

gd

GAME DEVELOPER MAGAZINE

FEBRUARY 2004





GAME PLAN



LETTER FROM THE EDITOR

Quality, Not Quantity

Whether it's talking, being the root of all evil, not growing on trees, or making the world go 'round, money's on our minds this month with the publication of our Third Annual Salary Survey (page 34). There are few greater pleasures in life than being paid to do something you love, and the news among our salary respondents is good overall: they're more experienced, getting paid more, and enjoying more perks.

The real question is how to ensure developers continue to love what they do. The industry's perpetual evolution produces a steady stream of new opportunities for talented developers, but these opportunities are being undermined by those aspects of game developer life that remain, often to developers' great frustration, unchanged.

The biggest problem is lack of stability. Large companies can inflate and deflate project teams so quickly that, upon going gold, many developers are unsure whether to expect a bonus or a pink slip. Conversely, developers can jump from small studio to small studio and never feel more job security than they would if they were working in someone's garage. Those few mid-sized studios in-between only offer employees a taste of both extremes.

Beyond just recognizing the problem, employers must realize that, while stability is desirable to most, it means different things to different people. For some it's fertile ground for professional growth or advancement. For others it's insurance benefits, a share of profits, or a retirement nest egg. The single biggest overall change I've seen in game developers in the past few years is thousands of young, transient, empty-apartment dwellers turning into family men and women. What a difference turning 30 makes—just ask PONG.

The commercial demands of game development awkwardly straddle aspects

of both staid corporate IT and chaotic Hollywood production models. Each of those industries has developed its own means by which to provide employees with the security they desire: IT through the relative predictability of mature software engineering practices and comfy, big-corporation benefits, Hollywood through guilds and unions. That's not to say either industry is immune to periodic downturns, offshoring trends, or other threats to ongoing industry-level stability, but they have found a way to provide their talent on an individual level with some kind of safety net.

If consolidation trends continue, a smaller number of larger employers will be able to provide stability with benefits but not immunity from layoffs at the end of every project cycle, which cast developers back into the pool of having to choose between risky small ventures, more revolving doors, or just heading off to another tech or entertainment sector and taking their irreplaceable knowledge with them. Unionizing seems an unlikely near-term turn of events, so what's the right way to protect developers and promote long-term career and industry stability?

I don't have an answer, but at least I'm not the only person wondering. The International Game Developers Association recently formed the Quality of Life Committee (www.igda.org/qol) to address these and other issues pressing to developers. Get involved in finding viable solutions for the human side of the game-business equation. Share your thoughts at the Committee's roundtable sessions planned for this year's Game Developers Conference, and help get the Committee's whitepaper off the ground. Because your skills, talent, and experience are worth more than just a paycheck.

Jennifer Olsen
Editor-in-Chief

GameDeveloper

www.gdmag.com
CMP Media, 600 Harrison St., San Francisco, CA 94107 t: 415.947.6000 f: 415.947.6095

EDITORIAL

Editor-in-Chief

Jennifer Olsen jolsen@cmp.com

Managing Editor

Jamil Moledina jmoledina@cmp.com

Departments Editor

Kenneth Wong kxwong@cmp.com

Product Review Editor

Peter Sheerin psheerin@gamasutra.com

Art Director

Audrey Welch awelch@cmp.com

Editor-At-Large

Chris Hecker hecker@d6.com

Contributing Editors

Jonathan Blow jon@number-none.com

Noah Falstein noah@theinspiracy.com

Steve Theodore steve@theodox.com

Advisory Board

Hal Barwood Designer-at-Large

Ellen Guon Beeman Monolith

Andy Gavin Naughty Dog

Joby Otero Luxoflux

Dave Pottinger Ensemble Studios

George Sanger Big Fat Inc.

Harvey Smith Ion Storm

Paul Steed Microsoft

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: 404.658-1415

Account Manager, Northern California & Midwest

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

Account Manager, Western Region & Asia

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

Account Manager, Target Pavilion, Education, & Recruitment

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

ADVERTISING PRODUCTION

Advertising Production Coordinator

Kevin Chanel

Reprints

Julie Rapp e: jarapp@cmp.com t: 516.562.7081

GAMA NETWORK MARKETING

Director of Marketing

Michele Maguire

Senior Marcom Manager

Jennifer McLean

Marketing Coordinator

Scott Lyon

CIRCULATION



Game Developer
is BPA
approved

Circulation Director

Kevin Regan

Circulation Manager

Peter Birmingham

Asst. Circulation Manager

Lisa Oddo

Circulation Coordinator

Jessica Ward

SUBSCRIPTION SERVICES

For information, order questions, and address changes

t: 800.250.2429 or 847.763.59581 f: 847.763.9606

e: gamedeveloper@halldata.com

INTERNATIONAL LICENSING INFORMATION

Mario Salinas

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

EDITORIAL FEEDBACK

editors@gdmag.com

CMP MEDIA MANAGEMENT

President & CEO

Gary Marshall

Executive Vice President & CFO

John Day

Executive Vice President & COO

Steve Weitzner

Executive Vice President, Corporate Sales & Marketing

Jeff Patterson

Chief Information Officer

Mike Mikos

President, Technology Solutions

Robert Faletta

President, CMP Healthcare Media

Vicki Masseria

Senior Vice President, Operations

Bill Amstutz

Senior Vice President, Human Resources

Leah Landro

VP & General Counsel

Sandra Grayson

VP, Group Publisher Applied Technologies

Philip Chapnick

VP, Group Publisher InformationWeek Media Network

Michael Frieder

VP, Group Publisher Electronics

Paul Miller

VP, Group Publisher Enterprise Architecture Group

Fritz Nelson

VP, Group Publisher Software Development Media

Peter Westernman

VP & Director of CMP Integrated Marketing Solutions

Joseph Braue

Corporate Director, Audience Development

Shannon Aronson

Corporate Director, Audience Development

Michael Zane

Corporate Director, Publishing Services

Marie Myers



United Business Media

GamaNetwork

SAYS YOU

A FORUM FOR YOUR POINT OF VIEW. GIVE US YOUR FEEDBACK... 

Mocap, My Precious

I really enjoyed Ed Hooks's article about motion capture and acting ("Chasing Gollum," December 2003). Most of the articles written about motion capture are boring, technical, and not really helpful. I was a motion capture animator for seven years in the video-game business, and recently wrote a book on motion capture, scheduled to be released in February 2004.

It seems we have a lot of the same ideas about the importance of the motion performer and setting up each shot to reflect the mood. Unfortunately, a lot of animators think, "I can change it however I want after the fact," which is an awkward way of looking at it.

Matt Liverman
via e-mail

Projecting Confusion

I enjoyed Steve Theodore's Artist's View column on "Procedural Textures" (November 2003). I am interested in exploring this subject more myself. One of the things that confused me in the article was his use of the term "projection." I associate the term with UV coordinates, but is a projection simply an instance of a procedural being applied?

Stefan Henry-Biskup
Liquid Development, via e-mail

Steve Theodore replies: *a "projection" is any method of relating a 3D object to a 2D texture. If you freeze that projection you get a UV mapping, but you can also keep the projection live (in 3DS Max, this would mean setting the individual UV map gizmos to create different UV channels). In the example I was using in the article, I was suggesting multiple projections (such as front, side, and back) for painting the basis of a texture and then using render-to-texture to combine them into a single UV map, since very few realtime engines can handle multiple UV sets. The advantage of*



that is you can paint the pieces at any resolution and orientation you want and still get them down into single, densely-packed UV map.

Common Sense++

In response to Jonathan Blow's "Predicate Logic" column (The Inner Product, December 2003), I too have seen plenty of over-engineering in game code over the past few years. People try to get too tricky and fancy with object-oriented design and C++ features combined. As he states in the article, "they shoot themselves in the foot."

However, I think he's throwing the baby out with the bathwater. My last project at my previous studio was started from scratch for the Playstation 2 using middleware for rendering. We had six programmers total, two of which had never shipped a game and two others that had shipped one game prior. In 10 months, we completed the project on time and under budget while not only keeping features, but adding some fairly major ones in as well. We were heavily object oriented and fully C++. Some of this can be credited to our lightweight production process and software design

philosophy. But I wholeheartedly believe that our overall engine architecture is what saved us. We didn't template everything, we didn't have ridiculous hierarchies or monolithic systems. We just had common sense.

Paul Reynolds
Humongous Entertainment/Atari, via e-mail

Striking a Nerve

Jonathan Blow's column "Predicate Logic" is very presumptuous. Bold statements are good, and everything he says makes sense. But he makes a lot of jumps and assumptions. In my opinion, the real question should be: do OO practices improve things over what they were before OO? The reason the programming community has so widely and completely embraced OO is: the answer is yes, a resounding and absolute yes. As tangled as OO can get to be, the spaghetti code that was used before it came along was far worse.

So the second question becomes: does OO have problems? A corollary question may be: is OO doing all it was thought it would do? The inevitable answer is that it does have problems and that it isn't as amazing as was initially hoped.

