasArts

Ellen Guon Beeman Beemania

Andy Gavin Naughty Dog

Joby Otero Luxoflux

Dave Pottinger Ensemble Studios

George Sanger Big Fat Inc.

Harvey Smith Ion Storm

Paul Steed WildTangent

ADVERTISING SALES

Director of Sales/Associate Publisher

Michele Sweeney e: msweeney@cmp.com t: 415.947.6217

Senior Account Manager, Eastern Region & Europe

Afton Thatcher e: athatcher@cmp.com t: 415.947.6224

Account Manager, Northern California & Southeast

Susan Kirby e: skirby@cmp.com t: 415.947.6226

Account Manager, Recruitment

Raelene Maiben e: rmaiben@cmp.com t: 415.947.6225

Account Manager, Western Region & Asia

Craig Perreault e: cperreault@cmp.com t: 415.947.6223

Account Representative

Aaron Murawski e: amurawski@cmp.com t: 415.947.6227

ADVERTISING PRODUCTION

Vice President, Manufacturing Bill Amstutz

Advertising Production Coordinator Kevin Chanel

Reprints Cindy Zauss t: 909.698.1780

GAMA NETWORK MARKETING

Director of Marketing Greg Kerwin

Senior MarCom Manager Jennifer McLean

Marketing Coordinator Scott Lyon

CIRCULATION



Game Developer is BPA approved

Group Circulation Director Catherine Flynn

Circulation Manager Ron Escobar

Circulation Assistant Ian Hay

Newsstand Analyst Pam Santoro

SUBSCRIPTION SERVICES

For information, order questions, and address changes

t: 800.250.2429 or 847.647.5928 f: 847.647.5972

e: gamedeveloper@balldata.com

INTERNATIONAL LICENSING INFORMATION

Mario Salinas

t: 650.513.4234 f: 650.513.4482 e: msalinas@cmp.com

CMP MEDIA MANAGEMENT

President & CEO Gary Marshall

Executive Vice President & CFO John Day

Chief Operating Officer Steve Weitzner

Chief Information Officer Mike Mikos

President, Technology Solutions Group Robert Falera

President, Business Technology Group Adam K. Marder

President, Healthcare Group Vicki Masseria

President, Electronics Group Jeff Patterson

President, Specialized Technologies Group Regina Starr Ridley

Senior Vice President, Global Sales & Marketing Bill Howard

Senior Vice President, HR & Communications Leah Landro

Vice President & General Counsel Sandra Grayson

Vice President, Creative Technologies Philip Chapnick



United Business Media

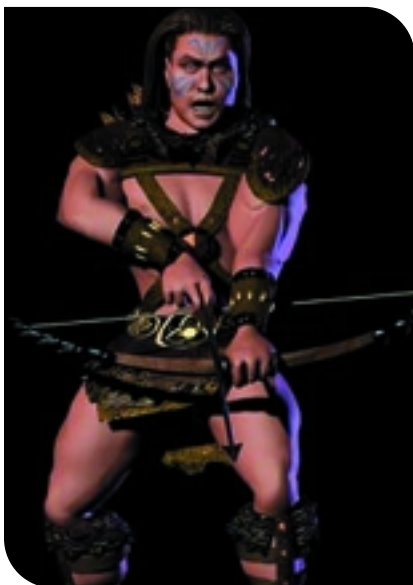
GamaNetwork

INDUSTRY WATCH

THE BUZZ ABOUT THE GAME BIZ | daniel huebner



3DO fights delisting. 3DO, trading at under \$1 and facing imminent dismissal from the Nasdaq exchange, made a number of moves intended to keep the financial wolves at bay — at least temporarily. 3DO will fight to keep its place on the exchange by executing a one-for-eight reverse stock split to push the share price comfortably over the \$1 bar. Furthermore, the company is cutting \$6 million in expenses by relocating its U.S. and U.K. operations, and is bringing in a new \$15 million line of credit, with CEO Trip Hawkins pledging to back the line himself if 3DO is unable to find the additional \$4.6 million in equity it needs to secure the credit. All of this is intended to keep 3DO safely operating through the end of fiscal year 2003, by which time the company believes it will have returned to profitability.



READY TO FIGHT. 3DO, publisher of such games as *HEROES OF MIGHT AND MAGIC IV*, is facing dismissal from the Nasdaq exchange.

Midway shares cut in half. Midway lost nearly half of its stock value in a single day after lowering its second-quarter guidance. The company cut its revenue forecast for the quarter from \$40 million to \$28 million and increased its projected operating loss from \$5 million to \$11.5 million. The revised numbers were attrib-

uted largely to game delays and cancellations, leading analysts to conclude that Midway is suffering from poor management oversight. The company is taking that criticism to heart by creating the new position of chief operating officer to oversee and improve its overall process management.

Vivendi Universal faces breakup.

Financial reporting irregularities and mounting debt may lead to the breakup of Vivendi Universal. Vivendi Universal is the parent company of a wide swath of the entertainment industry, including game makers Knowledge Adventure, Blizzard Entertainment, Sierra Entertainment, and Universal Interactive. The company is fighting to manage a crushing debt of 30 billion euros, while at the same time facing an investigation from French stock market regulators. CEO Jean-Marie Messier and CFO Guillaume Hannezo, two of the men responsible for building Vivendi from a water utility into a media giant, have already resigned.

EA acquires Black Box Games.

Electronic Arts is purchasing Vancouver-based Black Box Games. The 100-person studio, noted for producing sports titles that include Sega's NHL 2K series and Midway's NHL HITS, has worked with EA on the *NEED FOR SPEED* and *NASCAR THUNDER* franchises. Black Box Games will become a wholly owned subsidiary of Electronic Arts. Financial terms were not disclosed.

SCEA takes management online.

Sony Computer Entertainment America announced a change in its management lineup, positioning itself for a battle in the broadband online game arena. Masayuki Chatani was appointed to head Sony's new Broadband Strategy Group, which will work on the broadband gaming initiatives. Jack Tretton was promoted to executive vice president and will oversee SCEA's internal game development group, formerly under the direction of SCEA president Kaz Hirai. Jim Bass, formerly the vice president of



MIDWAY ON THIN ICE? The company, known for publishing such sports titles as *NHL Hitz 2002*, recently lost half its stock value.

finance, was promoted to CFO, while Andrew House was promoted to executive vice president and will continue to work with the third-party publishers.

EA joins the S&P 500. While 3DO faces delisting from the Nasdaq, Electronic Arts is stepping up to the S&P 500. Standard and Poor's added the game publisher to its widely referenced stock index after the close of trading on July 19, promoting EA from the S&P Midcap 400 index. 🐝

UPCOMING EVENTS

CALENDAR

DIGITAL HOLLYWOOD
BEVERLY HILLS HILTON HOTEL
Beverly Hills, California
September 23–25, 2002
Cost: \$495 all access,
\$295 day pass
www.digitalhollywood.com



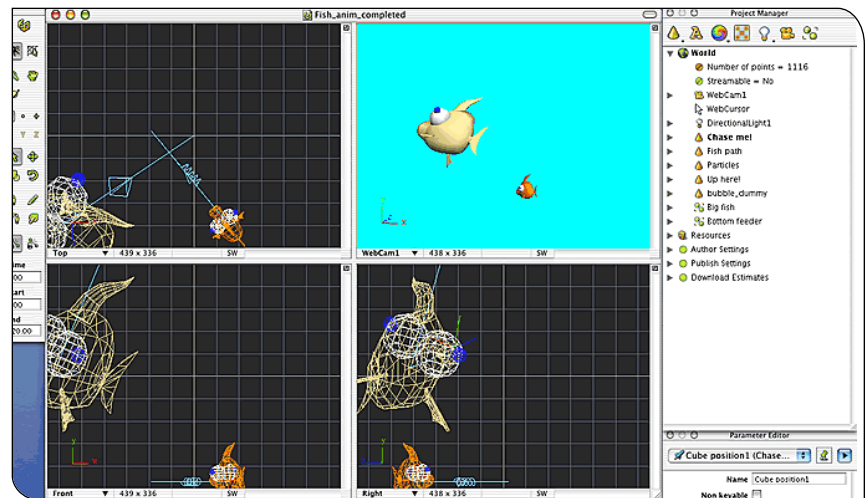
MindAvenue's Axel Edge 1.5

by *steve boelhouwer*

With Axel Edge 1.5, a modeling, animation, and publishing tool available for the PC and Macintosh (OS 9 and X), Canada's MindAvenue continues its pursuit of stalwarts such as Macromedia's Director. Version 1.0 showed promise as an easy-to-use product with a pure web focus, and newly released version 1.5 builds on that foundation. I'll take a look at this new release with an eye toward its suitability for creating web games.

Axel comes in two different flavors: Axel Edge (retailing for \$950) is the flagship product, while Axel Core (retailing for \$350), is the light version, lacking the IK, advanced reactions, and Lightwave import features of its big brother. Both the PC and Macintosh versions have modest system requirements and should run well on virtually any studio machine. Testing on a dual P3-800 workstation with 512MB RAM and an Nvidia Quadro 2 graphics card found the software to be both stable and speedy.

In addition to its animation and interactive capabilities, Axel differs from most 3D web-authoring packages by including a relatively full-featured polygonal modeler. In most other systems, you create 3D content in a stand-alone package and then import the scene file into the authoring system. With Axel you get a full modeling environment with an interface (on the PC) that is a happy combination of 3DS Max and Maya. Object hierarchies, and indeed everything else, are viewable through the Project Manager, a complete tree view of the



Animation and control frames and menus available in Axel Edge 1.5.

entire 3D world and current project settings. Individual object parameters are also easily accessible through a parameter editor window.

Internal modeling tools were important in Axel 1.0, as its 3D import capabilities were limited to VRML. With Version 1.5, you can now import Lightwave objects and scenes, as well as 2D shapes in .EPS format, and a 3DS Max importer is currently in beta testing. Axel's modeling tools include all of the usual suspects, such as extrusion and lathing. Text can be displayed either as extruded 3D geometry or a 2D overlay. Axel Edge also provides a capable skeletal (bones) system with forward and inverse kinematics.

Object shading and texturing is basic but more than sufficient for most web purposes. Vertex shading is supported, as are animated textures (AVI, QuickTime, MPEG, and Flash). Texture maps can be up to 512x512 pixels in size, but projection methods are limited to cylindrical or planar. Environment mapping is supported to simulate reflections. Axel also smartly provides the option to stream large texture files to minimize the initial download time. Also new to this latest release are multiple rendering styles (standard, cartoon, or wireframe) that can be assigned on a per-object basis. Scenes can be lit with directional, point, or spotlights.

Building animation is also straightforward. Axel utilizes a standard keyframe

STEVE BOELHOUWER | *Steve is the vice president of creative services for The Vendare Group, a Los Angeles-based network of game, entertainment, and marketing companies. Contact Steve at steve@vendaregroup.com.*



metaphor, and all animated parameters are represented in a timeline, which Axel refers to as the Sequencer. It's interesting to note that Axel measures animation in seconds, not frames, with the smallest keyframeable time slice being 1/10th of a second. As such, it's not possible to set a specific animation frame rate (the playback system's capabilities determine the frame rate). A capable constraint-and-joint system allows for the creation of logical object hierarchies. The package does not include a physics system per se, but it does include a nifty spring constraint to create a springy joint between two objects. There's also a usable particle

system, and shape morphing for geometry is supported. Individual animation clips can also be encapsulated as an animation reaction, which takes the clip off the main timeline and makes it available for playback based on an interactive trigger such as a mouse click.

Speaking of interactivity, Axel's approach is a bit different from that of similar products. Interaction is built primarily using a visual schematic editor. Similar to the metaphor used in high-end compositing programs, the interaction editor allows you to link sensors (triggers based on time, mouse, keyboard, and other inputs) to reactions (animation queues, sound playback, web page functions, and so on). It's a very intuitive system that should appeal to those artists who dislike using script-based tools — such as Director or Flash — to build interactivity. That being said, this latest release of Axel Edge has a basic scripting utility to allow for the creation of custom sensors and reactions. Similar in syntax to JavaScript, it's far from a complete programming environment, but it allows access to most numerical and Boolean parameters of scene objects and supports a good set of mathematical operations.

Once a project is complete, publishing for the web is cake. The program creates a stream (content) file, which is embedded into a web page using standard <OBJECT> and <EMBED> tags. A download simulator allows content authors to replicate the dial-up experience. Axel can also publish windowless animations if the host browser supports them. By default, the Axel web player renders content using its own software renderer, although hardware rendering (OpenGL) can be enabled.

MindAvenue doesn't have statistics on the installed base of Axel players, but it's safe to assume it's far smaller than web old-timers like Shockwave and Flash. Fortunately, the Axel player download is quick and painless, and Flash export is promised for Fall 2002.

The mostly scriptless interface makes Axel Edge very easy and flexible to use in terms of building projects. For game developers, however, it's also its biggest drawback. The lack of any user-definable

data structures means there are no variables, no dynamic or user-entry text capabilities, and no internal tools to query server-side data. Therefore, common game functions such as scorekeeping can't be accomplished practically with Axel. The built-in sensors and reactions address most of the basic tasks for projects such as an interactive product demonstration, and the basic scripting interface does allow for some customization, but they fall short for creating games of any serious depth.

MindAvenue's web site (www.mindavenue.com) hosts a few clever examples of game content (including a cute first-person shooter where the object is to blast clogged noses with nasal spray) and there are some ingenious workarounds to the limitations of the scripting system.

But game development at that level isn't really what Axel was designed for. Instead, the program excels at what could best be termed "interactive animations." Obviously, many game projects actually fall into this category, and Axel would be an ideal tool for those. It should appeal in particular to web designers, especially those working on a Macintosh, who have limited experience with 3D and/or a scripting language. Its integrated modeler is a plus for those who don't own a stand-alone 3D package, but those developers who need more depth and power will require bigger guns than Axel.

ANALOG DEVICES' SOUNDMAX SMART TOOLS

by todd m. fay

In the past, audio professionals have managed without dedicated middleware tools, relying on in-house toolkits to incorporate and manipulate proprietary audio technologies. However, with more and more developers striving to create relevant audio content for their products, a need for more sophisticated interactive audio tools has arrived.

Analog Devices answers the call with SoundMAX SMartTools. SMartTools grants sound designers the power of so-

AXEL EDGE 1.5 ★★★★★

STATS

MINDAVENUE

Montreal, Quebec, Canada

www.mindavenue.com

PRICE

\$950 (MSRP)

SYSTEM REQUIREMENTS

PC Software: Windows 98/Me/2000/NT 4 (SP 3 or higher)/XP; QuickTime 5.

PC Hardware: Intel Pentium II processor (350MHz or faster); 128MB RAM; 45MB available hard disk space; 16-, 24-, or 32-bit color display adapter; monitor resolution of 1024×768 or greater; three-button mouse.

Macintosh Software: Mac OS 9.2 with OpenGL 1.2 and CarbonLib 1.4, or Mac OS X (v. 10.1); QuickTime 5.