Hyrum Tanner
BYU—Center for Instructional Design
via e-mail

Jonathan Blow replies: *I agree with you entirely on these points. But I am not trying to say in the article that all OO is bad. What I am trying to say is, the game industry is currently going through a period of overzealous commitment to the OO paradigm, wherein anything that isn't object oriented and extremely formal is considered bad. I think that's very damaging and a big mistake.*



E-mail your feedback to
editors@gdmag.com, or write us at
Game Developer, CMP Media LLC,
600 Harrison St., San Francisco, CA 94107



INDUSTRY WATCH

KEEPING AN EYE ON THE GAME BIZ | *kenneth wong*

VICE CITY still under fire. Responding to criticism from the Haitian American groups, Rockstar's parent company Take-Two issued an apology and promised to change a portion of *GRAND THEFT AUTO: VICE CITY* that featured the mission objective "kill all the Haitians" (a reference to a specific drug gang). The measures, however, fail to satisfy the protesters: Jean-Robert Lafortune of the Haitian American Grassroots Coalition says the sheer presence of the game constitutes "a clear and present danger for Haitian nationals." The latest lawsuit from the Palm Beach County Haitian American Coalition named not only the game makers but also several retailers (Target, Wal-Mart, and Best Buy) as defendants. This particular controversy did not surface until nearly 13 months after *VICE CITY*'s release.

PSX sold out in Japan despite ambivalent analysts. Sony's multifunction device PSX debuted in Japan to mixed reviews from industry analysts: Kazumasa Kubota of Okasan Securities called it a "publicity stunt" and predicted



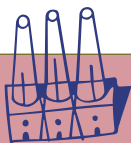
PSX debuted in Japan to a mixed reception.

sales momentum would dissipate after a month or two; Kazuya Yamamoto of UFJ Tsubasa observed that "lowering the specifications of the PSX hurt Sony's image." (Sony released the device without some of the features previously promised.) On the other hand, Hideki Watanabe of HSBC Securities noted that, compared to similar products from Matsushita Electric and Pioneer, the PSX is less expensive and highly competitive. Despite the skepticism of some analysts, the consumers depleted the first production run of the unit within a few days.

What's in a name? Microsoft, which once sued the makers of Lindows OS for infringing on its Windows OS trademark, is now being hit with a similar lawsuit from Mythic Entertainment, the makers of *DARK AGE OF CAMELOT*. Mythic alleges that the software giant's upcoming MMORPG *MYTHICA* is likely to cause confusion among the consumers. Mythic has previously raised objections to Microsoft's use of the name *MYTHICA*, but Microsoft refused to desist. Filed in the U.S. District Court for the Eastern District of Virginia (Alexandria), Mythic's suit seeks permanent injunction and economic remedies.

Sid Meier inducted into CMA Hall of Fame. Sid Meier, creator of the *CIVILIZATION* game series, is among the top five individuals the public has voted for induction into the Computer Museum of America (CMA) Hall of Fame Class of 2002 (www.computer-museum.org). The CMA Hall of Fame recognizes innovators who have contributed to the computer industry's milestone achievements. Meier will join other nominees past and present, ranging from Charles Babbage, inventor of the Analytical Engine, to Tim Berners-Lee, father of the World Wide Web. 🎉

Send all industry and product release news to news@gdmag.com.



THE TOOLBOX

DEVELOPMENT SOFTWARE, HARDWARE, AND OTHER STUFF

Intel Compilers 8.0. Intel released version 8.0 of its compilers, designed for optimizing performance of C++ (and Fortran) applications running on Intel processors. The Intel C++ Compilers for Windows and Windows CE .NET support optimization of code for Intel's range of CPUs, including its Xscale PDA processors as well as the current variations of its 32-bit and 64-bit desktop chips. www.intel.com/software/products/compilers/

Softimage Behavior. Softimage has shipped version 1.5 of Softimage Behavior, an IDE/SDK that allows users to simulate the behavior of crowds of any conceivable size, even crowds numbering in the thousands. New in version 1.5 is a batch-processing system that

can be used to model complex behaviors in the background, enabling you to continue working in the foreground. www.softimage.com

3Dconnexion supports DCC apps. 3Dconnexion has enhanced the drivers for its line of 3D motion controllers, providing expanded features and customization for the DCC market. Its Spaceball, Spacemouse, Spacenavigator, and Spacetraveler are now supported by 3DS Max, Cinema 4D, Bodypaint 3D, Maya, Motionbuilder, and Photoshop, with the mapping of some or all of the six axes optimized for manipulation of the on-screen view, camera position, or objects. www.3dconnexion.com

—Peter Sheerin



UPCOMING EVENTS CALENDAR

NARRATIVE GAMES

ZKM
Karlsruhe, Germany
February 6–12, 2004
Cost: Euro 400–Euro 1,600
www.sagas.de

WORKSHOP ON INTERACTIVE ENTERTAINMENT

UNIVERSITY OF TECHNOLOGY, SYDNEY
Sydney, Australia
February 13, 2004
Cost: \$50–\$150
<http://research.it.uts.edu.au/creative/ie/>



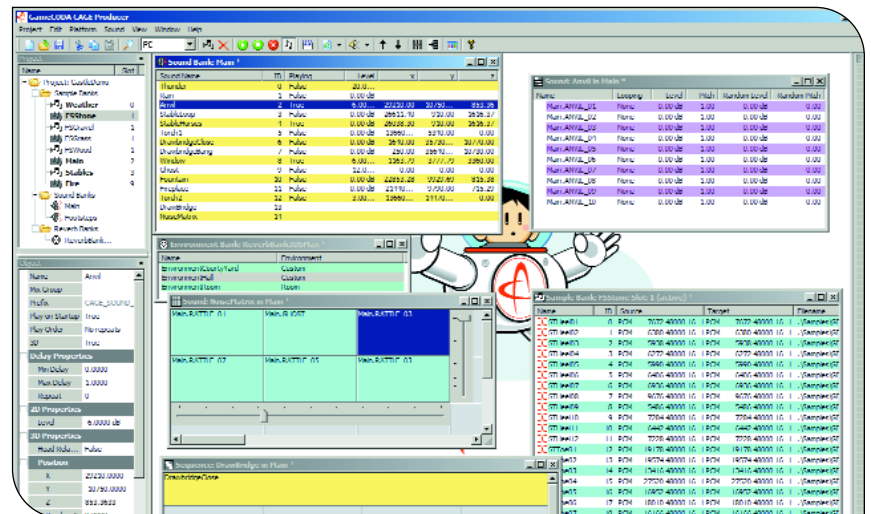
Sensaura's Gamecoda

by alexander brandon

Gamecoda by Sensaura is a complete audio solution for PC, Playstation 2, Xbox, and Gamecube (with internal development versions running on WinCE and Mac OS). Having only been about 14 months from its conception, it has yet to make a substantial impact on the scene, but Sensaura claims that at least two very large game publishers have already taken it on-board for a number of upcoming titles. I'll take a look at its functionality, scalability, and ease of implementation in this review. For this review, I tested version 1.5, the latest version available. (Version 2.0 was in beta at press time, and is set to be released at GDC 2004.)

Gamecoda makes its interface fairly simple, using a four-layered system, consisting of a Sensaura abstraction layer (SAL), low-level Gamecoda API, Gamecoda toolkit, and high-level CAGE (Console Audio Game Engine) game audio API with its tools (CAGE Producer and CAGE Plug-ins). The abstraction layer is the lowest level, sitting between the main CODA API and the actual hardware, making future ports to other platforms a much smoother process.

The Gamecoda API is the core of Gamecoda, containing the usual functionality found at this level, such as buffer and source creation and playback, along with many other features. Some of the main features include 3D audio, envelopes, LFOs, filters, seamless buffer queuing, and reverb. Software versions of all features are available, should they not



The containers in Sensaura's Gamecoda enable fine-tuned control through a logical interface.

be provided in hardware on the platform, thus making the API completely flat and truly cross-platform.

The high-level CAGE API integrates with the CAGE Producer application and provides a very simple interface (sound can be played back using only six calls), while adding all of the CAGE features such as automatic streaming, randomization, sequencing, and sound instancing. Full source code for this API is provided. While I can't comment on programmer interaction with the system, as it hasn't yet been released with a title, CAGE and the CAGE Producer are where I got to get my hands dirty, and are the sections that I think expose some of the most advanced functionality of the system.

CAGE organizes sounds in an abstract

sense as objects, rather than as individual files, which is the next level of sound file management in modern projects. The objects act as containers (known as "sample banks") for any number of sound files, be they .WAV, .VAG, or whatever format your platform and engine prefer. The containers can have a great number of properties set and can then be controlled in playback through another container known as a "sound bank." Thus, if you wanted to have 12 footstep sound variations and wanted to change their timing to be dependent on "walk" or "run," you could place all of your sounds in a single sample bank, and assign properties to two sounds in a sound bank: "walk" and "run." Footstep sounds on different surfaces can be accommodated by simply creating other sample banks containing the relevant samples, and a slot system allows for easy swapping of

ALEXANDER BRANDON | Alexander (abrandon@midway.com) has been involved with game audio since 1994 and is currently the audio manager at Midway in San Diego.



similar sample banks in the game. If you discover that a few of the “heel” sounds are too loud and don’t want to change the files themselves, simply create a mix group and you have full control of volume there. I found this aspect of Gamecoda to be straightforward and powerful, and it took just a few minutes to get a few samples together and play around with them in real time.

Properties exist in the user-friendly CAGE Producer GUI to manipulate 3D placement, reverb, volume, panning, repetition, and the properties of the files themselves. One particularly useful indicator tells you exactly what compression is used by any file (or set of files) that you click on. If you want to compress files to a particular platform’s codec, or resample them to a different rate, it will do this for you, singularly or in batches, leaving the original files intact.

Some extended functionality began to emerge later during my testing: interleav-

ing and matrices. With interleaving you can load multiple variations of the same track and interleave them into a single file for greater efficiency and synchronized playback. This can be useful for varied music playback as well as sound effects. It also means you can stream a stereo VAG file or Xbox ADPCM file. Interleaved streams can also contain markers, allowing the programmer to trigger in-game effects based on musical cues.

A matrix lets you set a “grid” of sounds that you can manipulate based on timing or other parameters as well. This gives far more depth than randomized playback, or even randomized playback with a bias against recently played sounds. I experienced this through a demo that simulated a car engine, with results that sounded pretty darn realistic. It’s incredibly versatile but a bit less obvious when it comes to methodologies. I can’t imagine what I’d use it for, but if it can simulate a car engine I’m sure it can do a lot more along two axes of sound playback.

CAGE has varied reverb settings based on different environments (if the player enters a large cavern, the reverb settings can reflect it). Unfortunately the settings are based on boxes, which can be difficult to manipulate with a complex level architecture. At least a basic set of primitives would be more useful.

Gamecoda also has taken a step forward by interfacing with two of the most used 3D rendering programs on the planet for game development: Maya and 3DS Max. While this is a great way for audio staff to help take some pressure off the level design staff, most studios have either engines such as Unreal or their own proprietary systems for level design, which are spread across widely ranging techniques. Some studios use a combination, having level designers build basic level shapes and artists touch up areas by importing prerendered objects, or importing levels entirely. For the latter method, Gamecoda is an absolutely perfect fit, but for others the game studio is left to find its own method of implementation of reverb zones and sound object assignments. Gamecoda does pick up the slack by providing easy ways to do this

in its documentation, and sensibly uses XML for its project file format. Covering more ground with existing engines would be the next step to take to cover as many bases as possible.

Finally, something that is helping shorten the increasing time taken for mixing sounds properly during the alpha and beta states of a game is interactive mixing. As games approach Hollywood levels of sound quality, having a good mix is becoming increasingly more important. Being able to ID sounds or groups of sounds and assign faders to them that control volume, filters, and such during gameplay is a feature showing up in some of the console dev kits, though Gamecoda has yet to provide this.

Gamecoda is one of the most advanced sound engines available and scores high for its cross-platform versatility and large list of features. But its youth shows, and it still has some growing to do to wear the king’s crown of game audio engines. Regardless, I’d recommend it for any kind of project for just about any genre. It’ll save studios without their own custom engine an awful lot of time.

Digimation’s Model Bank Collection

by tom carroll

Pretend for a moment that you’re in the 3D modeling and animation field. It’s 4:45PM and you’ve just put the wraps on a shot showing Japanese fighter aircraft strafing a Tyrannosaurus rex. Your boss enters the office and before you can say, “The shot’s a wrap, CJ,” he swivels your chair around and shouts, “The brass just saw the rushes—Zeros are out, Stukas are in; the T-Rex is now a monstrous anaconda that’s fighting a giant grasshopper. I’ll be back in 15 minutes and you’d better have something up and running!” What would you do (besides checking whether your résumé is current)?

If you have Digimation’s exhaustively thorough Model Bank Collection, a compilation of more than 4,600 detailed and fully textured 3D models, you can simply

GAMECODA ★★☆☆

STATS

Sensaura
Middlesex, U.K.
+44 (20) 8848-6636
www.sensaura.com

PRICE

\$10,000–\$25,000

SYSTEM REQUIREMENTS

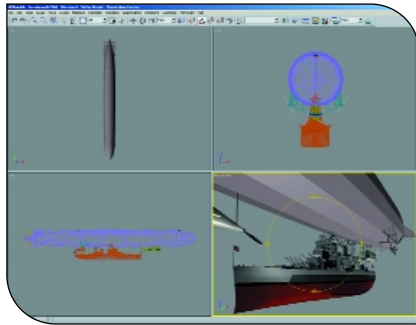
PC (Windows), relevant game console development kits.

PROS

1. Cross platform.
2. Advanced data management and file organization.
3. Less programmer low-level work and more control to the audio staff.

CONS

1. No real-time (interactive) mixing.
2. Needs engine editor support (Unreal, and so forth).
3. Reverb zones represented only as boxes.



A "dirigible dreadnought" assembled from Digimation's Model Bank Collection.

replace one set of models with another and concentrate on the rigging and animating. The Model Bank Collection spans nine CDs and contains 1,160 distinct models covering all things vehicular (military and civilian), animals, architecture, furnishings, plants, anatomy, and more.

Also, each model comes in four levels of detail (very high, high, medium, and low), which accounts for the 4,600 models in the total package. The availability of high-quality LOD models is invaluable to both cinematic and real-time videogame developers; judicious application of LODs helps speed rendering time without sacrificing the quality of the finished product.

While it's unrealistic to dig thoroughly into every model on the nine CDs, the ones I looked at were appropriately detailed, and each LOD was carefully matched with a respectable set of texture maps and materials. The selection menus for the Model Bank Collection are simple and effective (click on a category, select the model you want, highlight it with the cursor, and read the name and location of each model).

In the videogame industry, it's pretty common knowledge that many developers go to great lengths to build their own proprietary models. It's also common knowledge that when modelers have to rely on sketchy reference photos of various assets (or their own imaginations) the results can vary widely from how things look in real life. Add to the mix the problems that harsh deadlines can impose and you've got the picture. Digimation's models are very representational of the actual assets, which leaves the developer free to make changes to the geometry to suit the application or alter the textures to make the asset unique. Each model also comes with a bonus: both bump and reflection maps.

Digimation's CD-based Model Bank Collection is available only in .MAX format, which shouldn't be a problem for most users who are familiar with such conversion programs as PolyTrans. The full collection costs \$995, but check the company's web site for special offers as well as details on how the collection is available via the Internet. The online version supports such formats as: .MAX, .3DS, .OBJ, .DXF, .COB (TrueSpace), .X (DirectX), .VRML, and .W3D (Shockwave 3D).

A test project that I pursued was the creation of a hybrid zeppelin/battleship called a "dirigible dreadnought," something I never would have done if I had had to develop my own models. Via Digimation's Model Bank Collection, I had a serviceable dirigible dreadnought up and on-screen within moments, allowing me to concentrate on customizing this Hindenburg of battleships to suit my own needs. And that's where this collection shines most—putting credible assets in the hands of creative people.

★★★★ | Model Bank Collection
Digimation
www.digimation.com

Tom (jetzep56@yahoo.com) is an environment modeler for Rockstar San Diego.

Alias Sketchbook Pro 1.0

spencer lindsay

Although the advent of the tablet PC seemed innovative when I saw Bill Gates bring it out at Comdex a few years ago, I could never really figure out what I would do with it. It's a computer without all the input devices I'm used to. However, after downloading and trying out Alias Sketchbook Pro 1.0, I think I need to get one. The only thing missing from Sketchbook Pro is the smell of my eraser and the scratching of the pencil. Really. Its sensitivity and flow are exponentially better than Photoshop's. Though it doesn't have a lot of the features found in Adobe's Photoshop or Corel's Painter, it makes up for their lack in its speed (not to mention

★★★★★ excellent
★★★★ very good
★★★ average
★★ disappointing
★ don't bother

price). Featuring an amazing set of pencil and marker tools, as well as a pen, eraser, airbrush, smear tool, custom brush editor, and a good layer feature, Alias Sketchbook Pro is a solid first release.

Upon opening Sketchbook Pro for the first time, I was a bit intimidated by the lack of visible tools. All you get is a white page with a small graphic tool pallet in the bottom-left corner. After reading the docs and playing with it a bit, however, I found the tools easy to find and use.

Most tools are selectable via a flick of the stylus, and the color mixer offers functionality similar to that found in Painter's excellent palette tools, and is extremely easy to use. The pencil and marker tools were my favorite. Most paint programs make an attempt to get the marker look correct, but this is the first program I've used that really made me believe that I was using a marker.

Although Sketchbook Pro has many great features, it also has a few not-so-great ones. Moving around in the sketch environment is extremely tedious. Instead of the simple "spacebar and drag" of Photoshop, there's a "zoom and move" tool that drove me nuts with its counter-intuitive gizmo floating at the end of my pen. Another bother was that there was no way to assign hotkeys to often-used commands. I know that this was built for Tablet PCs, but adding hotkey support would make this a much easier tool to use, both on desktop workstations and with those Tablet PCs that feature keyboards. The eraser tool was sometimes linked to my stylus eraser, and sometimes not. Finally, a lasso tool and .PSD import would add some essential functionality.

Although I'm sure that 2.0 will shake some of the bugs out, at \$179.00 for the download version and \$199.00 for the CD version with printed manuals, Alias Sketchbook Pro is a great addition to any digital artist's tool set. 🎨

★★★★★ | Alias Sketchbook Pro 1.0
Alias
www.alias.com

Spencer (slindsay@rockstarsandiego.com) is a technical artist for Rockstar San Diego.

Emerging from the Tunnel

Ion Storm's Randy Smith on distinguishing the art of games

We caught up with Randy Smith just as he was putting the finishing touches on THIEF 3 as project director at Ion Storm.

Randy has worked on the stealth-based THIEF series since its inception, starting out his career as a designer at Looking Glass Studios. This latest THIEF integrates current techniques such as emergent gameplay, although there's much more going on, as evidenced by Randy's introduction of the ability to see your character's limbs from the first-person perspective.

Game Developer: What challenges were posed by THIEF 3's "body awareness" feature?