Macintosh Hardware: PowerPC G3 or G4 processor (450MHz or faster); 128MB RAM; 45MB available hard disk space; 1024×768 resolution monitor.

PROS

1. Integrated 3D modeling tools.
2. Intuitive visual interaction editor.
3. Strong web-publishing features.

CONS

1. Embedded scripting language not powerful enough for serious game development.
2. Limited 3D file import options.
3. Web player penetration smaller than competitive products.



called “animated audio.” With it, sound designers can create interactive and non-repetitive audio content. Gone are the days when you hear the bird sample tweeting every seven seconds, the same exact way, over and over and over and . . . you know what I mean.

SoundMAX is based on the Staccato Sound System, an API and toolset released in the late 1990s by Staccato Systems; Analog Devices (ADI) acquired Staccato Systems in 2001. Anyone familiar with Staccato Systems might remember the company as “those guys that make that audio thing for racing games . . . So what? I don’t make racing games.” Well the ADI ARTG (Advanced Rendering Technology Group) is busy creating audio tools beyond physically modeled car engines. And let me tell you, you won’t believe your ears when you hear what you can do with SMartTools.

So what exactly is included? Right now SMartTools ships with two tools for sound designers (along with the API for audio coders). The first, SMartAnimator, relies on a series of audio playback algorithms to create nonrepetitive audio streams. The other tool is the SMartSynth, a sound generation tool that uses physical modeling synthesis to create realistic sounds. Both tools are professional grade and would cost you a ton of money to acquire. That is, if ADI weren’t giving it away for free. (More on this ingenious pricing later.)

SMartAnimator currently ships with one algorithm: Crossfader. Crossfader employs complex sets of (you guessed it) cross-fades and other parameters, which can react to the game state in real time. This algorithm is built with persistent sound effects in mind, such as engines, crowds, and other ambient effects. It’s slightly annoying that some of the parameters are named for engine sound controls, but this is a small price to pay.

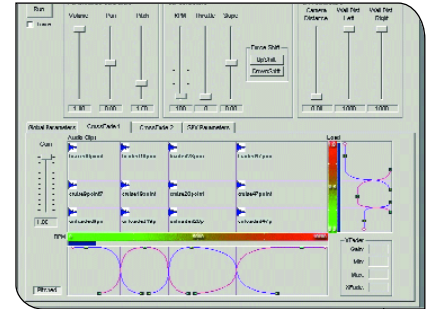
A session with SMartAnimator goes something like this: You create your sound palette like you would in any other situation. Once you have your sound set together you import them into SMartAnimator and use the easy-to-comprehend GUI to design “sound behav-

iors.” You then test out your sound behaviors and export your final file. The audio coder drops the file into the latest build, and magically you have animated audio for your game.

The toolset is cross-platform between PC, Playstation 2, and Xbox, with each platform having a dedicated export button built into the SMartAnimator GUI. You create your audio behaviors once in SMartAnimator, click on the export button for the platform your title is shipping on, and that’s it. Once you export the behavior for one platform, you can export for the other two as well simply by clicking on the appropriate buttons. How easy are they going to make this for us?

ADI is working on three additional algorithms, all of which will serve unique purposes. ParticleBurst will be ideal for nonrepetitive one-shots, ParticleFlow will randomly recombine short sound clips and is designed with soundscapes and ambience in mind, and ParticleCycle will allow sound designers to create nonrepetitive cyclical sounds (footsteps, machine-gun fire, and the like). Assuming this last one is an improvement on the footstep algorithm I heard at the Game Developers Conference, we should all be very, very impressed. Keep in mind I have yet to test these in-progress algorithms, but if they function as well as Crossfader does, things are looking up for game audio content creators.

If you are familiar with the progression of this technology, beginning with its roots in the Sondius project through its first incarnation under ADI, you know that SoundMAX game audio tools, due to their dependency on physical-modeling (read: highly processor intensive) algorithms, weren’t always ideal solutions. Well, the SoundMAX team reevaluated their approach to developing audio tools for game developers, and SMartTools is now a true success. Developer-friendly, SMartTools’ newer algorithms depart from strict reliance on physical modeling for real-time sound generation, depending more on event-modeling synthesis (the SMartAnimator algorithms, for instance).



SMartAnimator, one of the ingredients of the SoundMAX SmartTools.

Physical-modeling solutions are still available through the use of SMartSynth. The focus has shifted on using SMartSynth to generate sound clips for use in SMartAnimator. The synth models sound outrageous. SMartSynth is a great solution for creating a variety of sound effects on a strict budget. It is also perfect for creating common sound effects that aren’t canned (read: licensed from a sound FX library).

These tools are extremely easy to use. Let’s face it, it doesn’t matter how great a particular technology makes your game sound. If it’s a monumental task for your audio team to learn the tool, it just isn’t worth it. It’s a huge improvement over ADI’s first attempt at a SoundMAX toolset. Additionally, the toolset and the API are meticulously documented in clear language.

This is the part that will convince your producer why this cool audio app belongs in your company’s next game: It’s free. What’s the catch? ADI wants their SoundMAX logo on the game’s box, a splash screen, and on sell sheets. Not a bad deal.

This technology has created a world of unparalleled potential for interactive sound designers. It’s pretty obvious to me that, given the newness of this system, game audio developers haven’t even scratched the surface of what SMartTools can do for interactive sound designs. 🎧

★★★★★ | SoundMAX SmartTools
Analog Devices | www.soundmax.com

Todd M. Fay (a.k.a. LAX) is an audio content creator and author. Contact him at todd@lax-element.com.

Laura Fryer Taking Care of Developers

Laura Fryer is the director of Microsoft's Advanced Technology Group for Xbox. She started out at Microsoft 10 years ago supporting games and multimedia titles before moving to Microsoft Games Studios in 1995. There she helped launch the MSN Gaming Zone and conceived and produced the first Microsoft release of the Zone. She then went on to produce numerous titles, including FIGHTER ACE and CRIMSON SKIES. In May 2000, while Microsoft was laying the groundwork for Xbox, she joined the Advanced Technology Group and helped build the team — and a new console — from scratch.

Game Developer. Is it a big responsibility having Seamus Blackley's old job?

Laura Fryer. Running ATG is a big responsibility, but my day-to-day job hasn't changed that much since I joined Xbox. When we started the Advanced Technology Group, we split the responsibilities. Seamus focused on his evangelism role and I focused on hiring, building, and managing the worldwide team with the help of people like Mark Thomas. So, the transition itself was very smooth because Seamus and I basically continued doing what we'd always been doing.

GD. What are the main goals of the Advanced Technology Group?

LF. Our mission is simple: we want to help developers make great games. We do that in a lot of different ways; for example, we provide whitepapers, tools, samples, developer events, and the fastest turnaround on questions in the industry. We also provide unique services like code performance reviews. Xbox developers are smart and can figure out their own bottlenecks, but we can save them a week and will probably find issues that they haven't seen before given how much code we review.

GD. What about for other members of the development team?

LF. If a developer wants to understand how to get more fidelity out of their textures or how to fully take advantage of the audio chip, they can contact our experts any time during the ship cycle and get help. We also provide a game evaluation team which can play the game and give feedback on specific issues, like solving a control problem, or general gameplay feedback.

GD. Give them half a chance and developers can usually find something to complain about when it comes to working with console manufacturers. What have Xbox developers taught ATG since



Laura Fryer, director of the Xbox Advanced Technology Group.

its inception, and how has ATG incorporated that feedback into its process?


LF. Developers never complain, they suggest! Seriously, ATG's job is taking care of developers, so we love getting feedback and we're always looking for new ways to improve our offerings. Everything from the hardware to the certification process has been improved by developer feedback. One of the tools we're working on right now enables artists to preview their art on the Xbox, and that tool is the result of feedback our art director, Dave McCoy, received from the Xbox artist community.

GD. With your bird's-eye perspective, what major differences do you see in approach between smaller developers and larger ones?

LF. In general, larger development companies are better funded and are therefore able to hire specialists for each area of the game. Smaller teams have to wear multiple hats, where you have a programmer that is also the lead designer

or an associate producer that acts as the chief tester. If you have a small team, you need to really focus on hiring, because a single mistake can cost you the project. Both types of development teams have passionate people working on them, but smaller teams are usually a little hungrier and more focused because if they don't ship their project, they may not have jobs. I really admire the small shops that break through with great games, because they bring a lot of vitality, creativity, and passion to our industry.

GD. Tell me about some of Microsoft's efforts to get more women involved in the game industry.

LF. This year we held an event at GDC for women to get together and network called "Celebrating Women in Gaming." It was a huge success and gave women a chance to meet and discuss issues that they saw in game development. We also sponsor several annual events to increase awareness about technology jobs for women. Last year we participated as a company in the "High Tech Camp for Girls," where we bring in high school girls interested in technology and give them presentations around the company. The teachers had trouble getting the girls to voluntarily leave our game evaluation lab! Hopefully in a few years we'll see some of those young women again as employees. 

Character Matters: Part 1

The Human Figure

The room was well lit if somewhat stark. The basement of what was once the town hall hadn't been built for comfort, and the heavy stone walls had been painted white so many times that the room was probably six inches smaller because of it.

The bare wooden floors, warped and spotted with more than a century's worth of unidentifiable stains, seemed barely solid enough to hold the weight of the 14 nervous teenagers glancing around uncomfortably under the fluorescent lighting. It wasn't that we didn't want to be there; on the contrary, most of us would have sold our grandma's best teeth to be granted a place there. But the unknown is always frightening, and we all watched the door, anxious in our anticipation.

Finally, after what seemed to be an age, the heavy oak doors swung open and in she stepped. Heart rates around the room soared to dangerous levels. Mouths went dry as 28 palms began to sweat. Confidently and without the slightest hesitation, she walked into the corner, removed her robe, and stood before us naked.

My first life-drawing class was both educational and unnerving. The timing of it was perhaps partly to blame. Mid-teen hormones make for a rocky ride at the best of times. Add in a two-hour session staring at a naked woman, and there was bound to be some danger of chemical overload.

Fortunately, I survived without serious incident, the worst moment being the teacher's very public ridicule of the fact that I used a 5H pencil in a life-drawing class (she was obviously trying to stifle my self-expression). Was it worth it? I think so. I learned two important things: First, how to draw the human figure, and second, that it is impossible to draw a

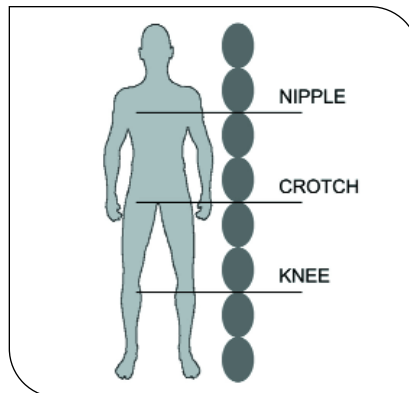


FIGURE 1. The human body follows fairly strict rules when it comes to proportion. This male figure stands eight heads tall, with other physical features occurring naturally at the intervals indicated.

nude female without appearing to stare at her nipples.

Drawing the human figure has long been an integral part of any classical artist's training. I doubt that even now any legitimate art school would be without life-drawing classes or some variation thereon. Understanding the process behind drawing people, among other things, educates a student in disciplines such as foreshortening, weight, mass, light, and composition, all of these concepts being invaluable to the well-rounded artist.

But is this relevant to us? Do we need these skills when working in the game industry? The power of our tools has, in many ways, removed the need for an artist to be versed fully in all aspects of

traditional art. Is there any real value to be had walking in the past footsteps of the paint-pushing, one-eared masters of the visual arts? Will the lessons they learned bring us the same rewards? The answer is both yes and no.

Having an accurate understanding of the final color produced by mixing burnt umber and cadmium yellow won't get you very far in Photoshop. Appreciating non-medium-specific concepts, however, can elevate someone working on a game's visuals from being a technician well-versed in the operation of extremely advanced software, to an artist. What's the difference? This is perhaps an argument for another day, but one area where a traditional understanding of artistic technique can be most helpful to the game artist is that of creating a character.

It is almost too obvious to point out (but I will anyway), that characters in a game can be anything from a yellow disk with a wedge cut out to a rather stunted Italian plumber with an unfashionably bushy mustache. Character design and creation is a vast topic in itself, and much has been written on the subject, but here we will look at the rules involved in creating a human character as they are applicable to a game aimed toward realism.

Proportion

As far as the human figure goes, there is a variety of methods for maintaining correct proportions, but perhaps the easiest and most commonly



HAYDEN DUVALL | Hayden started work in 1987, creating airbrushed artwork for the games industry. Over the next eight years, Hayden continued as a freelance artist and lectured in psychology at Perth College in Scotland. Hayden now lives in Bristol, England, with his wife, Leah, and their four children, where he is lead artist at *Confounding Factor*.



FIGURE 2 (left). Differences between the male pelvic bone (red) and the female (blue).

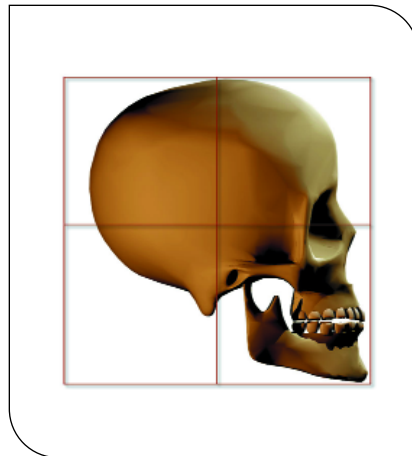


FIGURE 3 right). The human skull is just slightly less deep than it is tall.

used is looking at the body in terms of the human head.

The female form is proportionally different from that of the male, but on average, a figure is approximately seven and a half heads high. Male figures are often depicted as being eight heads high, and women seven, with the tall, powerful male stereotype being pretty standard in games (Mario being a notable exception).

As you can see in Figure 1, the crotch is located at three heads below the chin, which, when using a total of eight head lengths for the male, is exactly the halfway point. Interestingly enough, the nipple line is around the two-head mark, the navel at three heads, and the bottom of the knee area at six heads. While this may seem a very rudimentary measure of proportion, it works surprisingly well, subject to alteration if unusually long legs or some other unique feature needs to be accommodated.

In addition to height measurements, male shoulders are approximately two head lengths across. Female shoulders measure around one and a half head lengths across.

Chest and Pelvis

The trunk of the body gives a feeling of mass and solidity to a character and is defined by the two large bone

structures that underpin it: the pelvis and the rib cage. Ribs are essentially arranged in the same way for both men and women. The pelvis, on the other hand, is higher and narrower for the male, and shorter but wider for the female, as Figure 2 shows.

The barrel shape of the rib cage defines the primary mass in the chest area, while the pelvis governs the width of the hip area. Female hips are about equal in width to the distance across their shoulders. Male hips are somewhat narrower than the shoulders.