Randy Smith: The feature has some of the harder challenges from implementing both first-person and third-person view modes. For example, in most first-person games you don't have to show your character animating when climbing, but with body awareness you have to, because the player character is visible and animating in the world. In most third-person games, you don't have to line up the character with world geometry all that precisely, but with body awareness you have to, because the camera is so close to the player character's model. Also, the camera is attached to the player character's head, so you need to create animations that hold the head steady in addition to being aesthetically pleasing, which is really hard.

GD: How do you define emergent gameplay?

RS: Emergent gameplay is the phenomenon in which gameplay challenges or solutions to challenges emerge (possibly unexpectedly) as a second-order consequence of game systems interacting with each other. So, say you've got AIs who chase the player, and you've got pressure plates that detect weight on them and trigger traps. The pressure plates were placed expecting the player would walk over them, but the design is that clever players can also lure AIs to set off the traps. Luring the AIs onto the pressure plates is an expected example of emergent gameplay in which the AI system and the physics system interact. Then, during playtest, you discover that some clever players are tossing objects onto the pressure plates to set the traps off, which is an unexpected example of emergent gameplay, in this case an emergent solution. There's also emergent problems, such as when the AI on the pressure plate gets killed by the trap but then a friendly AI hears the noise and as a consequence starts walking towards the pressure plates to investigate—suddenly the player has to protect



Randy Smith demonstrating the importance of emergence.

that AI from the pressure plates, which is an emergent problem.

GD: What role should it play in future games?

RS: As you can see from the examples, emergent gameplay supports player choice and expression in a way that you can't get in a game where every possible challenge, solution, and outcome is understood and explicitly implemented ahead of time by the developers. The important thing to me is the fact that interaction is what sets games apart and makes them a unique art form. If the history of other art forms is any indication, then I believe the future of interactive art is in more complicated forms of interaction, and emergence is likely to be a designer tool which contributes to pioneering that future. But it's probably the case that whether entertainment software follows this development is up to fickle consumer demand.

GD: What other tools do you think are worth experimenting with in distinguishing the interactive qualities of games?

RS: Well, I think simulation is going to continue being really important for empowering player expression and sophisticated interaction. If you don't have at least a little simulation in your game, if everything is emulated, then the most sophisticated player expression you can achieve is still discrete, and that's a pretty limited form of interaction and expression.

Another tool I'm interested in right now is narrative. During THIEF 3's development, we experimented with the narrative presentation. The player is presented with some ambiguous but emotionally charged material towards which they can express a variety of reactions using their standard in-game tools. The game detects and responds to a handful of possible non-mutually-exclusive responses. This design is meant to capitalize on a player trend we noticed, in which players would unexpectedly contribute to the background story in various ways, such as knocking out all the guards and leaving them in one room.

There's an assumption that videogames are supposed to be somewhat realistic experience simulators. I think once interactive art really establishes itself as a fine art, as opposed to simply an entertainment medium, then this assumption too will start to be questioned. Again, this is pretty parallel to the history of other art forms, but it's probably a long way off.

GD: What games are you playing now?

RS: MARIO KART: DOUBLE DASH!!, DEUS EX: INVISIBLE WAR, THE LEGEND OF ZELDA: THE WIND WAKER, ANCIENT DOMAINS OF MYSTERY, and DECKER. 🎮

Designing the Language Lerp: Part 2

Two months ago I looked at predicate logic as a possible way to help simplify game programming; last month I introduced a programming language called Lerp. The paradigm behind Lerp is a fusion between traditional imperative programming and the declarative style of predicate logic.

Toward the end of last month's article, I implemented C-style **structs** as syntactic sugar over predicate logic databases. This means that the programming language has very good introspection support, and all sorts of interesting pattern-matching queries can be performed on fundamental data structures.

All the time, random people propose high-level programming features that sound good at first but, when applied to real problems, are not actually helpful. (In fact, this seems to be the norm for new language features.) Not only have I proposed a feature unproven in the world of games, I have used it to replace **struct**, the traditional workhorse of the imperative language. Given all this, the onus is upon me to show that the database features are useful for nontrivial real-world problems.

A Simple Real-World App

I thought it would be fun to choose a program from an earlier column and port it from C++ to Lerp. After a bit of consideration, I chose the application from "Interactive Profiling: Part 1" (December 2002). The program is relatively simple; you have a standard mouse-look and keyboard-walk interface for moving around the world, the ground is flat, and there are about 100 crates scattered around. The crates are rotated at different orientations, and there are a number of different textures on them. Originally, the point of this program was to provide a simple world for a profiler to run on. Since profiling is not the point of this exercise, to help keep things simple in the port, I reduced the profiling information to a traditional frame counter.

This application has been a good crucible for the early design of the language. I started with a vague idea (I wanted to base a language on predicate logic) and selectively refined the language based on the demands made by concrete programs like this one—hopefully keeping myself grounded in reality through the process.

Nature of the Test Program

One interesting aspect of this test program is that it draws all those crates at a very low level. It uses `glVertex()`, `glTexCoord()`, and `glColor()` to render each crate at one vertex at a time. Lerp is intended to be a scripting language, and in a real

game, you'd rarely use the scripting language to render at this low a level. Instead you would use a `render_mesh()` function, which would be implemented in fast and efficient C++ code.

In fact, this test application is so low-level that it even computes the lighting for the faces itself, by dynamically computing surface normals and dot-product-ing them with the light direction. If the program were trying to do less work, it could store surface normals for each entity and tell OpenGL to do the lighting itself.

As a result of all this, the program is actually much more stressful than necessary to meet our domain requirements. If it can be made to run quickly, we're in good shape.

Program Fundamentals

The test app uses a `Vector3` class to represent points, and it does a bunch of math on those points, passing the results to the aforementioned OpenGL calls to render the world.

Naturally I need to implement a way to call OpenGL functions from Lerp. That's straightforward enough; I can just code wrappers in C++ that can be called from Lerp.

I could also implement a `Vector3` class in C++, manipulated via wrappers, but that would be a cop-out. The whole point of this program is to stress-test Lerp's data structure-handling mechanism, and `Vector3` is the fundamental data structure of this program. So `Vector3` needs to be implemented using the database features.

Implementing Vector3

How to implement a `Vector3` as a database? I could take the approach we tend to use in C++, which is just to have some named slots `x`, `y`, and `z`, with a floating-point value for each. But lately I've been rethinking these named coordinates, as they don't generalize very well to higher dimensions—for the fourth dimension, sure, we can use `w`, but beyond that the letters just get arbitrary. And if Lerp is going to be a high-level language, the code I write for `Vector3` should just be reusable for `Vector4`, `Vector11`, or any other dimension. So, whereas I might introduce the ability to use names as aliases for the dimensions at some later date, for now I am just going to num-



JONATHAN BLOW | Jonathan (jon@number-none.com) is a game technology consultant who consults with people about game technology.

ber the dimensions 1, 2, and 3. This is in keeping with the differential forms and Clifford algebra traditions of naming dimensions e_1 , e_2 , and so forth, except I am leaving off the e . Also, it introduces uniformity between the way we reference vectors and matrices (we use integer subscripts for matrices).

So a `Vector3` will just be a database containing facts that tell me what the coordinates are. Each coordinate will be represented by one fact, perhaps like this: `[`coordinate 1 0.337]`, meaning, “The value of coordinate 1 is 0.337.” But on second thought, every fact stored in a vector is going to have ``coordinate` as the first entry, so we can just eliminate that redundancy. Hence the facts will look like `[1 0.337]` or `[3 1.562]`. From now on I will also call facts tuples, since really what we are looking at here are tuples of arbitrary data elements.

For a `Vector3` to be valid, the facts inside it should be two elements long, the first element being an integer, the second element being a floating-point number. (There’s another important constraint that we’ll talk about later.) It would be nice to have some error checking, so the program can tell us when we make mistakes. So I created the ability to declare a database schema inside a `struct` declaration. The schema controls which facts can be stored in a database, and it provides error checking as well as program documentation. For a `Vector3` it looks like this:

```
struct Vector3 {
    [?Integer ?Float];
}
```

The syntax for a schema uses square brackets just like a fact tuple would. This schema indicates that the first slot can be any integer and the second can be any float. But we ought to be more specific than this; the first coordinate of a `Vector3` can’t be just any integer; it has to be between 1 and 3. So I added some new syntax to declare this in a compact way:

```
struct Vector3 {
    [(1..3) ?Float];
}
```

At this point, it’s easy to see that an optimized version of `Lerp` could look at this declaration and know to store the `Vector3` floating-point values compactly into an array. That’s good to keep in the back of our minds, but we won’t obsess over it now.

Since I added this concept of “..” indicating a range of integers, I made it work in imperative expressions also. You can say `each (1..n) {...}` and the code will loop n times over the block. This is used in the example code to initialize the crates.

Syntactic Sugar for Vector3

Using the syntax introduced last month, we can manipulate this new `Vector3` type. Supposing we have a `Vector3` called v , we can find its x coordinate by evaluating `v.[1 ??]`. In other words, for the tuple with 1 in the first slot, what is the value in the second slot? But what’s interesting—and very hard to do in

C++—is that we can also perform queries that go backward from the usual direction. So we can say things such as `each v.[?? 0.0]`. In other words, give me a list of the coordinates that are zero.

Still, the `v.[1 ??]` seems cumbersome; in C we would just say `v.x`. Also, we need to think about initializing the vector. In C, we can say `v.x = foo`. As of last month’s sample code, we used the operator “.+” to add things to databases (and “.-” to remove them); so to set coordinate 1 of the vector, we’d say `v .+ [1 foo]`.