Limbs

In terms of proportion, once again using the head as a unit of measurement, the distance from the top of a male figure's head to his fingertips, when his arms are at his side is about five head lengths (for a female it is about four and a half). Legs are about four head lengths, crotch to toes, in the male, three and a half in the female.

Head

Barring deformity, the average human head when viewed from the front is just less than one and a half times as tall as it is wide. Often, problems in head construction arise in terms of depth, where a

head can either stretch too far backward and end up looking like Giger's alien, or be squashed too far forward.

Looking at the human skull (which, funnily enough, governs the shape of the head) in Figure 3, we can see that the distance from its back surface to the front of the jaw comes to about 90 percent of the distance between the top of the head and the base of the chin.

What often leads to misjudgments in this respect is the fact that the neck (which starts from about halfway along the lower jaw) extends all the way back to the rear of the skull, with only the rounded extremity of the occipital plane (that's the back of the skull) overhanging.

As far as neck width from left to right is concerned, it's clear that the female neck is narrower than that of the male. Still, both are essentially defined by the sterno-mastoid muscles, which come from behind the ear to the base at the front of the neck, and the trapezius muscles, where the neck joins the shoulders on either side. Proportion here depends on the musculature of the specific figure under construction, but the usefulness of the neck to convey strength and power or grace and elegance should not be underestimated. The most beautiful of faces perched on top of a Mike Tyson neck is unlikely to work, and vice versa.

Face

Finally, we come to the face. It often disappoints me how little apparent effort is put into the faces of some game characters. In the real world, we are vastly more interested in looking at people's faces than we are other areas of their body (I know what you're thinking, but the evidence is quite clear).

If your game offers little chance of ever getting close to another human before you shred them with your twin-barreled, plasmatronic Annihilator 4000, facial features are unlikely to play a significant role. However, it is important to remember that a face is the center of emotion, the hub of a body's senses, and a focus for communication. If we are

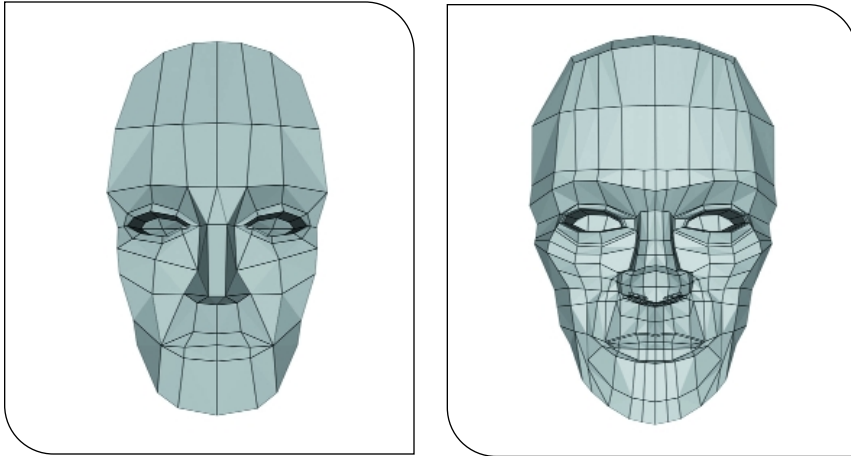


FIGURE 4. Low-polygon heads (left) of games past have given way to greater detail available for features modeled in geometry, such as nostrils (right).

aiming to create characters in a game that actually have character, we cannot afford to neglect the face.

Until fairly recently, the vast majority of facial definition in games had to be done in texture. It was not uncommon for a nose to be little more than a six-triangle protrusion with the mouth hardly modeled at all. Today, we bask in the luxury of polygon-pushing hardware that lets a face become the artist's playground. This is not to say that every eyelash and every fold of skin should now be fully modeled; on the contrary, texture detail will still add much to the definition of a face (Figure 4). But at the very least, we can finally go crazy and add nostrils.

There are a number of ways that the layout of the face can be summarized. One of these is to use a single unit of measurement as a point of reference, as we did with the head when looking at overall bodily proportions. The most obvious thing available is the eye.

As you can see in Figure 5, the eye-line is classically halfway down the face. At this point, the average head is about five eyes wide. The central eye space is essentially the bridge of the nose, the two spaces to either side of this are where the eyes are actually placed, with the spaces on either side of the eyes reaching the outside of the head. None of these measurements is meant to be exact — faces vary

from one person to the next, and more so between different races — but these proportions are basically accurate.

Still using the eye as a reference, the width of the base of the nose is approximately the same as that of the width of an eye, and the distance from the base of the nose to the bottom of the lower lip is also about the same as the width of one eye turned vertically.

Additionally, an inverted equilateral triangle, whose top edge goes from the

outside corner of one eye to the outside corner of the other eye, should find that the other two edges meet just below the lower lip.

These rules can help in the correct placement of facial features relative to each other. Humans are so sensitive to nuances of others' faces that even a small error can result in the kind of misaligned face that only the unfortunate stars of "When Plastic Surgery Goes Wrong" possess.

Symmetry

All in all, the whole of the human body is remarkably symmetrical. In real life, subtle variations in feature placement, limb length, blemishes on the skin, and other minor imperfections unbalance this symmetry slightly. For game characters, however, tiny differences of this kind should generally go unnoticed.

Mastering the human figure in three dimensions is not an easy task, but learning the basic rules of proportion and becoming familiar with rudimentary anatomy will make the job easier. Next month, I'll discuss the process of taking the human figure in to 3D as a fully modeled character. 🎨

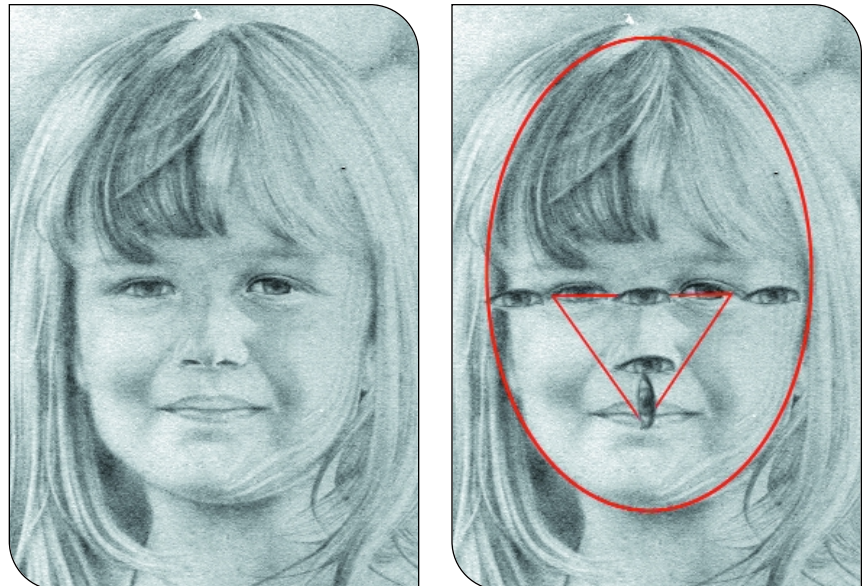


FIGURE 5. A drawing of a face, shown normally (left), and enhanced with eye-based proportional measurements (right).

Shoot your Troubles

Say you've just bought a piece of software or hardware for your audio rig to crank out your next triumph of a soundtrack or sound set for your latest title. Or perhaps you're excited to be working on audio for a console game and you get to use the magic SDK that comes with it in addition to whatever third-party software is available. What happens?

Typically, you pop on the web or trundle down to your local pro audio shop. And when you first get that shiny new box, this is where things, for a few of us, get confusing. Don't laugh — its true.

Some people tear open that box, plug everything in, and hope that it works. Sometimes, this magically happens. Most of the time, though, it doesn't. When it doesn't, whom do you talk to? Where do you go? When something breaks, how do you fix it?

These may sound like simple questions, but whereas sometimes getting a problem solved is as easy as picking up a phone, opening a web browser, or firing off an e-mail, other times it's the worst kind of hell on Earth. Following are a few tidbits that can help you streamline the inevitable troubleshooting experience, so you can get yourself back into the game more quickly.

Before you dismiss the following information as too simple to be effective, keep in mind how many people don't actually do these things, including you.

Let us audio gods forsake our overblown egos and realize that the eagerness to get our rigs up and working in a split second can blind us to the fact that our gear needs regular maintenance and, at times, just as much attention as a car when it breaks down. You might be amazed at how much faster you can get things rolling following these simple steps, in order:

Analyze the problem. Isolate it. Make

sure you know what's broken. Don't stop with "It doesn't work."

RTFM. Although doing so is as appealing as drinking gasoline, read those blasted README files and manuals that come with your product. Manufacturers do take the time to put very helpful information in them.

Check the web and newsgroups. Console folks usually have oodles of online information to wade through. Download the latest drivers and patches, look at the FAQs, and search around manufacturers' web sites as well as user-run sites.

When you first get that shiny new box, this is where things, for a few of us, get confusing.

Get on the horn. If none of the above leads to a conclusion (granted, those FAQ files and updates aren't all perfect), its time to call or e-mail someone. Some vendors, such as Sweetwater Sound, staff tech experts who can solve your problem without having to go to the OEM (original equipment manufacturer), but if you buy direct or through vendors without that added bonus, you need to call the people who built your gear. Contact information for company representatives is usually readily available on the web or in the packaging. If you find someone was particularly helpful, remember his or her name and put it in your address

book for (the inevitable) next time.

Let me briefly describe how I went through this process just the other day when I was dealing with my PC/Roland XP-80 setup. Like any programmer who wants to take an SDK or dev kit and gleefully set it on fire when bugs overwhelm him or her, I was ready to do the same to my gear.

My previous knowledge of MIDI sequencing led me to believe that all one had to do was put the synth workstation in "Local Control: Off" mode to control it with a sequencer on the PC. Not so with the XP-80 and Cakewalk's Sonar. I couldn't for the life of me select the patches I wanted or change banks. I couldn't even send information to more than one MIDI channel. I've used Cakewalk products for 10 years, so this situation understandably made me feel a bit, well, green.

After about an hour of frustration fiddling with buttons and drop-down menus, I took a deep breath and followed the preceding troubleshooting steps. After figuring out exactly what I needed to do, I found a link on Sweetwater Sound's Knowledge Base (www.sweetwater.com/support/ts) that described a procedure. Let me tell you, it was no easy solution, but it did work. Even though I was ready to call someone up and let them taste my pain, it turned out that taking it easy and being more methodical worked after all. While this may be hard to accept for a musician, now I'm back to writing again, which is what I wanted to be doing in the first place. 🎸



ALEXANDER BRANDON | Alex is busy gathering old game soundtracks for a massive compilation and is the audio director for Ion Storm Austin. He is also a member of the board of directors for the Game Audio Network Guild (www.audiogang.org) and is on the steering committee of the Interactive Audio Special Interest Group (www.iasig.org). Contact him at alex@ionstorm.com.

The Story So Far

In the July 2002 Better by Design, I posed a challenge. I set forth the rule “Let players turn the game off,” which stated that the player should always be able to save and exit the game at any point, losing at most a few seconds of progress as a result. Then I invited readers to comment on other rules that trump this rule or are trumped by it.

Most of the responses I’ve received were emphatic in their endorsement of this rule. More than one reader said that there should never be any reason to require the player to replay a section of a game. Although there was one suggestion dismissing the save-game issue as a religious one with no solution, even that person agreed with the substance of the rule. The more I delve into applying rules to game design, the more I find that characterizing conflicts as insoluble “religious issues” is just a convenient way to dismiss thoughtful discussion.

Disbelieving disbelief. I had suggested that maintaining suspension of disbelief might justify freezing out the save-game option for a period, to keep from reminding the player that it’s only a game. But Jin Choung of Gigawatt Studios challenged me on that one. He reminded me that “the alternative [to saving at any time] for a poor player is having to repeat a segment of game over and over and over. That is not immersive or fun. For a good player, the effect would be heightened suspense. For the bad player, it ends in tedium and monotony.” He’s right. If we want to keep growing our audience, then certainly our obligation as developers is to support the novice or subpar players at least as much as the experienced, talented ones. Jin’s argument implies that our biggest win comes from making the save-game process so smooth and transparent that players can use it intuitively without conscious consideration.



HALO’s frequent checkpoints enable quick saves.

Take small bites. Two readers suggested that the technical limitations of consoles or hand-held systems sometimes make it easier to limit saves than to allow them at any time. But both pointed out that this is the lazy developers’ excuse for the problem, and it’s better either to find a solution to the technical problem or to come up with a more elegant design that finesses it. David Bunnett of Stormfront Studios was particularly eloquent about this, pointing out that HALO had “solved this problem effectively with frequent checkpoints at which the action is briefly abated and games can be restored.”

Breaking up large action segments of a game into smaller segments is good advice in most cases. There’s a rule lurking in there, trumped perhaps by the occasional specific need to arrange those smaller segments into a larger continuous whole in order to move a story along or build tension or excitement in a critical sequence of a game. Certainly it’s always a good idea to break up long, noninteractive segments into shorter pieces or

find ways to achieve their purpose in the interactive part of the game.

David 3, Goliath 0. David Bunnett had some other good points I’d like to relate. He came up with one of the best potential exceptions to the rule — extreme sports games that depend on the player putting together elaborate sequences of moves. “Such sequences wouldn’t be meaningful if players could save in the middle; they have to be live end-to-end to be fun,” he says.

This caveat could apply to any game with a time-critical sequence of moves. It would be tempting to try to come up with specific exceptions. But David was ahead of me with that one and suggested this refinement of the rule: “Hold the frustration and inconvenience associated with saving and quitting to a minimum, but use common sense when doing so. Don’t cut the heart out of your game just so players can save at any time. Do consider the genre and platform(s) of your game when designing the save feature.”


The sixth sense. Perhaps the best advice embedded in that rule is the concept of using common sense. “Use common sense when applying rules” is a great uber-rule with tremendous trumping power. No rule is so universal or flawless as to preclude the application of common sense. Of course, that can apply not just to game design, but life in general. Can we apply other game design rules to our lives? Maybe I’ve spent too much time in front of the computer. After all, real life doesn’t have saves. That reminds me, how long has it been since I backed up my computer? 🐾



NOAH FALSTEIN | Noah is a 22-year veteran of the game industry. You can find a list of his credits and other information at www.theinspiracy.com. If you’re an experienced game designer interested in contributing a game design rule to this column, please e-mail Noah at noah@theinspiracy.com (include your game design background) for more information about how to submit rules.

Take the following scenario: Fully decked out in your battle armor, you head down the smoke-filled corridor of the latest gothic/ techno dungeon you are attempting to plunder. You reach the edge of a deep pit, whose bottom is lined with sharp spikes that could do your body a great deal of harm. Jumping over the pit laden with all your battle gear is out of the question, and you're not going to face the demon hell-kitty without it.

Luckily, there is a long wooden plank in the wreckage of that tavern you just destroyed. Not as strong a board as you would like, but it will have to do. You fetch it and stretch it across the span. As you take a step, the board wobbles and creaks a bit, but it seems like it will hold up. On your next step, the board creaks even louder. You debate whether to lighten your load a bit but decide to take a chance. On the next step the board breaks, and body piercing has just taken on a new meaning.



Continuum Mechanics: Bending Stuff Past the Breaking Point

Like many game developers, I really enjoy building stuff. I don't mean a polygonal tree trunk or a bathing beauty built from a multi-resolution mesh. I'm talking about building large physical objects, things that you can actually hold in your hand and would hurt like crazy if dropped on your foot. It probably has something to do with the way that the virtual world where most of us spend our days building things competes with our primitive needs to create physical stuff that we can wave around and thereby look impressive. Often, however, it's as much fun to bend and break things as it is to build them.

The virtual 3D worlds we create for today's games look fairly nice: there are impressive amounts of detail all over, and people move around in a fairly realistic manner, interacting with each other and the environment. With only a few notable exceptions, however, these worlds are largely static and unchangeable. Even when you can break something up, it happens in a largely predetermined manner: an object explodes in a shower of particles of fire and smoke to reveal a new, "broken" version of the object. We need to go quite a bit beyond this level of interactivity to appeal to the builder-destroyer in all of us.

Step Inside the Continuum

In order to make a dynamic scenario where wooden planks wobble, creak, and possibly break, we need to understand how objects behave when various forces are applied. The study of materials subjected to external influences is known as continuum mechanics. Bonet and Wood's *Nonlinear Continuum Mechanics for Finite Element Analysis* (see For More Information) provides a complete explanation of the summary of continuum mechanics that follows.

External influences can be a variety of different things, from very common forces such as gravity to a foolish player jumping up and down on a rickety bridge. Other influences could be more abstract in nature, such as fire or a corrosive chemical that can change the strength of a material.

For this discussion, an object represents a continuous material that is connected by internal forces. On an atomic level, interaction between the atoms that make up the object hold it together and give the object its strength. If the object has the same internal strength in all directions, it is said to be isotropic. For simplicity, I will be dealing only with isotropic materials. However, it is important to understand that many materials, such as wood, can behave in a much more anisotropic manner.

Material can be further classified by how it behaves once deformed by external applied forces. If the object returns to its undeformed shape once the applied forces are removed, the object is said to be made of an elastic material. If the object

does not return to its undeformed shape when the forces are removed, it is made up of a plastic, or inelastic, material. For example, a plank of wood is generally an elastic material, while a lump of clay is quite plastic. It should be obvious, however, that the properties of a material can change based on the same external forces that can deform it. For instance, a sheet of glass is generally elastic and breaks easily. However, once the external influence of heat is applied, it becomes much more plastic and can be bent to retain a shape.

To understand how this works, let me take the example of a small piece of bungee cord. When the cord is at rest, it will have an initial length, L , and a surface area, A , as shown in Figure 1.

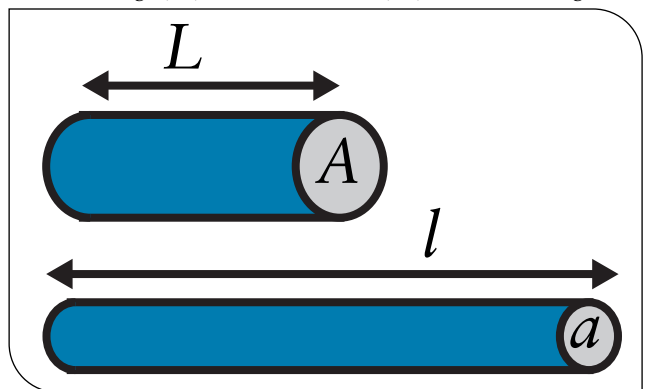


FIGURE 1. A bungee cord at rest (top) and stretched (bottom). The strain on the cord is one component of continuum mechanics.

Once stretched by pulling on each end, the cord increases in length, l , while decreasing in area, a . As anyone who has ever stretched a bungee cord knows, the act of stretching the cord creates a large strain on the cord. If the strain becomes too great, the cord can break. In order to be useful, it's important to be able to measure this strain. One easy measurement of strain is the change in length of the material divided by the rest length. This is commonly known as the engineering strain, defined by the notation ϵ_E .

$$\epsilon_E = \frac{l - L}{L}$$

Engineering strain is just one example of a measurement of strain. Another similar example is Green strain, ϵ_G , named after George Green, the early 19th century mathematician. While the Green strain formulation is a bit more complicated, it can be useful for some applications:

$$\epsilon_G = \frac{l^2 - L^2}{2L^2}$$

For measuring the strain in a one-dimensional system such as a spring, Green strain is probably not appropriate, since it acts like l^2 . However, for 2D and 3D problems, it performs better than simple engineering strain.

Another important concept in describing continuum mechanics is the notion of stress, σ . Stress is defined as a value of force per unit area. Stress is related to strain by the use of a proportionality constant known as the Young's modulus (or elastic

JEFF LANDER | When not spending his time developing PC and console games, Jeff is busy working on building his first house. Hopefully he can make it a bit stronger than his virtual stuff. Send your master carpentry tips to jeffl@darwin3d.com.

modulus), E . This constant, usually a pretty large value, can also be thought of as the “stiffness” of the material. For a simple linear, elastic material, stress is defined as:

$$\sigma = E\varepsilon$$

$$\sigma = E \frac{l-L}{L}$$

This is the first part of Hooke’s law of elastic materials. This is the same Hooke who described the behavior of spring forces that we often use in game development (see For More Information). The second part of the law describes the fact that there is a strain contraction that is perpendicular to the stretch direction. This proportionality is a constant known as the Poisson’s ratio, Δ .

$$\frac{\Delta w}{w} = \frac{\Delta b}{b} = -\nu \frac{\Delta l}{l}$$

$$0 < \nu < \frac{1}{2}$$

For a bar with a width w , height b , and length l that is deformed, the strain in both w and b is equal and proportional to the strain in the direction of stretch, l . Which, said differently, is simply the relationship between the elastic properties of a material. This constant is generally a positive number less than 0.5. If the Poisson’s ratio of a material were equal to 0.5, the material would be incompressible and its volume would remain constant no matter how much it was deformed. If a ratio of less than 0.0 were used, the material would actually gain volume as it was deformed.

That adds up to quite a few formulas and terms to throw out there, but the concepts are actually very simple. Together, the Young’s modulus and the Poisson’s ratio describe the properties of the material. The two constants are sufficient to account for many of the differences between materials such as metal and rubber and pretty much any kind of elastic material you would want to simulate.

London Bridge

A couple of years ago there was a shareware game released called *BRIDGE BUILDER* by Alex Austin, now of Chronic Logic. This game allowed you to create a bridge by connecting steel spans across various obstacles. This level of physical simulation is a simple example of the kind of dynamic deformation that can be created using the definition of stress I described above.

Back in March 1999, I wrote a “Graphic Content” column for *Game Developer* on mass-and-spring dynamics. In that article, I used point masses that were connected by springs to simulate dynamic soft-body objects. A spring was defined as a connection between two point masses. The rest length between the two points was saved at initialization. As the simulation progressed, the points were free to move around, subject to external forces such as gravity. As the points moved, the spring forces were then applied. The spring force acting on a particle, p , took the form:

$$f_p = -k_s(l-L)$$

In this formula, the force acting on the point p is equal to a negative spring coefficient times the current length of the spring minus the rest length of the spring. This formula requires much of the same information as the definition I gave for stress and strain. I can easily calculate the value of stress on any particular spring in my simulation. This measurement of stress gives me an easy picture of what is going on with all of the springs. When the stress of a particular spring is equal to 0, the spring is at its rest length. If the value of stress is positive, the spring is being stretched. If the stress value is negative, the spring is in compression.

I can establish an arbitrary failure point for the springs in the system. For example, if the springs are stretched too far or compressed too much, they could break. As an additional control over the simulation, the Young’s modulus can be used to define different types of springs that are made up of different materials.

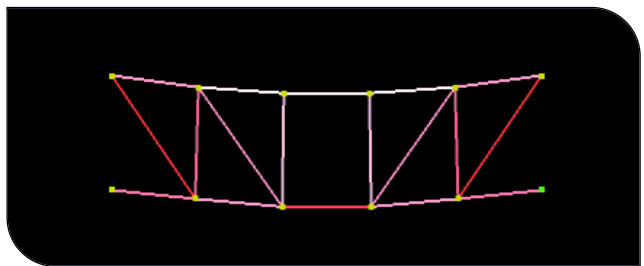


FIGURE 2. A rope bridge made of springs. The force of the stress on the ropes increases from white to red.

You can see an example of my “civil engineering” simulator in Figure 2. In this simple example, I have created a rope bridge. The calculated stress values in each spring have been color coded from white, representing no stress, to red, representing the breaking point for the rope. By adjusting both the spring constants, the Young’s modulus, and the stress breaking point, I can create a bridge that simulates everything from stiff and brittle to flexible and forgiving.

Elementary Problems

It may seem like this system would be enough to create all sorts of deformable and destructible objects. For the most part that is true. However, anytime you start trying to build objects from a mass-and-spring system, you almost immediately run into a fairly fundamental problem. Objects that are made up of groups of springs tend to have problems holding their volume. In fact, with connected springs there really is no volume, so volume needs to be implied by the connectivity of the springs themselves. To make sure the volume of an object is preserved, a large number of extra springs must be created as crossbeams, as you can see in Figure 3.

This is a terribly inefficient way to create solid models. It would make much more sense to begin with a primitive building block that represents a chunk of volume. In continuum mechanics, one method for calculating the deformation of a solid mate-

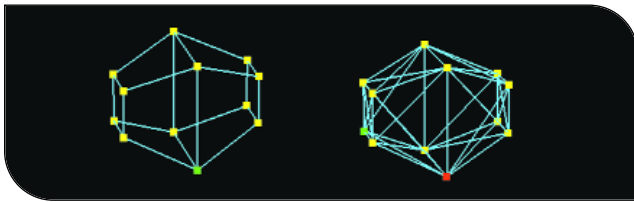


FIGURE 3 Spring objects unsupported (left) and with crossbeam volume supports (right).

rial is to divide the continuous material into a finite number of elements. In this finite-element method, a shape function and the nodes along the boundary define the element structure. Several graphics researchers have explored the use of finite-element methods for computer animation. In particular, the work of James O'Brien and Jessica Hodgins is well suited to the task of interactively deformable objects (see For More Information). Their method made use of tetrahedral finite elements. This choice makes a lot of sense for game developers, as we are quite familiar with creating complex surfaces with triangular meshes. You can see an example tetrahedron in Figure 4.

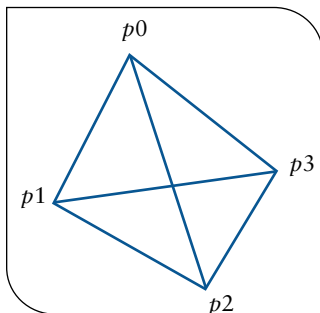


FIGURE 4 Tetrahedrons serve a variety of interactive simulations well with their triangular meshes.

applications, these factors are then combined into a large global stiffness matrix, creating a linear matrix equation. That equation needs to be solved numerically and then the node positions can be updated and the finite-element system advanced. As you can imagine, solving a large linear system of equations can be quite time consuming. In the paper by O'Brien and Hodgins as well as another paper by Gilles Debunne and others (see For More Information), a more efficient technique is used. In this method, each element is solved independently, greatly increasing the speed of simulation.

For the finite-element system, you must take care when establishing the coordinate system. Each node (or vertex) in the finite-element mesh has an initial, undeformed position. Once forces are applied, the object begins to move and the node positions change. However, just because the nodes have moved doesn't mean that the object has deformed.

As an example, think of a single tetrahedral element in space. If we were to apply a simple rotation to the object, the posi-

The volume of a tetrahedron is defined by starting from any vertex, then making vectors from the three edges, a , b , and c , that connect the other vertices to that vertex. The volume is:

$$V = \frac{1}{6} |a \cdot (b \times c)|$$

In mechanical-engineering applications, a finite-element system is solved by computing the stress and strain factors for each node in each element in the entire object. In many

tions of all the nodes would change. However, relative to each other, the nodes have not changed position and the object has not deformed. It is clear that we need a method for computing whether an object has deformed within its own local coordinate system. This is analogous to other general 3D problems such as when we have to transform objects into a different local space in order to perform lighting calculations.

Getting Barycentric

Since an object can deform, coming up with a consistent base frame of reference is difficult. One method is to use barycentric coordinates to establish a consistent frame of reference within each element.

By looking at Figure 5 (on page 34), we can see how this works on a triangle. The center of mass of a body is the point at which it would be balanced under the influence of gravity. This center point is also known as the barycenter. In the case where the mass at each vertex is equal, this point can be calculated a number of ways. If lines are drawn from each vertex to the center of the opposite edge, the point at which the lines intersect is the barycenter, b . (This point is also simply the average of the three vertices.)

The barycenter and the three vertices define a coordinate system for that triangle. In this system, each point on the triangle is defined by three barycentric coordinates. The vertices make up the boundaries of the system: $p0 = (1, 0, 0)$; $p1 = (0, 1, 0)$; $p2 = (0, 0, 1)$. The barycenter of the triangle has the barycentric coordinates $(1/3, 1/3, 1/3)$. The length of the barycentric vector must be 1, meaning the sum of the barycentric coordinates must add up to 1. Given the barycentric coordinates $(b0, b1, b2)$ of a point p , and the three triangle vertices $(p0, p1, p2)$, the position of that point is:

$$p = (b0 * p0, b1 * p1, b2 * p2)$$

Finding the barycentric coordinates for a point is as easy to compute with a few more calculations:

$$denom = (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y)$$

$$b0 = \frac{(p1.x - p.x) * (p2.y - p.y) - (p2.x - p.x) * (p1.y - p.y)}{denom}$$

$$b1 = \frac{(p2.x - p.x) * (p0.y - p.y) - (p0.x - p.x) * (p2.y - p.y)}{denom}$$

$$b2 = \frac{(p0.x - p.x) * (p1.y - p.y) - (p1.x - p.x) * (p0.y - p.y)}{denom}$$

With a coordinate system in place, it can be used to describe the shape of a triangle. If the barycenter is projected onto each edge of the triangle, you will get the points $e0$, $e1$, and $e2$. The barycentric coordinates of these points encode the shape of the triangle. For example, in an equilateral triangle, the points $e0$, $e1$, and $e2$ divide the edges in half. In this case, the barycentric coordinates would be $(0, 0.5, 0.5)$, $(0.5, 0, 0.5)$, and $(0.5, 0.5, 0)$ respectively.