But we have a consistency problem here that can cause more tedium. Not only must each tuple of v be well-formed, but there should be only one tuple for each spatial coordinate. If v has two different entries with 1 in the first slot, we’re in trouble. So before saying `v .+ [1 foo]` as shown above, we need to say `v .- [1 ?]`. In other words, remove any tuple with 1 in the first slot. If we don’t remember to do that, we create a glaring data inconsistency. That requirement is annoying and it encourages errors. Furthermore, there’s no way for the system to catch the errors, because the compiler doesn’t know about this constraint.

To help improve error checking, and to make programming nicer, I added the idea of a domain/range relationship to the struct schema declaration. The power of the database approach is that we can query the data in a freeform way, so we’re not restricted; but often, as with the `Vector3`, we are modeling functions (not arbitrary relations). In these cases, one direction of query is forward, and another backward. We want to make it easy to talk about the forward direction, since that will be the most common case.

So in the schema, you can use the symbol “|” to indicate a division in a tuple: everything to the left is the domain; everything to the right is the range. The definition of `Vector3` becomes:

```
struct Vector3 {
    [(1..3) | ?Float];
}
```

Sometimes the domain of a tuple won’t be on the left-hand side. In those cases, I provide an alternative syntax for declaring it; see the sample code if you’re interested.

Now at least if we assert two tuples for the same coordinate, the system can signal an error. But I also added some syntactic sugar to imperative expressions, using C-style array subscripting syntax. If you say `v[1]` as an r-value, that’s equivalent to the expression `v.[1 ??]` (since the compiler knows by looking at the schema that the domain of v is one element wide). If you use `v[1]` as an l-value, as in `v[1] = foo`, the compiler removes from v any old tuple that matches `[1 ?]` before adding the new tuple `[1 foo]`.

Using this nice brief syntax, we can define basic vector operations like addition, scalar multiplication and the dot product. Listing 1 shows two examples. They are pretty neat, because they are brief and they work on sparse vectors of arbitrary size. The sparsity part requires some small language additions that are beyond the scope of this article; also notice that I have gen-

eralized in the Listing from Vector3 to Vector. Thinking about implementing these features is, for now, an exercise for the interested reader. I like that I can define a versatile vector class so simply and clearly.

The Matrix4

Writing a simple 3D application involves using some matrices as well as vectors, so let's look at a matrix definition:

```
struct Matrix4 {
    [(1..4) (1..4) | ?Float];
}
```

It's very much like a vector, except the tuple is three elements wide because there are two indices. Now, suppose I have some matrix m and I want to extract the first column vector from it. Nicely, the database query operation just provides the right answer for us—we don't need wrapper functions or anything. We just evaluate $m.[? 1 ?]$ and the result is a database consisting of length-2 tuples that fill in the question marks; in other words, the values from all tuples that have a 1 in the second slot. This has the same form as a Vector4, but for the purposes of type checking, can the compiler know it's supposed to be a

LISTING 1. BASIC VECTOR OPERATIONS

```
proc dot_product(Vector a, Vector b) {
    return + each a[?i] a[i] * b[i];
}

proc *(Vector v, Float factor) {
    Vector3 result;
    each v[?i] result[i] = v[i] * factor;
    return result;
}
```

Vector4? There's a sense in which it can; if you write down the schema for this resulting database, it's just the schema for Matrix4 without the second slot: [(1..4) | ?Float]. That matches the schema for Vector4, so if we allow anonymous databases to type-match based on their schemas, we achieve some pretty reasonable type-safety.


Here's another trick: the trace of a matrix m is just $+ \text{each } m[?i \ ?i]$. Note that we're using the array subscript notation for brevity, as we did with vectors; since we've specified the same variable name for both index slots, the values in the two slots must be equal, so the `each` iteration only travels along the diagonal of the matrix. Math operators in Lerp automatically expand across `each` just like function calls. For example, $+ \text{each } (1..10)$ evaluates to 55, $* \text{each } (1..5)$ evaluates to 120.

I get warm, fuzzy feelings from this. These expressions are so short and simple that they probably don't justify creating a wrapper function in many cases—if you want to get a column vector from m , you may not want to call some `get_column_vector()` function, when you can just say $m.[? n ?]$. This is interesting because it may allow compound math expressions to merge more organically than they might otherwise.

But How Does It Run?

When you try out this month's sample code, you'll find that it runs at a reasonable speed. On my 1.5GHz Pentium-M laptop, it clocks in at 30 frames per second. Keep in mind that the language system is almost entirely unoptimized—the bytecode is bloated, the function interfaces are slow, the code garbage collects every frame, and all the databases are still implemented as linked lists. Despite all this, the application runs at a smooth frame rate. If nothing else, this goes to show that computers are awfully fast these days.

Source Code and Next Month

I've demonstrated that, despite its foundations in wacky language features, Lerp can accomplish real-world tasks. I encourage you to download the sample code (available at www.gdmag.com) and play with the interpreter yourself. Next month, we'll look at some all new pattern-matching language features. 

A Joint Effort

As we all know (much too well), skeletal deformation isn't a realistic way of representing a body flexing. Most modelers have had to watch in horror as their lovingly detailed character folds up like an accordion when it raises its arm or turns its head. Skeletal deformation always tends to collapse around joints under extreme rotations. Moreover, the problem gets worse with every additional degree of freedom. Even simple hinge-joints, such as elbows and knees, are prone to strange effects, but shoulders are the worst, because they rotate on all three axes at once and are very mobile (see Figure 1).

In film production, the traditional skin deformer has largely been supplanted by a muscle system that simulates the behavior of muscles and bones under the skin. This technology, which used to require a squad of dedicated coders and technical delegates, is gradually trickling down to us mere mortals. There's already at least one Max plug-in available (that you can download from www.cgcharacter.com) and a Maya one is on the way. Unfortunately, muscle technology is pretty new. If the standard time lag between offline and real-time deployment of technology holds true, we'll have to wait a few years for a workable real-time muscle system. In the meantime, we're stuck with the same old skeletal engines, so we've got to make the best of them. This month we're going to look at a simple bone-based strategy for dealing with crummy deformations in real-time skeletal systems.

The Incredible Shrinking Joint

The logic behind using extra bones to correct bad deformations is pretty straightforward. Ordinarily, joints collapse because of the way skeletal systems manipulate the vertices in a mesh. For each bone influencing a given vertex, the skeleton says, "If you are attached to this bone only, you'd go there." The system then collects all of the positions possible for the vertex and finds their mathematical center using a weighted average, which serves as the final position. The problem is, this process works only for positions (see Figure 2). Instead of saying, "One of your bones rotated 90 degrees and the other 0, so you rotate 45," the system says, "Go halfway between where you were and where you would be on the bone that rotates 90 degrees." The depressing result of this is all too familiar. If you're working on a cinematic or a vertex-based real-time engine that doesn't use skeletons, you can probably fix the problem with deformers or joint-driven freeform deformation lattices. In a real-time skeletal engine, which means most modern game engines, you're out of luck.

Enter (to a blaze of heroic fanfare) our hero—the fix-up bone!

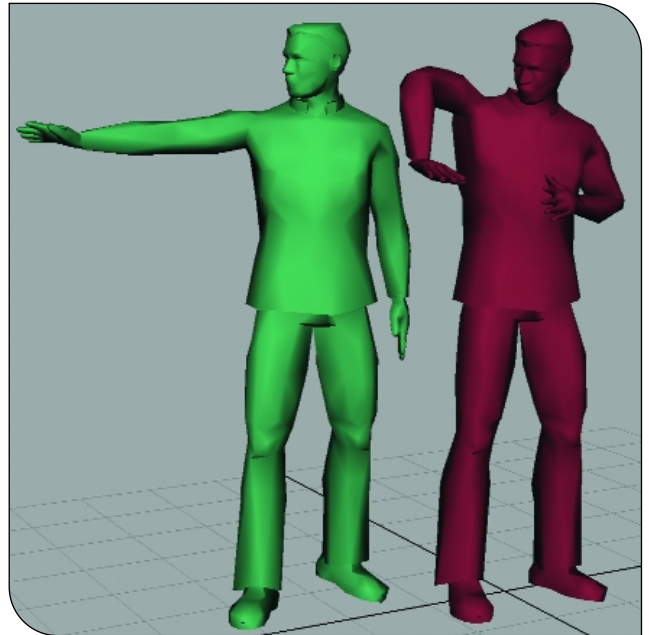


FIGURE 1. Shoulders are the Achilles' heel of the character animator.

A fix-up is an ordinary bone, located in the same place as the problem joint. The fix-up substitutes a correct rotational interpolation for an incorrect linear one by turning some fraction (usually half) of the original joint's rotation (see Figure 3). The partially rotated bone provides a more accurate target for vertex interpolations. Thus, when you weight vertices to a fix-up bone, they rotate around the joint rather than passing through it. Vertices partially weighted to a fix-up will still show the same collapsing interpolation we know and hate, but the effect is reduced, because the fix-up halves the amount of error. In a really complex model you may want more than one fix-up at a particular joint to minimize the interpolation errors even further.

While it's not a substitute for a full cinematic toolbox of deformers and flexors, the fix-up bone has one great strength: simplicity. As far as your engine is concerned, the fix-up bone is just another bone—no special code, no extra art, no additional performance cost. On platforms with severe transform limita-



STEVE THEODORE | Steve started animating on a text-only mainframe renderer and then moved on to work on games such as *HALF-LIFE* and *COUNTER-STRIKE*. He can be reached at steve@theodox.com.