For a triangle in any orientation, the barycentric coordinates of these test points can be calculated. This provides a

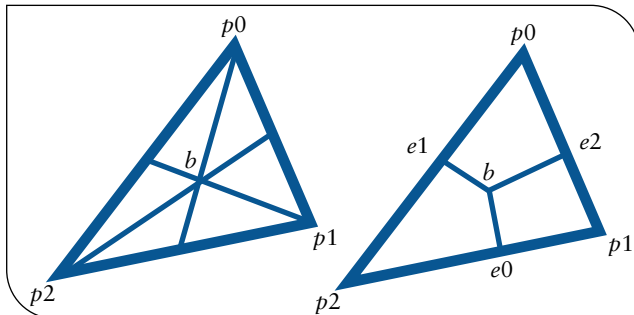


FIGURE 5. Barycentric coordinates. The barycenter is where lines drawn from each vertex to the center of the opposite edge intersect. It is also the average of the three vertices.

method for measuring the shape deformation of a triangle, regardless of its orientation. Obviously, this method only measures the deformation of shape. The size or scale of the triangle can increase as long as the shape is preserved and the barycentric coordinates will stay the same. To counteract that, forces are applied at the vertices drawing toward the barycenter that preserve the area of the triangle. While the method I described here is for a triangle, the same method applies to a tetrahedron. However, in the 3D case with a tetrahedron, there are four barycentric coordinates to describe a point, and the test points are projected onto the face opposite each vertex.

Using this technique, a simple matrix transformation will convert world position back into the base reference frame so the displacement of each node can be determined and the strain value can be calculated. Using the material properties of the object, the strain values of the nodes, and the volume of the tetrahedron, forces are then applied to each element node to attempt to return the object to its rest shape.

Those Numbers Are Lame

The O'Brien and DeBunne papers use a different set of properties to describe the elastic solids rather than the Young's modulus and Poisson's ratio that I described here. To simplify some of the calculations, they use the Lamé coefficients, λ , representing the rigidity of the material, and μ , representing the incompressibility of the material. I have also seen these referred to as the Lamé modulus and shear modulus, respectively. In any case, there is an easy conversion between the Lamé coefficients and the Young's modulus and Poisson's ratio:

$$\lambda = \frac{\nu E}{(1 + \nu)(1 - 2\nu)}$$

$$\mu = \frac{E}{2(1 + \nu)}$$

Since these are both constants for each material, they can be easily looked up from a table of material properties. A nice list of materials can be found at www-mtl.mit.edu/users/lincc/constants.html.

The complete mathematics behind the finite-element calculations get pretty involved, but the concept is fairly easy to grasp. The key concept involves tracking each element as it travels through space and applying forces in that local material frame to maintain the element shape.

You can see my simple finite-element simulator in action in Figure 6. The two bars are meshed identically, the only difference being the material properties. They are anchored at the back, and gravity is being applied across the rest of the object.

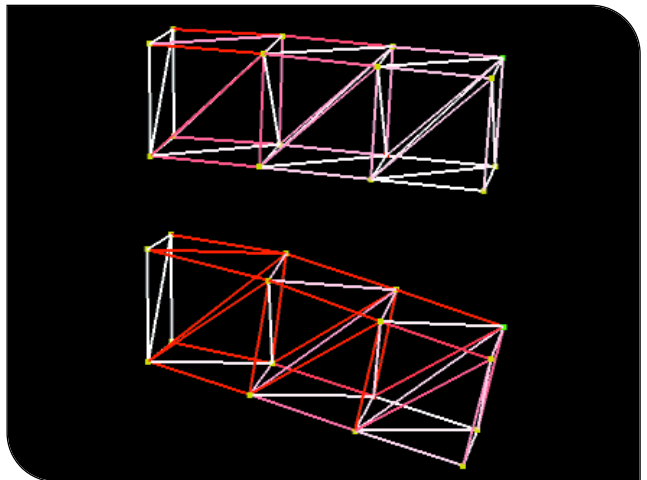


FIGURE 6. A stiff finite-element bar (top) and a flimsy one (bottom), as produced by the author's simulator.

How Do I Use This?

These techniques can be used literally as the building blocks for deformable objects and to exhibit various properties associated with real, flexible objects. For example, you should now be able to create a dynamic rope ladder that a player can climb, which may stretch and break if overloaded.

While finite-element methods look daunting at first, they are actually very similar to mass-and-spring systems. In both cases we are simply applying forces to a system of nodes. Finite elements use a geometric relationship between the nodes to describe the object's shape rather than simply the distance between them.

Even the complete finite-element system I just described is capable of handling large, complex objects with various properties in a much more efficient manner than could be accomplished with a mass-spring system. In particular the system doesn't suffer from the volume preservation problems that are sometimes associated with mass-spring systems.

CEL DAMAGE, from Pseudo Interactive, provides further proof that these techniques can add an interesting element to realism, as well as physical consistency in a variety of new gameplay ideas. This game is the first I know of to create a world where all of the objects in the world are governed by continuum mechanics. The developers used a low-resolution finite-element representation for the physics of the system.



CEL DAMAGE took the concept of continuum mechanics to a cartoonish (and successful) extreme.

Instead of the linear tetrahedral shape function I described here, they used a quadratic basis function. This representation makes it quite easy then to deform the higher-resolution visual models using a technique similar to free-form deformation.

Future Issues

There are some places where this kind of system will not be useful for games. The finite elements as described will always return to their base shape when all external forces are removed. That is to say that the system is designed for totally elastic materials. If you wanted to create a more plastic material, such as a lump of clay, you would need to investigate some other techniques.

In my sample application (available for download from the *Game Developer* web site at www.gdmag.com), it is not currently automatic that you can load in an object and it will generate the finite-element tetrahedral meshing for you. It needs to be done manually in the modeling program or by connecting nodes into elements. This sort of connect-the-dots work is a bit of a pain, so automatic meshing would be a good thing.

There is a lot of information out there on this topic. In fact, meshing is an entire field of research of its own. Many of the methods involve a technique know as Delaunay triangulation. You can find a great starting point at the web site run by Steven Owen called the Meshing Research Center. He has listed links to pages, algorithms, and commercial meshing packages.

I also didn't get into the topic of collision detection between objects. Tetrahedron-to-tetrahedron collision detection is actually pretty easy. Collision response, however, is more difficult. I am currently using simple penalty methods to handle the response.

ACKNOWLEDGEMENTS

Thanks to James O'Brien of the University of California at Berkeley and David Wu of Pseudo Interactive for their input on this article.

However, this method does not handle certain things such as friction very well.

The O'Brien paper goes into great detail about a technique for breaking these finite-element objects into little pieces. While the complete technique is much too computationally expensive for a real-time application at this time, it is a really interesting technique. I have implemented a simple fracture system by just breaking elements at node points. I am currently exploring how to extend this simple system and create a much more realistic effect without any further speed penalty.

I hope this article has gotten you thinking about ways of making our game worlds more physically interactive. By creating a world for the player that is complex and physically self-consistent, we can start to achieve forms of emergent gameplay where the solutions to in-game problems are not restricted by scripted actions. 🎮

FOR MORE INFORMATION

BOOKS

Basics of continuum mechanics:

J. Bonet and R. Wood. *Nonlinear Continuum Mechanics for Finite Element Analysis*. New York: Cambridge Univ. Press, 1997.

GAME DEVELOPER

Dynamic spring behavior:

J. Lander. "Collision Response: Bouncy, Trouncy, Fun." *Graphic Content*, March 1999: pp. 15–19.

PAPERS

Elastic deformation:

G. DeBunne and others. "Dynamic Real-Time Deformations Using Space and Time Adaptive Sampling." In *Computer Graphics (Proceedings of SIGGRAPH 2001)*. August 2001: pp. 31–36.

Finite-element methods for object deformation:

J. O'Brien and J. Hodgins. "Graphical Modeling and Animation of Brittle Fracture." In *Computer Graphics (Proceedings of SIGGRAPH 99)*. August 1999: pp. 111–120.

www.cs.berkeley.edu/~job/papers/obrien-1999-GBA.pdf

The physics of CEL DAMAGE:

D. Wu. "The Physics That Brought CEL DAMAGE to Life: A Case Study." *Game Developers Conference 2002 Proceedings*.

See www.pseudointeractive.com/about.shtml

WEB SITES

PONTIFEX, sequel to Alex Austin's BRIDGE BUILDER

www.chroniclogic.com

Barycentric coordinates

<http://mathworld.wolfram.com/BarycentricCoordinates.html>

Conversion factors, material properties, and constants

[www-mtl.mit.edu/users/lincc/constants.html](http://www.mit.mit.edu/users/lincc/constants.html)

Steven Owen's Meshing Research Center

www.andrew.cmu.edu/user/sowen/mesh.html

Game Design Lessons from Real Life: Game Object Interactions

There are a lot of silly little things about the game industry that I love, and a lot of that I could do without. One of the things I love is that whether it's game design or a piece of code, there's always something new to learn about the business, even from seemingly unlikely sources.

A lot of what I have learned about the game industry in my lifetime hasn't come from the trenches; it's come from the part of my life that's not involved in games. Through observation, movies, comics, and experiences interacting with people, I've learned more about creating an interactive world than I ever have from actually playing one.

If what follows in this article appears to be non-game related at first, stick with it; it may be entertaining at the very least, and hopefully educational for you if you've had your morning cup of coffee. Like many things in life, there's something relevant to making games somewhere in there.

One big challenge developers face in our medium is that game design is one of those gray areas that's extremely difficult to teach in school (and up until very recently no one taught it at all). To complicate matters further, there are many different approaches to game design. While a simple game such as *SPACE INVADERS* doesn't need much more than a page to design, large modern games need a great deal of thought before the first line of code is written. Some games have many contributors to the overall design, while others divide resources and conquer. Of course, there's no single correct way to design a game, it's as varied in approach as authoring a book, planning a revolution, or making a documentary on beetles.

Despite the challenges game designers face in figuring out exactly what it is they are doing, there are some universal foundation pebbles that designers of all large, complex games can identify and use. I've got a few of these pebbles right here in a bucket, so let's reach in and look at one.

Starbucks

I go to Starbucks a lot. There are two locations I frequent daily: one near my house, the other near the office. I know

almost everyone there by name, and they regret knowing me. The Starbucks near my house is also near a bunch of other houses, so it's full of teenagers, real estate brokers, and so forth, while the one at the office gets everyone from Sears employees to truck drivers to office folk.

More than a few times a week I sit around at Starbucks and watch life go by. Really, there is nothing better than a Starbucks for watching the patterns of life reveal themselves to you. There are patterns both in the macroscopic day of the week and in the microscopic time of day. Some are quite subtle, but you can observe all of them in action with a comfortable seat and a hot cup of coffee. It's chaos theory on caffeine. My interest in observing these patterns is furthered by the fact that the two locations each has its own distinct patterns.

Now, I could have used any location for such important game design research, I suppose, but Starbucks is good for a few reasons. First, it doesn't serve alcohol, so my memory of what happens there isn't impaired as the time passes, and I can generally stay focused. (To that end, I valiantly tried several other locations that served Guinness, none of which worked out well for game design research.) Second, Starbucks has very comfortable seating. Comfortable seating is extremely important in any long-term experiment. And finally, they have wireless Internet access. O.K., perhaps that should have been higher on the list.

Most weekdays I go see what's going on at either location first thing in the morning. Generally there's a line to get caffeine, a few people sitting down, and a lot of traffic going through the store. Now is not the time to chat to the staff, and if you persist, your chances of getting a free coffee later in the day degrade rapidly. The smile they have for you is the smile they have for everyone. You say hello to a few regulars who appear there at the same time as you every day, and you move on with your day.

Weekend mornings are different. I go up there with my daughter and we pass through. On these days I am not so much me as I am simply the man who accompanies his daughter. Some regulars hang out there on weekend mornings, lingering longer, perhaps with a paper or a friend, and several "week-enders" hog the comfortable chairs. There are unwritten meeting times every weekend for casual groups.

How does any of this apply to computer games? Well, I think

GRAEME J. DEVINE | *Graeme is currently programming for DOOM III, in production at id Software.*

NOVA POUNDS
COFFEE \$2
CAPPUCCINO \$3.50
LATTE \$3.00
ALE - 3 GROSS
MEAD - 2 GROSS
HOCK - 4 GROSS



Illustration - Kevin McSherry

of everything in a game as an object, be it a soldier, a tavern, an airplane, or a villain. They are all objects. Most of the time we forge ahead with these objects and, in our eagerness to get on with populating a game world, add them into our game environment without much thought.

This brings us to the second part of a game, the environment, within which the objects interact: deep space, open fields, and underwater caverns are kinds of environments. Environments provide the binding fabric for the objects — think of it as the code that provides the laws of nature. Physics, lighting, time, and particles would all be coded as part of the environment. The environment is the conduit that passes the outputs of the various game objects to the inputs of others, ensuring that the objects exist, look, and feel right to the player.

Objects

Because the majority of player time is spent interacting with objects or causing objects to interact with each other in the environment, designers cannot afford to dump them into the game world with giddy abandon. Objects serve as shells for the components of a game. They have defined input parameters, and based on that input they affect the environment and perhaps other objects within the game system. Let's look at a well-defined object and something poorly defined.

Say we're making a massively multiplayer online game and we need a tavern. We'll call our tavern "Novapounds." Novapounds serves ale, accepts gold, and dispenses ale that gives you some small amount of health, but at the expense of some dexterity. Sounds nifty, doesn't it? Because of its extreme popularity in this online world, Novapounds taverns open up all over the place.

So far with this game object, we have gold coming in and some not necessarily tangible benefits coming out. However, it's a good place to meet people; you can load up on six-packs of ale to go, and you're set for a fine evening killing monsters.

Now, to vastly oversimplify a game system here for the purpose of an example, what happens to all the gold?

A well-designed MMOG creates an economy that cycles what it has and allows for gold to flow back into the game system. But in our case with Novapounds, we just zap the gold into the ether and it's gone from the world forever. Pretty soon there's no gold, and you're left inserting new gold into an econ-

omy that is never going to balance.

A simple solution might be to have a well-armored wagon stop by several times a day, pick up the gold, and carry it off to a bank. But all we've done here is move the problem to another building, albeit an aptly named one. How do we cycle the gold back into the game system?

There are several design opportunities here to explore. First off, we could build the concept of robbery into the tavern, the wagon, and the bank. Each of these has increasing risk/reward scenarios that seem well suited toward a game system. Second, we could also use a "game tax" to pay for the various objects we have in and around the environment doing bidding on behalf of the game. For example, 10 percent of what you take from monsters and looting is paid to the city upon entering its gates. (Someone's got to pay to keep the golf courses green.)

Third, we could make the owner of Novapounds the hooded villain behind the curtain, which would be a poor choice, since now you're tying the design of an object to some game story element.

These opportunities are fraught with problems themselves, but they illustrate how a simple object, badly designed, can corrupt the complete game.

Objects need a diverse set of input parameters that make sense in order for that

object to work well.

When Good Objects Go Bad

A well-defined object goes on to provide additional interest to the world it serves. Just as my local Starbucks is observed to change given the store's location and time of day, each object in a game should change based on its location and time within the game system. And as an object changes during the course of a game, it is important that it continue to behave correctly at all times.

What exactly does that mean? A well-defined object, even something as complicated to define as a tavern, needs to operate as a good citizen in the environment at all times. If there's a morning rush, an afternoon lull, or a sudden urge among all the



other game objects to hang out around it, the tavern object needs to continue operating per its set of rules, so it can present good information to the player in the spirit of the game. These other game objects know nothing about being a tavern — and shouldn't — but the tavern should provide input to them that allows them to behave appropriately, just as we receive input upon entering a coffee shop that tells us what is going on there and how we should behave.

The best example of a game whose objects behave well together may be the original *SIMCITY*. There are so few objects in that game but so, so many possibilities. Many an evening I would spend carefully placing industry blocks far away from my commercial and residential blocks, always trying to create Utopia but inevitably ending up with Detroit. On the flip side, a game that initially presented many balance issues was *ULTIMA ONLINE*. The bunnies outside the cities were hunted to extinction, bringing down fiercer animals looking for food, which made it impossible for newbies to leave the city walls lest they become an easy meal.

Another example of poorly thought out object interaction can be found in action games, which often do not allow for player death. A player can be armed to the teeth, get beaten by some bad guy, and then come back to the same spot with nothing more than a large stick. Thus players are unable to fend off an attack from any kind of enemy around their current position. While a large stick is sufficient defense against bunnies, it doesn't serve you so well when the other guy has a rocket launcher.

Those are perhaps extreme examples, but almost every game has something in it to illustrate some kind of breakdown in object interaction. I worked on an RPG once that, in theory, had you spend many hours gathering a party to undertake a great quest fraught with danger and peril. Right before the game went gold, we noticed that if the player character made a beeline for the end of the game, skipping the middle, it was possible to complete the game in a matter of minutes. No rules were broken, but it was obvious that we had taken a singular avenue of thought making the game, following the written story as we tested rather than exploring everything. Because of our slavish devotion to our story as written, we had ignored the obvious path.

Circumstances such as the examples I've just described might seem like egregiously obvious design flaws, but the fact is I've read a lot of game designs that fail to think through and document how all the objects will interact with each other in the environment. Such a lapse in object design can end up in a great-looking but direly flawed beta that leads to late changes — changes that try to fix these flaws but never really do, because it's too late in the process. Very little can be done at that point to prevent late-emerging flaws from becoming flaws that ship with the game.

How can you as a designer avoid flawed object interactions? Thinking of all the possibilities while standing in the shower, driving your car, and dreaming are preferred methods (remember, switch the shower on) among game designers to address

this and many other game design problems. Such techniques go to show that effective tools for game design are still few and far between, so you will have to rely on classic tools such as Word and Excel to outline the ins and outs of various game objects in the world, show their relationships, and provide the day-to-day roadmap for project development. The act of actually creating these object profiles often sheds most of the light, and should help you avoid problems down the road.

Documenting Object Interactions

Let's think about how we would communicate an object's parameters in a game design document. If we were to spend a few words on our Novapounds tavern in a game

Name: Novapounds

Type: A Tavern Building

Use: Spending gold here will give you health at the cost of some dexterity. For every one gold coin spent here, you gain four points of health and lose half a point of dexterity. You can only go four points past your maximum health; the dexterity loss is temporary and lasts for 10 game minutes.

Notes: Novapounds was founded in EA 35 by two small, furry creatures that happened to have a love for smol-grass, the base of the warm liquid that tastes not totally unlike tea, sold here. They ran the business successfully until EA 42, whereupon lawyers finally caught up to them, nailed them to the wall, and bound them with so much red tape that they were never heard from again. The business fell into the hands of the state and has been a source of profit for the inner worlds ever since. The funds collected at each Novapounds are used in various non-profit rehabilitation programs for the local criminal population. Unfortunately, this has proved to be a complete and utter failure, but since no one has a better idea, the practice continues.

Bottom Line: Money comes in here, and goes out to rehabilitate bad guys, making the loop complete and the need to just make money appear out of thin air go away.

design, it might go something like this.

This example object description not only gives a brief overview of the object with brief headings describing its name and use, but more importantly it describes how the object interacts with the world. It would be one of many such objects in a game design, each with a brief overview describing how it fits into the world, before a much more detailed overview later in the document. If you think through all your characters, functional places, vehicles, and so forth in this manner and give them a way to fit into the environment naturally, you'll have a good, concrete basis from which to move forward. This basis



Origin's *ULTIMA ONLINE* (left) and Maxis' *SIMCITY* (right) are both games that allow plenty of interaction between different objects within the game.

won't connect all the pieces together, but as you complete, polish, and finish the design off, it should provide a logical rule set for your game world and characters.

Before you even get to the level of detail present in these notes, there should already be a very broad game overview that fits all the pieces together in the grand scheme of things. If we were to insert Novapounds into such an overview, it might go

GREAT GAME

GREAT GAME is a massively multiplayer online game; it's set in the future, which is surprisingly full of goblins, elves, and furry creatures large and small. [Go on to describe universe.]

GREAT GAME revolves around a "living economy." That is to say, there's a strong attempt with the game system to cycle everything in the world, mirroring the way a real-world economy works. There could be times of growth, spurring grand quests and adventures, and times of despair, whereupon a large percentage of the players may turn to crime as a way to stay alive.

[Go on to describe role of players.]

In each of the towns, we've established several franchises that can equip players with clothing, weapons, the finest footwear this side of Proctus Glandus, information, and health. These gathering places provide some of the core mechanics to the living economy, as gold is cycled from the monsters, to the players, and back to the monsters.

something like this:

And so we're on our way to a good game design document. There's much more to a game design document than just these components, of course, but there's no doubt that the up-front thought and work done here on how game objects interact together can break or make a game.

Not every kind of game requires such in-depth documentation, and indeed there is a growing antidocumentation sentiment among some designers and producers out there for certain kinds of games. However, large games that represent a vast, complex environment with many interactions between the game objects absolutely need to be documented. Without sufficient documentation for game objects in particular, your project will run late, be open to mistakes, and lack specific goals that allow the game to finish. Mark my words.

As I sit here in Starbucks wrapping this up, there are people chatting, others are reading books, and perhaps a business deal or two is in the works. It's all going along well, cycling the world, behaving properly within its parameters. Starbucks has taught me quite a bit about game design so far, and surely there's more to learn, perhaps even another pebble or two.



My Friend, the Covariance Body

Game worlds are becoming increasingly complicated. As programmers, we write rules that determine how the game world behaves. Those rules need to produce increasingly complex results as output; as for input, they need to deal with larger volumes of world state, and the world state describing any particular entity is becoming ever more intricate.

As a result, unsurprisingly, games are becoming harder to create. One way we might cope with this reality is simply by becoming better, more experienced programmers. But there's a limit to how far we can push before we snap or something, so we need other options. One option is to use ever-larger teams of programmers to create games, but it's clear from experience that that doesn't work very well.

So what can we do? We can adopt more powerful methodologies; this has benefited us greatly in the past, as we program in C++ now instead of assembly language, and we're much better for it. Unfortunately, new methods and concepts don't come along very often. So we need to actively seek them out, or invent them.

Statistics

Our game worlds contain huge amounts of information, and we need to process all of it to produce meaningful results. I propose turning to the well-studied field of statistics to find ideas that can help us here. One of the purposes of statistics is that you can summarize large amounts of data in useful ways, and you can perform well-defined mathematical operations on that summarized data.

Statistical methods can be extremely robust in the presence of noise or incorrect inputs, where discrete algorithms would fail outright. This is important because rigid algorithmic failures are a big

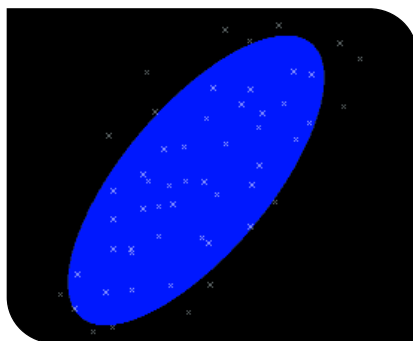


FIGURE 1. A set of data points and an ellipse that summarizes them. The ellipse is drawn at 1.7 standard deviations.

source of headaches in modern games. Stochastic algorithms can provide a terrific speed boost as well; for example, Monte Carlo Ray Tracing (MCRT) and its algorithmic descendants are widely believed to be the best methods of high-end scene rendering. But game programmers' statistical tool-sets tend to be very small, so I'd like to introduce a tool that I call the "covariance body."

You can find brief presentations on the idea of covariance on various web sites or in statistics or physics books. That material is often unfocused, difficult to wade through, or just hidden in a place where most game developers won't come across it. The purpose of this article is to present covariance bodies in a utilitarian way.

Two months ago I made a passing reference to covariance matrices, in my column about transmitting vectors over a network ("Transmitting Vectors," July 2002). I discussed the fitness of squares versus hexagons for tiling the plane and said that covariance matrices can tell you

their relative compactness. This month I'll show how.

"Covariance Body"

Covariance body is a nonstandard term that I chose to represent a certain set of values that we can manipulate as a coherent entity. A covariance body can summarize a set of vectors in any number of dimensions; for simplicity, we'll restrict ourselves to 2D here, but the generalization is straightforward.

The quantities that make up a 2D covariance body are a 2×2 matrix, representing the covariance of the vectors; a two-vector, the mean of the input vectors; and a scaling factor that I will call "mass," which is the sum of the masses of the input vectors. In the simplest case, the input masses will be one, but it's often useful to weight the contribution of various vectors (for example, weighting them according to how confident you are that they belong in this particular set).

People who have done a lot of physics programming will find covariance bodies extremely familiar; they consist of the same stuff that you work with when rotating a physical body in space. This correspondence to physical bodies is nice, because when you think about what will happen when you manipulate these things, your intuition will usually be right. Mathematically, these quantities together represent an ellipse positioned in space, as Figure 1 shows.

The covariance matrix determines the shape of the ellipse. So what is "covariance," exactly? In games we often deal with scalar time-series data, and we know



JONATHAN BLOW | Jonathan is a thinking of leaving the USA. Where should he live instead? He would prefer someplace that has no software patents and doesn't say things like "axis of evil" when explaining its foreign policy. Send e-mail to jon@number-one.com and let him know.

that the variance measures how much the input value tends to change from sample to sample. But for today's purposes, we want to visualize the distribution of input values and think of the variance as characterizing the approximate width of the distribution of input values (Figure 2).

When you have multidimensional inputs, you can compute the variance for each dimension separately. But you can also compute the correlation between the dimensions, and that is what I mean by covariance. In the literature you will also see the term "variance-covariance matrix," which for the purposes of this article means the same thing as "covariance matrix."

Computing a Covariance Body

So, you've got a bunch of input points. These could be positions in the game world, pixels in a bitmap, or whatever. Each point has a certain mass associated with it, representing its relative importance. The total mass of all the points is $m_{\text{total}} = \sum m_i$, where m_i is the mass of an individual input. The weighted mean of these points, analogous to their center of mass in the physical world, is: $v_{\text{center}} = m_{\text{total}}^{-1} \sum m_i v_i$, where v_i is the i th position.

Let v'_i be the position of a point relative to its center of mass, that is,

$$v'_i = v_i - v_{\text{center}}$$

Then the covariance matrix, which describes how mass is distributed in space around the center, is the outer product: $m_{\text{total}}^{-1} \sum m_i v'_i v'^t_i$. Expanding this product for the 2D case gives us the 2×2 symmetric matrix:

$$m_{\text{total}}^{-1} \sum \begin{bmatrix} x_i^2 & x_i y_i \\ x_i y_i & y_i^2 \end{bmatrix}$$

where $v_i = (x_i, y_i)$.

The covariance body consists of these quantities, the $n \times n$ symmetric covariance matrix C , the $n \times 1$ vector for the center of mass x , and the scalar total mass m . Occasionally I will refer to a particular covariance body using the notation $\{C, x, m\}$.

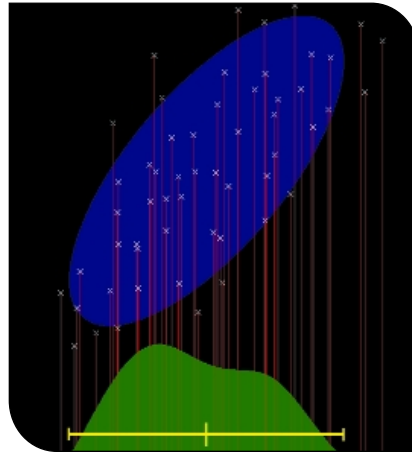


FIGURE 2. We consider the one-dimensional statistics of the data points by projecting them down to the X-axis (projection shown by red vertical lines).

Visualization

The covariance matrix C tells us the variance of our mass as a function of direction. That is not very much information compared to the large amount of data with which we started, but it can serve as a tidy summary. We can take a simple shape that has the same variance as our input data and visualize that. Another way to think of this is that by computing the variance of our input mass, we have least-squares fitted this simple shape to the input.

The Gaussian curve is one of the most natural shapes for talking about the distribution of mass. The Fuzzy Central Limit Theorem states that when you sum together enough arbitrary random processes, the result is a "normal distribution" (a Gaussian; see For More Information). Since most sets of data result from the interaction of a large number of random-seeming processes, the Gaussian is a good way to describe them.

The 2D version of a Gaussian is known as a Gaussian bivariate distribution; it has two variances, a maximum and minimum. The level sets of this Gaussian — points for which the function has a constant value — are ellipses. One way to visualize this Gaussian in 2D is to draw a charac-

teristic ellipse.

The eigenvalues of a covariance matrix tell you the maximum and minimum variance of its Gaussian as a function of direction; you square-root the eigenvalues to get the length of each axis of the ellipse. This is because variance is computed in squared units, and you want lengths in the native units of your input. Thus, you want the standard deviation, which is just the square root of the variance. The square root operation won't cause any problems, since the eigenvalues are guaranteed to be nonnegative (though they might be zero).

The eigenvectors of the covariance matrix tell you the direction in which each eigenvalue is achieved; this provides the ellipse's orientation. The mean tells you where the ellipse should be positioned. That's all the information you need to draw the ellipse on the screen.

Manipulation

One nice property of covariance bodies is that you can quickly combine two bodies A and B. The body you get as a result is the same thing you would have computed if you had reiterated over all the input masses comprising A and B. You can speculatively combine covariance bodies in hierarchies and speedily diagnose the resulting shape. This was very useful to me in a recent image-processing application, which required churning through a large number of shape combinations.

Suppose you wish to add two covariance bodies, $\{C_1, x_1, m_1\}$ and $\{C_2, x_2, m_2\}$. Intuitively, the mass of the resulting body is $m_3 = m_1 + m_2$. It's also not difficult to figure out that the new mean is a weighted average of the old means, depending on each body's mass: $x_3 = (m_1 x_1 + m_2 x_2) / (m_1 + m_2)$. To compute C_3 , we need to shift C_1 and C_2 so that they represent each body's mass distribution with respect to the new mean. Then we can add the matrices together, and that gives us C_3 .