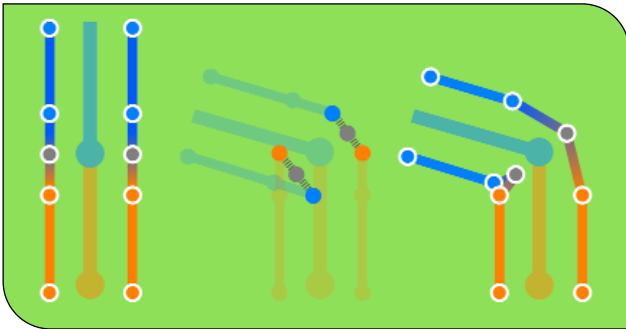


FIGURE 2. This two-bone structure shows why joints tend to collapse. Blue and orange vertices are weighted completely to their respective bones. The brown vertices are weighted 50 percent to each bone. When the blue joint bends, the position of the brown vertices is calculated by averaging the position they would have had if they were attached solely to orange or blue bones. This produces the collapsing and pinching typical of most skinning systems.

tions, you may need to think about the cost of adding lots of fix-ups, but few modern engines are transform-bound. You can add fix-ups to your model without even talking to a programmer or a producer. In the game production world, this makes them the ideal art tool.

Building Strong Bones

The only real trick to building fix-up bones is understanding how they relate to the animation skeleton. Ordinarily, we can assume that the clavicle bone is the parent of the upper arm, the upper arm of the lower, and so on; the links connecting most bones look and behave more or less like physical bones. In addition to the physical arrangement of the bones, this arrangement represents the serial arrangement of the forward kinematics (FK) hierarchy: moving the upper arm moves the lower arm, moving the lower arm moves the hand, and so on. As long as you think in FK terms, the animation skeleton works like a real one. Fix-ups, on the other hand, don't fit neatly into this structure. Physically, the fix-ups need to sit exactly on or very close to the bones they are assisting. In the bone hierarchy, though, they are generally going to be siblings of the bones they help.

For example, consider an arm made up of two regular bones: bicep and its child forearm. If you want to add an elbow bone, you'd naturally assume that elbow would be the child of bicep and the parent of forearm. However building the arm this way would create a dependency loop, because elbow is dependent on what forearm does to derive a partial rotation. FK rotation of forearm would cause elbow to rotate as well, thus changing the pose you've just set on forearm. Even worse, inverse kinematics (IK) wouldn't be possible at all, since IK can't make sense of a zero-length bone. To resolve the paradox, elbow has to be a sibling of forearm rather than its parent. Not only does this arrangement eliminate the dependency loop, it means that elbow and forearm share the same local coordinate system

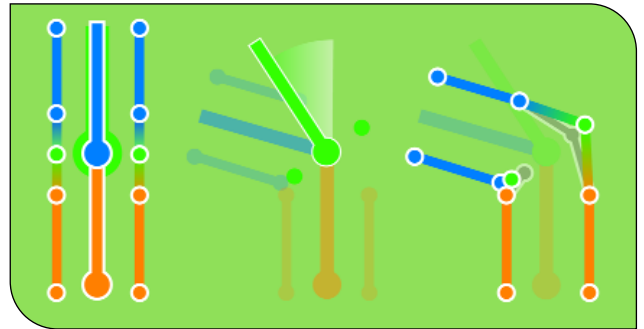


FIGURE 3. Here the green vertices are weighted to a fix-up bone, which rotates at half the rate of the blue bone. The resulting vertex positions preserve much more of the volume at the joint than the original skinned (brown) vertices.

(bicep's transform), which makes interpolation much easier.

In principle, the fix-up doesn't have any physical length, since it only represents rotation around a point. Since it's parallel to the regular skeleton, it won't have any children, and will never need to move or scale. In practice, though, you'll probably want to add a non-animating stub bone to the fix-up to make selection easier and so you can visually check the operation of the fix-up as your skeleton animates. In Maya, adding a stub is quite simple: just append an extra joint to your fix-up bone, positioned as you see fit, by manually parenting an extra joint object to the fix-up joint. Max, on the other hand, has a fetish for aligning bones with their children and will want to forcibly align the fix-up so that its local X-axis points directly at the stub. Once the fix-up is active, you may find that the local X-axis points in an inconvenient direction (it'll usually be partially hidden inside the bone that the fix-up is assisting). For this reason, it's usually easier to just use a different kind of object (for example, a box primitive) as the fix-up bone. You can then shape it as you like with poly-modeling tools. This won't have any effect on its behavior. Remember to turn the stub's renderable property off if you don't want it messing up test renders.

Driving Mr. Fix-Up

Since the basic job of a fix-up is to rotate by a fraction of another bone's rotation, the process doesn't demand a lot of expression-writing wizardry or 3D math. If you've completed all the animation tutorials in your package, you should be able to design fix-ups. Because the fix-up's task is so straightforward, it's easiest to skip the complexities of expression writing and drive the fix-up using orientation constraints. Not only are constraints easier to create and maintain than expressions, their operation and implementation are fairly similar for most art packages. In this respect, constraints are emphatically different from expressions. (Using constraints also simplifies your job if, say, you need to write a column describing your techniques for a magazine.) But if you're more comfortable with expressions, driven keys, reactors, or other advanced control techniques, you can certainly use them to drive fix-ups as well.

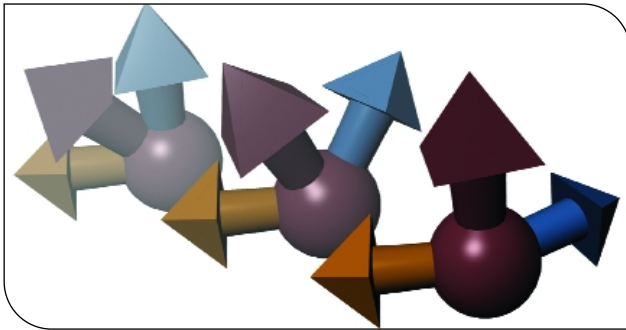


FIGURE 4. An orient constraint, weighted equally to a moving bone (blue) and a stationary rest target (orange), effectively halves the rotation of the moving bone.

Setting up the constraints is quite simple. You begin by constraining the fix-up bone to the bone whose rotation you need to reproduce. To return to our earlier example, the elbow fix-up would be orient-constrained to follow forearm. In order to halve the rotation, you simply need to add a dummy target object aligned to the rest orientation of the bone you're trying to assist. If you weight the constraint equally between the dummy and the real target (see Figure 4), the result is a halved rotation. The dummy should be the third sibling of the fix-up and the target. In our example, it is a child of the bicep. For most applications, that's pretty much all there is to it.

If you need to fine-tune the behavior of the skin, you can adjust the response of the fix-up by manipulating the weights in the orient constraint. Joints under heavy clothing, for example, may want fix-ups that rotate at one third, rather than half the rate of the moving limb. For joints with very pronounced underlying anatomy, such as the patella of the knee, you may want to simulate the sliding of skin over bone by slightly moving the fix-up as well as rotating it. You may even want to tweak the final behavior by hand-animating the dummy constraint object, although this is really overkill in most applications.

There are a few details to keep in mind. If the dummy rest target and the real target are too close (less than 180 degrees apart), odd things may happen, because the constraint may not know how to properly interpolate between them (see Figure 5). Fortunately, this doesn't happen often, since few joints have a range of motion greater than 150 degrees. The most common cause of problems is building the skeleton at one extreme of a very large range of motion: for example, an arm built hanging straight down from the shoulder, which is then animated over the character's head. In such a case, you may need to reposition the dummy to the middle of that shoulder's range of motion so that the constraint doesn't approach the 180-degree limit. Changing the orientation of the dummy will also change to the orientation of the fix-up. Luckily, you only need to care about the relative behavior.

Wrapping Up

Adding fix-ups to your skin deformer is no different from adding any other bone. The one rule you need to respect is

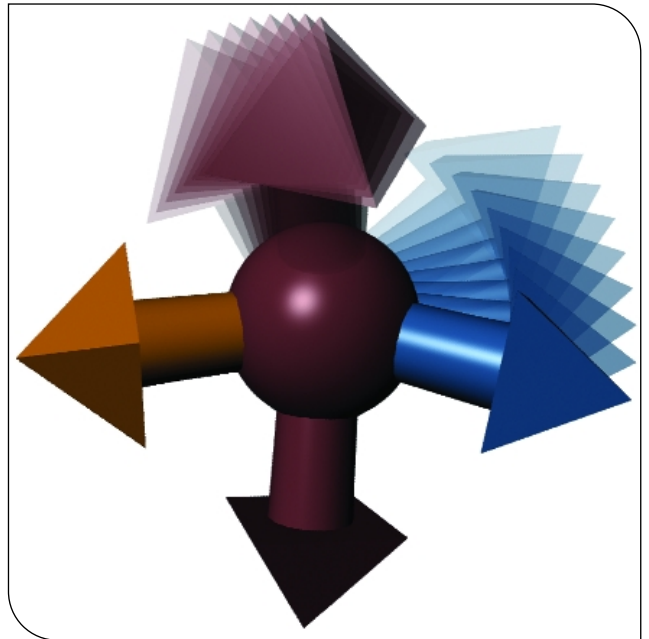


FIGURE 5. Orient constraints can flip suddenly if the angle between the rest and moving targets approaches 180 degrees.

that the fix-up should be active and working before you include it in your skinning deformer. The vertices you bind to the fix-up will be bound to the pose of the fix-up as it is when you include it in the skin deformer. So you need to be sure that, at skinning time, the fix-up is behaving the way it will during animation—particularly if you have adjusted a constraint target. You will probably have to manually assign the vertices. The fix-ups are usually so small compared to the limbs they assist that envelope-based assignment will take little notice of them. In any case, you may need to experiment a little to find the right weightings. Though tedious, this isn't difficult.