To see how to shift a covariance matrix to a new reference point, expand the expression $m_{\text{total}}^{-1} \sum m_i (x_i + s)(x_i + s)^t$ where s is the vector from the old refer-

ence point to the new one. Expanding this product gives you: $C + sx^t + xs^t + ss^t$, where x was your old reference point (until now, the center of mass).

Other linear operations can be summed with similar ease. In my image-processing application, I wanted to least-squares fit a set of planes to the RGB values that comprised each covariance body so that I could later reconstruct a linear approximation to the color over the whole region. Interestingly, the math for this least-squares fit used the same covariance matrix that I was already computing; it just used it in a slightly different way, where I would solve an $Ax = b$ problem to get the results, instead of an eigenvector problem. So the only additional information I needed to keep around was the b vector for the right-hand side of this equation. When summing covariance bodies, I would just compute $b_3 = b_1 + b_2$, and suddenly I had an aggregate least-squares color fit over the summed body. It was cool.

Compactness

In my “Transmitting Vectors” column, I talked about using the hexagon versus the square to tile the plane, in order to

transmit the position of an entity over the network. I said that the hexagon was a better choice, because it is more compact than the square.

Covariance provides one method for finding the compactness of a shape. In 1D, given two distributions of the same amount of mass, the distribution with the smaller standard deviation is more compact. The situation is more confusing in 2D, where a distribution can be compact in one dimension but widespread in the other. I tend to measure compactness using the average standard deviation with respect to direction, which you can find by integrating over one quarter of the ellipse:

$$\frac{2}{\pi} \int_0^{\frac{\pi}{2}} (a \cos \theta + b \sin \theta) d\theta$$

This works out to $(a + b) 2/\pi$, where a and b are the lengths of the axes. If we only care about relative compactness, we can throw out the scaling factor and just use $(a + b)$.


If we’re dealing with shapes of constant density and area, like the square versus the hexagon, a circle represents the most compact possible shape. This is verified by our compactness measure: if $\text{Area} = 2\pi ab$, then $ab = k = \text{Area} / 2\pi$, thus $b = k/a$; if you allow a to vary and you minimize $(a + b)$, the answer you get is

$$a = b = \sqrt{k}$$

which is a circle.

So to find the compactness of a 2D shape, we just need its covariance. You can compute the covariance for a 2D shape in closed form; just integrate the covariance matrix over each point in the shape. You can generically handle polygons without writing a specific equation for each one; all you need is the closed-form equation for the covariance of a triangle (an exercise left to the interested reader). Then, to find the covariance of a polygon, you just triangulate it, compute the covariance body for each triangle, and combine those bodies.

Sample Code, and Next Month

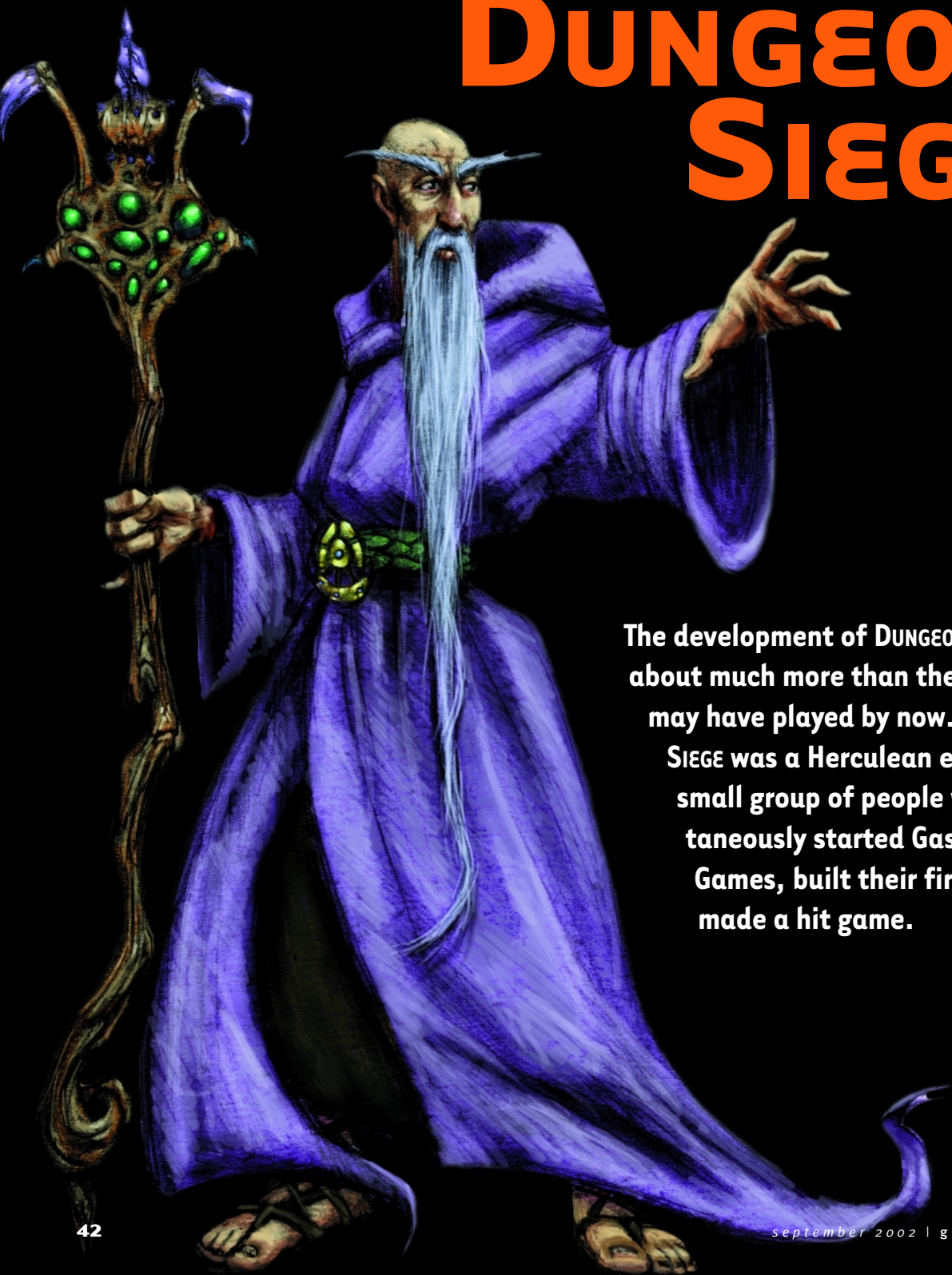
This month’s sample code illustrates the construction and combination of covariance bodies to summarize data. You can input a bunch of masses by clicking on the screen, and the program draws the resulting ellipse. So far this has all been a little bit divorced from the day-to-day concerns of game programming, but I’ll apply covariance to a concrete and relevant topic next month, when I’ll look at scripting languages. Until then, you can download this month’s sample code from the *Game Developer* web site at www.gdmag.com. 

FOR MORE INFORMATION

Central Limit Theorem

<http://mathworld.wolfram.com/CentralLimitTheorem.html>

Gas Powered Games' DUNGEON SIEGE



The development of **DUNGEON SIEGE** was about much more than the RPG you may have played by now. **DUNGEON SIEGE** was a Herculean effort by a small group of people who simultaneously started Gas Powered Games, built their first RPG, and made a hit game.

Prior to his career as a prophet, Chris Taylor, Gas Powered Games' president and founder, was responsible for the creation of

both the hit RTS game *TOTAL ANNIHILATION*, as well as Cavedog Entertainment, from which it came. For reasons that are now obscured by the mists of time, Chris was inspired to try his hand at something new, and Gas Powered Games was born. A number of us who worked with Chris in the past, primarily on *TOTAL ANNIHILATION*, had similar passions, and it wasn't long before we found ourselves cozy once again in a closet office in Kirkland, Wash. We had just finished work on a successful RTS game, a genre we loved working on, but for a number of reasons we opted to try a different genre. We knew we were skilled enough to make another RTS, but the world was swimming in them at that time, so it seemed like a good opportunity to try something new. We all liked the fantasy aesthetic and there were very few good RPG games at that time, so somehow doing an RPG seemed like the natural choice.

We were starting completely from scratch. We had to find an office and buy phones, fax machines, desks, and all those little things most of us take for granted working at an established company. Our first engineering meeting was held at Chris's house with a whiteboard on the floor and a half dozen of us huddled around it. A month later, when we finally did find a space, we had to make it usable. I remember Chris crimping network cables and climbing up on ladders through ceiling tiles running the network wire in the new office. I also clearly remember my first day in that office, when I built my own PC from random parts we bought, unfolding a cheap Costco table on which to set it all up. I loaded up MS Dev and faced an empty header file, which at that moment repre-

sented our entire technology base.

Chris repeatedly told us we were in for a wild ride, but I don't think that prophecy sank in until years later. So merrily and naively, we set out on what turned out to be a four-year journey that was as challenging as it was educational.

What Went Right

1. Exceptional team. It's clear to me that the single best thing about *DUNGEON SIEGE* is the team that created it. So many things are right about our team that it's difficult to elaborate on this without sounding as if I'm gushing. From the beginning, Chris Taylor set the tone for the company with strong values and an outrageous personality. Chris might be best described as a force of nature; he can help an old lady cross the street, tell you a joke that will make you seek psychological counseling, make a shrewd business transaction, and convince you that an impossible feature is actually easy, all in the span of about five minutes. He is certainly the hardest-working and most driven person any of us know. He instills the kind of respect that is pivotal in binding a team into a single, coherent unit.

The team itself is consistently calm in the face of pressure, persistent in the face of adversity, and absolutely dedicated to success. I have never worked in a group of so many selfless, dedicated, and positive people. At times we can be so focused and single-minded that by some definitions we might be considered a cult.

Another positive trait that contributed to the team's efficacy was that we shunned any kind of classism. At GPG, we have a culture where everyone is valued. We don't single out contractors or junior people as different, everyone's ideas get heard, and it's implicitly understood that every single person plays an important role in creating the game.



GAME STATS

PUBLISHER: Microsoft
NUMBER OF FULL-TIME DEVELOPERS: 27 at ship date
NUMBER OF CONTRACTORS: 5
LENGTH OF DEVELOPMENT: 3 years, 8 months
RELEASE DATE: April 5, 2002
PLATFORM: PC
DEVELOPMENT SOFTWARE USED: MS Dev C++, 3DS Max with Character Studio, Visual SourceSafe, CodeWright, ICQ, RAID (bug tracking), Photoshop, Excel
DEVELOPMENT HARDWARE USED: Ranged over course of development from 400–1000MHz CPUs with 128–512MB RAM
NOTABLE TECHNOLOGIES: Bink, Miles, SmartHeap
PROJECT SIZE: Approximately 800,000 lines of source code for game, editor, and associated tools; 60,000 lines of scripts; 21 million total lines of .GAS configuration files; 8,500 textures, 2,000 animations, 2,600 object and actor meshes, 3,700 terrain meshes

2. Exceptional art. Somewhere between the prototype and our first E3, something magical happened. It may have been when our technology started to hobble along such that people could finally see what the game might look like, or maybe it was when our tools became really usable. For whatever reason, since our first E3 and for about three years thereafter, the game just kept looking better every day. Art director

BARTOSZ KIJANKA | Bartosz served as the tech lead for *DUNGEON SIEGE* while also acting as VP of technology at Gas Powered Games. Feedback on this article is welcome at bkijanka@gaspowered.com.

Steve thompson and his team straddled an exponential curve of outdoing themselves and rode it to the very end. Over time the models morphed from “Hey, is that a bear or a giant rat?” to “Wow, let’s see that again!”

Similarly, the level designers did a remarkable job. As the editor settled, their productivity skyrocketed. Near the end of production the volume of their output was stupefying. I’m still shocked when I think of our multiplayer world, which is far larger than the already large single-player world, and how it was built in a tenth of the time. Amazingly, as the level design team picked up speed, the quality of the output and attention to detail also improved.

From very early in the project we had a flexible effects system that was only marginally used because we were short of people, so not many assets took advantage of it. Late into the game’s development, the art was already quite far along when we had an additional and surprising growth spurt. Eric Tams, our overworked content engineer, went ballistic and added so many special-effects embellishments that we were, quite frankly, astonished. Suddenly, the effects system was working overtime as swords were flaming, staffs were sparking, and so many other things we don’t have names for were happening. It was a nice surprise, and one example among many of team members working with inspiration to make a difference. This sort of inspiration really helped the game evolve on all fronts.

3. Extreme flexibility. As a small company our flexibility is a distinctive advantage over a large company. Compared to what I’ve seen of many large companies, we can make important decisions 100 times faster. If you’re trying to innovate, the ability to make decisions quickly is absolutely critical. At a large company, I’ve seen people discuss a minor issue for days on end. At GPG, if we need a resource, or if we’ve identified a project course correction, we meet with Chris Taylor and most of the time we will decide upon and commit to a specific course of action in under five minutes — very refreshing.



Initial sketches of characters and creatures that inhabit the world of DUNGEON SIEGE (from top to bottom): Gargoyle, Droog, and Krug.

One of the more extreme examples of this sort of flexibility was our temporary test team. Deep into the debugging stage of the project, we were experiencing increasing frustration with the process. Up to this point, Microsoft, our publisher, was doing most of the testing, and although they were on-site for some time, they had moved back to the mother ship, complicating matters. Try as they might, the scope of the game was simply enormous. We needed more help. We realized this on a Friday, and by next Monday morning we had configured our own test lab. By the end of the week the lab was fully staffed and running double shifts.

We could be this flexible as a company only because our teammates were this flexible. Everyone carried multiple responsibilities, and some of us carried so many it was hard to keep track. As one example, Jacob McMahon wore the hats of VP, designer, producer, HR, accounting, bill paying, payroll, legal, and who knows what else. He also managed to place all the monsters in the game and gameplay-balance the entire thing, while only occasionally speaking in tongues. Such effort is representative of what we had to do in order to succeed.

4. Solid architecture. The engine underwent numerous architectural changes during development, and ultimately we were left with an engine that’s both powerful and flexible. Because development included a lot of research, the architecture was forced to evolve through repeated reengineering to support new discoveries as well as new requirements.

Based on our experiences during the development of TOTAL ANNIHILATION, we knew even before we wrote the first line of code that we had to focus on a data-driven design. So although much changed about the architecture over time, the critical concept and goal of a data-driven engine persisted. Today it seems more fashionable to take this approach, but four years ago there didn’t seem to be much focus on this issue. At that time, many game engines would

inevitably stumble onto being data-driven, but it seems that few game engines were doing this in a premeditated and planned fashion.

Our goal of data-driving the DUNGEON SIEGE engine was to support all the features of an action RPG but not hard-code how these features interact in order to create the look and feel of the game. Aside from the actual art assets, the look and feel of the game is defined by a combination of configuration, text files, and scripts. We used a general-purpose scripting language for AI, animation control, and spells, and a specialized language for scripting special effects.

The scripting system, Skrit, played a key role in the architectural success of DUNGEON SIEGE. Written by Scott Bilas, it is implemented unlike any other scripting system we've seen to date. The language itself is conventional, but extending it by exporting engine functionality is easier than ever. Exporting a function is as simple as adding a single tag in the C++ code. Additionally, because Scott had to create a powerful back end to support Skrit, it became the basis for a remote procedure call (RPC) mechanism that made implementing multiplayer much easier. We leveraged the same technology for save and load and for many other systems.

Based on another early architecture decision, the editor was essentially built on top of the game. Specifically, it was built on top of what we called the "world" layer of the game. The world contains all of the game mechanics, less the user interface. The world layer has no knowledge of a user interface, or even high-level gameplay goals such as quests. It's simply the game environment containing the map and all its life forms. The actual game executable is built by using the world as a foundation and adding a game component above it. The game component gives you the user interface, creates your hero, and has all the other trappings required to present the gaming experience to the user.

Alternatively, the editor is simply the world layer with an editing component on top instead of a gaming component.



Early explorers Fedwyr and Klars discovered much of the route traveled in DUNGEON SIEGE.

The editor component allows you to create and manipulate the world, whereas the gaming component restricts you to a particular experiencing of it. The benefit of this approach was that the editor took advantage of much of the work we did for the game engine. The drawback was that a careless change in the world layer for the sake of the game component could break the editor component. Chad Queen, who built the editor, was relentless in keeping the editor running such that this vulnerability wasn't ever much of a problem.

5 • Instant messaging. We faced the typical set of communication challenges that plague most development teams. However, we had at least one success worth mentioning.

Consider the typical means of communication in the office: Normally when you urgently want to speak to someone, you call them. At other times, when you need a more formal and structured medium for communication, you e-mail someone. And sometimes, when you're sitting next to someone and just want to ask a quick informal question, you look over your shoulder to see whether they're in a good place to be interrupted.

In our first year, as a small group of

fewer than a dozen people, we worked on the same floor and essentially in the same space. Because of our proximity, all of our informal communication happened in person. (It's amazing how much important work is facilitated through informal communication.) But when we started to grow we found ourselves in a communication conundrum. When engineering moved to a different floor of the building, that informal channel of communication with the other groups in the company was lost.

After a brief period of disorientation we identified the problem and called upon ICQ instant messaging to the rescue. It's a powerful tool and at times can actually be more effective than informally speaking to someone. Because ICQ is so informal, you can be very direct without being perceived as terse. You simply type in your question and you're done. No chitchat or small talk needed, no pressure to introduce yourself or your topic gracefully.

Instant messaging works very well in conjunction with formal e-mails and semiformal phone calls. It allows us to direct without offending anyone and is more efficient than waiting in line to speak to someone. If it's not a good time, the recipient of your message can simply



A panorama of Castle Ehb.

ignore your request until his or her next convenience. Instant messaging had such a positive impact on our work that I can't imagine ever working without it.

What Went Wrong

1. Extreme ambition. Extreme ambition can be empowering, but it can also bring much pain. A large portion of Gas Powered Games consists of extraordinarily ambitious people. Everyone has their own reasons, but collectively, we all want to win, and want to win badly. While this drive helped us persevere through some of the challenges we encountered during development, it also created some new ones.

Our ambition translated into a number of painful symptoms such as feature creep, over-optimism, and a project scope that was ultimately larger than our ability. The problem was compounded by the fact that nobody on the team had really worked on an action RPG before. We looked at the "leading brand" RPG and said to ourselves, "Hey, this is a fun genre. Let's make something like this except, much, much bigger, in 3D, without loading screens, and push the technology to its absolute breaking point!"

Clearly we couldn't have imagined the sheer amount of work required to build the kind of RPG that we wanted. And we wanted all kinds of interesting things: lip-synching, cooperative networked level editing, dual monitor support, wavelet terrain compression, deformable terrain, and many other gems. Some of these features ultimately didn't fit into the vision of our game, while others were simply extraneous. To our credit, we killed most



The Fury unleashes its wrath.

of these before even starting development, but we were left with many others to be enthusiastic about for months.

Our ambition when mixed with our pride also set us up for some hard lessons. The first year of Gas Powered Games was the honeymoon period. Everyone was full of ideals and had the boundless energy required to discover their inevitable pitfalls. Many of us were excited to have shed the shackles of Big Brother software companies and finally be in charge of our own destinies. No more lame meetings, no more stupid schedules, no more dog-and-pony shows. We were not going to adopt any of the stale, dated, and oppressive habits of game companies past.

Four years later, we have a deep appreciation for organization. Chain of command, ownership, discipline, and planning are things that we hold in very high regard and are further developing for the future. We still don't like Big Brother, but we realize that Little Cousin may help us get more work done.

2. Aborted efforts. We spent too much time exploring dead ends and implementing unnecessary features. The game's basic gameplay principles and aesthetic goals remained consistent from the beginning of the game to the very end. Unfortunately, although we were clear on how we wanted the game to look and feel, the technical requirements weren't obvious. We were feeling optimistic, and anything and everything sounded like a good idea at the time. Design meetings felt like a shopping spree at the supermarket. We loaded up the cart and set out on a rather long road of



One of many serene alpine environments.

trial-and-error punctuated with discovery.

The animation editor we attempted to build is a good example of the trial-and-error approach. At the time we had a developer who was both skilled and extremely passionate about creating our own animation tool. Given our ambitious state of mind, we were easily sold on it, and the developer spent a better part of a year working on the tool. Unfortunately, a year into it his calling took him elsewhere, and the editor work came to a screeching halt. This serious setback forced us to reconsider the necessity of this custom art tool. Almost immediately we reexamined what was possible with 3DS Max as senior programmer Mike Biddlecombe dove in to create a few plug-ins that did the job beautifully. Lesson learned: always use third-party tools when they're available.

Although the aborted editor is our poster child of inconclusive work, there were many other technologies we spent a lot of time on but didn't use. For example, we started on OpenGL but ultimately moved to Direct3D. We also wrote level-of-detail systems and sophisticated collision algorithms that we ultimately scrapped.

3. Complex engine. We have created a powerful but ultimately complex engine. To our credit, the engine is stable, relatively well commented, and the source is consistent in terms of coding standards. However, the engine is sufficiently complex to make debugging and maintenance times often longer than expected. Complexity arose from several factors.

First, the engine is large enough that although in the local sense things are documented, in the global sense there are

implicit rules that are not. This unfortunately translated to a larger number of nonobvious bugs. The nonobvious bugs tended to be in the “incorrect result” category rather than in the critical program-failure category, which made them harder to track and slowed debug times. The implicit rules in the system made it more likely that a seemingly simple change might create an unwanted side effect down the road even if we did get correct behavior in the short term. The best examples of this problem were single-player changes that worked absolutely great but broke something in multiplayer or simply failed to work in multiplayer.

Additional complexity can be attributed to a few elements of the engine that should have been abstracted but weren't. One example of this is our node-based coordinate system. The coordinate system reflects the segmented nature of our terrain. Every position primitive includes an ID of a terrain block (node) and a 3D coordinate within that node. Global coordinates aren't valid for more than one simulation, so if you want to work with a point outside of your particular node, you have to do space conversions. This introduced extra steps to every space calculation and increased the risk of human error. We got used to working with this system and shipped the game without abstracting these artifacts away, but we paid a high price in maintenance of affected code. In the future we hope to wrap this up in a cached global coordinate system that would make our vector math look like the math to which everyone else is accustomed.

4 ● Slipped schedule. When I started this Postmortem, I wanted to avoid talking about the two most common problems faced by development teams, communication and schedule. Every single project seems to have trouble here. I've managed to steer clear of communication rants, but I must mention our schedule, since we made a spectacular game but spectacularly late. Originally we were aiming to finish in just about two years, but ultimately we shipped in almost four.

During this time, however, we weren't just building a game, we were also build-

ing a company. We spent a considerable portion of our time simply learning how to be a company and learning how to make an RPG, something that we had never made before. Learning and ambition aside, our schedule first slipped considerably when three out of the six engineers left the company about a year into the project. They each had their own reasons and they didn't leave in unison, but the blow dealt to our planning put the engineering schedule into a permanent state of shock.

After that first year, the turnover at the company was very low, and it certainly wasn't the only culprit regarding the slipping schedule. In order to make a competitive title that was original, we essentially had to invent new features that we would use to compete with existing games. This goal simply required a lot of research, which is inherently difficult to plan or schedule. It also tends to lead to . . .

5 ● Epic crunch. Our crunch was not your garden-variety crunch. Our crunch was not measured in months, but in years of long hours and few free weekends. It was a crunch of sheer astronomical proportions. The kind of crunch that will make you weak in the knees to think about and give you a thousand-yard stare when you're done with it. The kind of crunch that bites the top of the beer bottle off and spits it out at you before taking a drink.

This kind of crunch can only take root in fertile soil, so we certainly are responsible for our own discomfort. Being the ambitious bunch that we are, not only did we want to make a great game, we wanted to make the best game we could afford, given the time and resources. This line of reasoning, taken to its limit, translated to most of us simply living for the game.

Our ambition was so strong we neglected our bodies, our relationships, and our spiritual well-being. I know many of

us did this past a point that most professionals would consider healthy, sane, or possibly ethical. The level of focus on our work was simply amazing, but the sheer self-neglect this fostered was simply irresponsible. We didn't crunch to make up for lost time, we crunched out of uncertainty. Since this was our

first RPG, we crunched to implement all those extra features. And at some point we crunched because we could no longer remember doing anything else.

A wise person once said that your passion can become your prison. We were so passionate about making DUNGEON SIEGE that we completely lost ourselves in its creation.

Having a tremendous desire to succeed is a noble trait, but being unable to reconcile that desire with the actual cost of attaining your goal is not. It's difficult to be critical of ourselves when we all cared so deeply about the game and worked so very hard to build it. But we must remember that we make games. We are toy makers, and we have a responsibility to our own humanity as well as to our trade. We must strive to live balanced and enriched lives so that we may always have inspiration from which to draw.

Lessons from the Dungeon

DUNGEON SIEGE was an extremely challenging project. Four years is a long time to work on a game. By comparison, it's long enough to earn a bachelor's degree in college, or go back for a Ph.D. Four years is time enough for births, deaths, weddings, breakups, and earthquakes. A lot of life happens in four years, and a lot of learning takes place.

Clearly, we walk away from DUNGEON SIEGE with valuable experience, both from what we did well and from our missteps. With experience comes perspective, and our perspective is much broader now than when we started. For better, for worse, and for everything in between, we will feel the reverberations of this experience for a long time to come. 🐉



Boiling Hot or Blowing Steam?

Today's Digital Game Distribution Market

Over the past decade or so, the vast majority of computer games have been published under a business model that looks something like the following: After a year or two (or more, frequently less) of development by a developer, a game gets handed over to the publisher, who manages manufacturing, package design, and assembly, then arranges for distribution via retail outlets. Prompted by a slick marketing campaign, a consumer spots the title on the shelf and makes a purchase. The developer (sometimes) gets a fat royalty check, the publisher (occasionally) gets lots of sales, the retailer moves units, and the consumer gets a wonderful gaming experience. A perfect example of market forces at work, right?

For the most part we'd be correct. Like many young industries do, the gaming industry has undergone a phase of meteoric growth. Market forces pressure strong publishers to become larger and more efficient, while weaker publishers scramble to form partnerships, find profitable niches, or close their doors.

From the standpoint of the independent game developer, the current system increasingly favors large software publishers with the distribution and marketing muscle needed to get vital shelf presence at retail outlets. Excluding in-house production via publishers and the dozen or so developers who have the resources to remain largely independent, taking even a small computer game from concept to shiny box at the local Wal-Mart can be a feat of Herculean proportions.

Recent developments in digital game



distribution are beginning to offer a glimpse of a future where many games — as purely digital content — could be distributed in their entirety over the Internet, giving small game developers new opportunities to get their wares to market.

At one end of this spectrum is the sale of small games over the web, currently being championed by such online services as Real One Arcade and Shockwave.com. Users play a free version of a small game online, then pony up a small fee — usually somewhere between \$10 and \$20 — to download the entire game. Most of these games aren't available in normal retail channels, giving independent developers a new way to reach paying consumers.

At the other end of the spectrum, digital content delivery systems — optimized for broadband Internet connections — promise to allow full-price commercial PC games to be rented online (via streaming media technology) or pur-

chased and downloaded outright.

A host of new start-ups and technologies have emerged to make digital game distribution both secure and a viable profit stream for retailers and publishers. TryMedia's ActiveMark technology enables retailers and publishers to create secure digital distribution networks of their own, or rely on TryMedia's own distribution service.

TryMedia thinks consumers are ready for digitally distributed content. According to their research, 73 percent of people who purchased a game online said they would prefer to buy and download all of their games.

Arguably one of the most talked-about of the new digital distribution technologies is Valve's Steam platform, which Valve president Gabe Newell announced at GDC 2002. Like other content delivery systems, Valve hopes Steam will supplement the physical distribution and sales of game software by allowing broadband-equipped users to purchase and download games directly over the Internet.

As promising as digital distribution may look, one need only look at the plight of the music industry to see how the future may look threatening to nervous game publishers and retailers. Thanks to Napster and its brethren, the music industry has been reeling from the proliferation of readily available file-swapping utilities and an Internet with a seemingly limitless appetite for free content. Why should game players buy what a few mouse clicks and a fat download could obtain for free?

Steam seeks to solve that problem by

continued on page 55

continued from page 56

making the shrink-wrapped product and the online component two parts of a cohesive, inseparable whole.

“It’s pretty hard to pirate code that is always trying to call back to its creators,” says Newell. “As games look more like a service — connecting to servers to acquire new content, to connect with mods and other players — it’s going to be increasingly difficult to pirate, as you have to figure out how to pirate an entire system, including back-end servers you can’t get physical access to.”

Since the dawn of the PC, the advent of significant new technologies — VGA, CD-ROM, 3D graphics, and the Internet — have offered challenges for existing

players in the market and new opportunities for those quick and smart enough to see the often-imperceptible scribbles on the wall. Savvy retailers and publishers will undoubtedly harness digital game distribution to help them push the computer gaming industry to new and even greater heights. Those same technologies should also empower small game developers to create new and exciting game content that may have never seen the light of day under the current system.

That’s what Valve’s Newell would like to see happen. “We certainly hope it opens up the world for a wider group of developers, and we also hope that it will allow for riskier game designs and alternative content development models to

flourish,” he says. “I was talking to Warren Spector . . . and he summed it up as ‘This is the way all developers want to be — it’s just a question of when the transition will occur.’” If the advent of digital game distribution results in more innovative game designs, additional avenues of distribution, and increased competitive pressure, then game players, developers and the gaming industry as a

JEFF JAMES | *Over the years Jeff has worn many hats: starving freelance author, game magazine editor, web site manager, and videogame producer. He is currently a senior producer for the Internet division of the LEGO Company. Comments are welcome at jeff.james@america.lego.com.*