As you can see, building fix-ups is far from difficult. When dealing with simple, repetitive tasks like this, it's always a good idea to write scripts to do the grunt work. Automation is not only a timesaver but also an easy way to reduce the possibility of human error and keep consistent naming conventions. This month's code (available at www.gdmag.com) includes scripts for Maya and Max that set up a simple fix-up on a selected bone.

While the simple fix-up bones described here are hardly the answer to all the limitations of conventional skinning, they're still a significant step forward, especially for low-polygon models. When you're comfortable with fix-ups, you should keep your eyes open for ways to apply the same basic strategy to more difficult skinning problems. For example, joint-based fix-ups like these don't help a lot with the shearing effect that causes biceps to shrink when they twist too much along the main axis. However, a fix-up located in the middle of the bicep, which counter-rotates against the twist of the arm, can be a huge help. Unfortunately, that requires a little more fancy footwork—enough for an article all its own. We'll get there in a few months. 🐉

In Search of the Perfect Foreign Orchestra

My last article for this column (“The Live Orchestra Recording,” December 2002) prompted a number of e-mails seeking advice to ensure successful foreign orchestra sessions. I could easily fill the pages of this magazine with a comprehensive list of dos and don’ts, but the single most important thing is finding the right orchestra for your particular project.

Find the contractor. Each orchestra has a contractor, to serve as your key contact person. The role of a foreign contractor is much more diverse than that of his or her American counterpart. In addition to booking the musicians and support staff, the foreign contractor should assist with hotel accommodations, transportation, acquisition of specialty instruments, and the resolution of many preproduction questions. Communication is critical. In the U.S., we take the Internet for granted. However, in many European cities, Internet portals are found only in Internet cafés. It is not uncommon to wait multiple days for e-mail responses. A contractor that has reliable daily access to the Internet should be considered highly.

Research the orchestra. Every contractor will speak highly of his or her orchestra, so you should research the orchestra’s credentials independently. The quality of foreign orchestras tends to be excellent but can vary. As such, the orchestra’s demo CD can be misleading. Similarly, don’t be impressed by an orchestra’s name. I have recorded with the Bratislava Symphony twice and was shocked to find that the second occasion contained different musicians from the first. To eliminate such doubt, Petr Pycha, contractor of the Prague Symphony and the new Filmharmonic, gave me the names of all the principal players and the official orchestra picture before our sessions. Ask to see a list of credits, then contact those people.



Members of the Czech National Orchestra, one of many foreign resources within reach.


It is extremely important to know if the orchestra has experience with your musical style. Many European orchestras have different musical traditions and may not be accustomed to your musical vocabulary. Remember, for many of these orchestras, it was less than 15 years ago that our cultures were segregated by the Iron Curtain. As a result, sometimes the younger players yield better results for music outside that region’s cultural tradition. Finally, consider the city in which you are recording. There is an undeniable correlation between the size of the city and the quality of the orchestra.

Draw an agreement. When formalizing your decision, make sure you include a quality-acceptance clause. Tough decisions are sometimes necessary, and occasionally a player needs to be replaced. This topic can be taboo with some European orchestras. I’ve had orchestras quickly accommodate my request for personnel changes, and I’ve seen tempers flare at the mere suggestion.

Also, make sure you have the same players every day. Rehearsing a cue one

day, only to find different key players the next day, can be disastrous. Include your technical and orchestration requirements in the agreement. Make a list of every instrument, every mute, even what kind of percussion mallets you will need, and include them in the agreement. Make sure the group can properly accommodate click, and video sync if necessary.

Finally, make sure you specify a form of payment; doing business with foreign orchestras can expose you to exchange rate fluctuations. Although the Euro is becoming the standard means of exchange throughout Europe, some groups like the Russian State Symphony Cinema Orchestra prefer the relative stability of the dollar.

Multinational efficiency. Another benefit to good Internet communication is that it allows immediate access to production materials regardless of location. For example, the GC Game Music Symphony Concert yielded about 5,400 pages of music. Using a web portal created by producer Thomas Boecker, the composers’ original scores were uploaded from all over the world. I then downloaded, reviewed, and approved all scores in New York. Once approved, the orchestra’s librarian downloaded the files, and printed, formatted, and compiled part books in Prague. This system allowed us to avoid transporting hefty amounts of music and worrying about the ensuing disaster should that music arrive late. Never consider any other form of snail mail, as customs regulations will inevitably delay your shipment. Finally, if you want the players to understand your intentions, remember that the international language of music is not English. It’s Italian. 



ANDY BRICK | In the past 18 months, Andy has ventured across the Atlantic seven times, recording game music projects with five world-class European orchestras and conducting the first-ever symphonic concert of Western and Japanese game music. You can reach him via www.andybrick.com or andybrick@aol.com.

To Globalize or to Localize ...

This is not just my 24th Better by Design column but also the conclusion of my 24th year in the game industry. There are also 24 time zones around the world. Coincidence? Numerological inevitability? Or possibly, a feeble excuse to write about international game rules?

I've just returned from the fifth Australian Game Developers' Conference. Although the game industry in Australia is still small compared to that of the U.S. or Europe, it is large in proportion to its population, and benefits from optimism, excitement, and an impressive amount of government support rarely available elsewhere.

The keynote speech by John De Margheriti of Microforte focused on Australia's unique position, which leads to a blending of the cultures and perspectives from three major world markets: Asia, North America, and Europe. That's a theory confirmed by my own past experience with Australian developers, who often combine styles from various regions.

But what's behind those styles? True, individual countries and regions have developed distinct game fashions and preferences. North American games are popular throughout much of the world, with the notable exception of Japan. Some Japanese console games achieve international popularity, but there are very few foreign-made console hits on Japan's top 10 list, and some of the top Japanese games never become international hits.

Some British games are worldwide hits, clearly on par with the popular American titles, but there are also some local hits that don't get accepted elsewhere. Many games made in France or Germany don't sell as well when translated into English. And yet, when I was at LucasArts, I was surprised to see that SECRET OF MONKEY ISLAND games, which are full of American puns and ironies that would logically be hard to translate, sold very well in Germany—per capita, more than six times better than its U.S. sales. So it's clearly possible to transcend



HOLLOW delivers a disco eye for the stormtrooper guy.

the language and culture barrier. Could there be game design rules that determine success or failure in individual countries?

One clue comes from a common characteristic in the games of five different German and Austrian developers that I've worked with recently to help them break into the North American market. These European developers generally tend to make games that focus more on details, specifically on opportunities for players to indulge in micromanagement, but less on story or character. While giving the player sole control over the details might help sell a game in German-speaking markets, it might deter buyers elsewhere.

I discovered another clue working with some Japanese experts to translate the insult-swordfighting episode from THE SECRET OF MONKEY ISLAND, where characters trade insults and rejoinders to gain advantage in fights. Although this went over well in America and Germany, our Japanese experts were detectably horrified (even though they

politely veiled their reactions). They were amazed that we thought it was funny to insult each other, and they found two insults involving farmers and ancestors particularly offensive. In Japan there are many social rules and good behaviors that children are taught along with their first words. Perhaps these rules are already part of Japanese game design as well.

A game's regional popularity may be affected by design rules, but it is ultimately determined by the subtle, intuitive cultural preferences. There are also signs suggesting that game developers around the world are gradually assimilating different cultural elements while retaining their own unique perspectives. Zootfly, a new developer in Slovenia, is creating a game called HOLLOW, which is influenced equally by the works of Kafka, Hollywood films, and Slovenia's own emergence from behind the Iron Curtain. The game is set in an alternate-history universe where a dictatorial state dominates most of Central Europe. But the stormtroopers of this regime have adopted the fashion sense of *Saturday Night Fever*, which the Zootfly team calls "Disco Totalitarianism." That's only one of several unique design elements, and I have difficulty imagining a U.S. company creating something similar. But pop culture's global influence is apparent even in Slovenia.

Although there are many groups all over the world managing to create games that succeed in other countries, there are definitely design rules that are endemic to specific countries and cultures that can affect the local popularity of games. If you'd like to enlighten me with a game design rule specific to your country, please e-mail me. ✉



NOAH FALSTEIN | Noah is a 24-year veteran of the game industry. His web site, www.theinspiracy.com, has a description of *The 400 Project*, the basis for these columns. Also at that site is a list of the game design rules collected so far, and tips on how to use them. You can e-mail Noah at noah@theinspiracy.com.

PolySlerp

A fast and accurate polynomial approximation of spherical linear interpolation (Slerp)

While working on our current Xbox title, *B.C.*, we realized the animation system was consuming more CPU time than we wanted. The increasing number of flying, swimming, and walking creatures and the number of blended animations were eating up to 10 percent of the CPU time. We explored different ideas to reduce this computational cost, including: simplifying the characters' skeletons drawn in the distance, limiting the number of blended animations on each character, streaming animations from the hard drive at 30 frames per second, and using approximation methods to interpolate the rotations. This last concept is the subject of this article. Our initial requirements were to find a method that would be both fast and as accurate as possible: we have very large dinosaurs in our game, and small errors on one bone rotation would be noticeable. We also knew we had to accept a trade-off between speed, accuracy, and memory footprint.

We're using unit quaternions to represent the rotation of each bone in the skeleton and therefore use the classical spherical linear interpolation (Slerp) defined by:

$$\text{Slerp}(\mathbf{q}_1, \mathbf{q}_2, t) = s(1-t)\mathbf{q}_1 + s(t)\mathbf{q}_2 \quad (\text{Eq. 1})$$

$$\text{where: } s(t) = \frac{\sin(t\omega)}{\sin\omega}, \omega = \cos^{-1} d \text{ and } d = \mathbf{q}_1 \cdot \mathbf{q}_2$$

Note that $s(0) = 0$ and $s(1) = 1$

Computing a Slerp requires the evaluation of several expensive trigonometric functions: three sines and one arc-cosine. Attempting to approximate Slerp is desirable and has been previously explored. Jonathan Blow made a notable contribution with `quasi_slerp` in his article "Hacking Quaternions" (The Inner Product, March 2002). Unfortunately our March 2002 *Game Developer* evaporated from the office and only recently fell into my possession. Given the pressures of your average project, this was perhaps fortunate as the learning experience has produced some new and exciting techniques that I may not have encountered otherwise.

In this article, we'll first discuss how to substitute the function s by a polynomial. The new family of interpolation functions is hence called `PolySlerpn`, where n is the degree of the polynomial substituting s . Then, as the substitution by an approximation introduces errors, we will study the type and the

THOMAS BUSSE | Thomas is the lead programmer on *B.C.*, a title developed by Intrepid Games, a satellite developer of Lionhead Studios. He can be reached at tbusser@intrepidgames.com.

amount of discrepancies and will elaborate methods to minimize them. We'll use a simple method, found by coincidence, which significantly improves the accuracy. We will also look into different implementation approaches, using tables or not, requiring a preprocessing stage or not, exploiting the Pentium SSE instruction set or not. Finally, we will measure the gain in time compared to the standard **Slerp** implementation and will quantify the average and maximum errors introduced on a large set of randomly generated pairs of quaternions. The next section studies the function s and defines the restricted domain we want to approximate.

Function s and Limit of Our Domain of Approximation

The function s is essentially parametric and the angle ω between the quaternions is the parameter. Before considering any approximation method, we need to see what this function looks like, depending on the value of ω . Figures 1 and 2 show the cases for $\omega = \pi/2$, $\omega = 3/4\pi$ and $\lim \omega \rightarrow 0$.

As \mathbf{q} and $-\mathbf{q}$ represent the same rotation, we only consider here the cases where $\omega \leq \pi/2$, thus in the case $\mathbf{q}_1 \cdot \mathbf{q}_2 < 0$ —equivalent to say $\omega > \pi/2$ —we change the sign of \mathbf{q}_2 , to guarantee that $\omega \leq \pi/2$. With this restriction, it appears intuitively from Figure 1 that a polynomial approximation is likely to be good enough for the task.

General Formulation of the Polynomial Functions

As stated in the introduction, we call **PolySlerp** $_n$ our approximation method of **Slerp**, using a polynomial P_n of degree n ; here is its definition:

$$\text{PolySlerp}_n(\mathbf{q}_1, \mathbf{q}_2, t) = P_n(1-t)\mathbf{q}_1 + P_n(t)\mathbf{q}_2 \quad (\text{Eq. 2})$$

where the general formula of the polynomial P_n is:

$$P_n(t) = p_{n,0} + \sum_{i=1}^n p_{n,i} t^i \quad (\text{Eq. 3})$$

Now, to compute the coefficients $P_{n,0}$ to $P_{n,n}$ and define completely the polynomial, we need $n + 1$ sample points. We choose to take points regularly placed on the t axis from 0 to 1 inclusive. The samples taken on s are $s(i/n)$ where i ranges from 0 to n inclusive. We use the notation $S_{n,i}$ to represent $s(i/n)$. Two simplifications appear immediately:

Because $P_n(0) = s_{n,0} = s(0) = (0)$ (see Equation 1) we are sure that all the $P_{n,0}$ are zero; that is, there is no offset at the origin:

$$P_n(0) = s_{n,0} = 0 \Leftrightarrow p_{n,0} + \sum_{i=1}^n p_{n,i} 0^i = 0 \Leftrightarrow p_{n,0} = 0$$

$P_n(1) = s_{n,n} = s(1) = 1$ means that the sum of $P_{n,1}$ to $P_{n,n}$ is one, and hence, we can compute $n - 1$ coefficients and compute the remaining one as the complement to one of the sum of all the others.



Proof:

$$P_n(1) = s_{n,n} = 1 \Leftrightarrow \sum_{i=1}^n p_{n,i} 1^i = 1 \Leftrightarrow p_{n,1} = 1 - \sum_{i=2}^n p_{n,i}$$

We rewrite our polynomial:

$$P_n(t) = \left(1 - \sum_{i=2}^n p_{n,i}\right)t + \sum_{i=2}^n p_{n,i} t^i \quad (\text{Eq. 4})$$

Table 1 lists, without detailing the intermediary resolution steps, the formulation of the coefficients $P_{n,2}$ to $P_{n,n}$ for P_2, P_3 , and P_4 . It has been obtained by solving the system of equations: $P_n(1/n) = s_{n,1}, P_n(1/n) = s_{n,2} \dots P_n((n-1)/n) = s_{n,n-1}$. Note that $P_1(t) = t$ which is equivalent to say that PolySlerp₁ is the linear interpolation, or Lerp.

Measuring and Minimizing Errors

As we work with unit quaternions and because we substitute s by an approximation, we can focus on the following types of errors introduced:

The error on the length of the resulting quaternion:

$$e_l = |p_s| - 1, \text{ where } p_s = \text{PolySlerp}_n(\mathbf{q}_1, \mathbf{q}_2, t)$$

The angular error with the Slerp:

$$e_a = \cos^{-1}\left(\frac{p_s}{|p_s|} \cdot \text{Slerp}(\mathbf{q}_1, \mathbf{q}_2, t)\right)$$

The Euclidean distance with Slerp:

$$e_d = |p_s - \text{Slerp}(\mathbf{q}_1, \mathbf{q}_2, t)|$$

We use the first two measures (the error on the length e_l and the angular error e_a) to establish a method to reduce the “synthetic” error e_d (error on the Euclidean distance). It is synthetic in the sense that when the two others are 0, e_d is always 0 as well.

Compensating the Error on the Length

To cancel the error on the length is simply a matter of renormalizing the result quaternion, which is to say, dividing the

quaternion by its length. It is, however, worth noting that the normalization can be slightly simplified in our specific case. The length of the PolySlerp_n is: $l = |a\mathbf{q}_1 + b\mathbf{q}_2|$, where $a = P(1-t)$ and $b = P(t)$.

We know that $|\mathbf{q}_1| = |\mathbf{q}_2| = 1$, which implies the length is:

$$l = \sqrt{a^2 + b^2 + 2abd}, \quad d = \mathbf{q}_1 \cdot \mathbf{q}_2 \quad (\text{Eq. 5})$$

Compensating the Angular Error

Minimizing the angular error happens to be more complex, and in fact I couldn’t find a direct analytical method. Let’s examine an indirect approach.

The angular error can be understood as the resultant quaternion being “too late” or “too early” as compared to Slerp. It is therefore possible to apply a transform to t to compensate for the discrepancy, but we’ll consider instead a method that modifies the coefficients $P_{n,i}$ to minimize the error. Using this technique, we realized more accurate results at equivalent computational cost.

Combining Polynomials to Minimize the Error

As I could find no analytical method, I looked for a numerical search method—binary or iterative—to minimize the error. As the values of the $P_{n,i}$ depend on the angle between the quaternions, we could, for instance, try to approximate the $P_{n,i}$ coefficients by polynomials, but, clearly, the number of free variables to search would quickly become prohibitive as n increases, and/or the degree k of the polynomials approximating $P_{n,i}$ gets higher. For an approximation of all the $P_{n,2}$ to $P_{n,n}$ ($n-1$ coefficients) by a polynomial of degree k , we would need $(n-1)(k+1)$ variables to search. We therefore tried another approach.

To explain the principle of this method, let’s first take a simple example. Figure 3 represents the angular error over t generated by a linear interpolation (PolySlerp₁) and by PolySlerp₂ for the case $\omega = \pi/2$. Intuitively, we can notice that the angular error coming from PolySlerp₂ is approximately half of

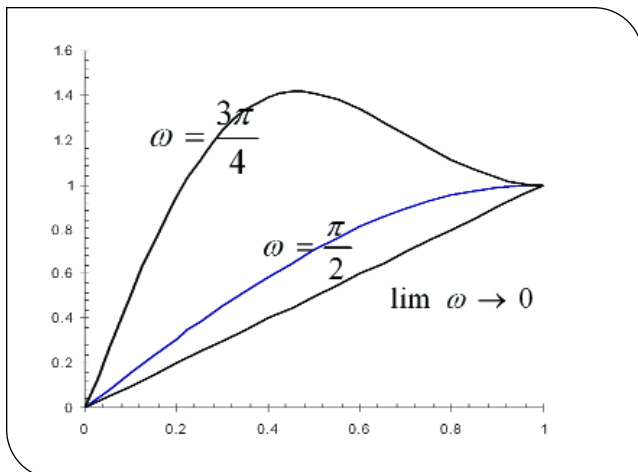


FIGURE 1. $S(t)$ for different values of ω .

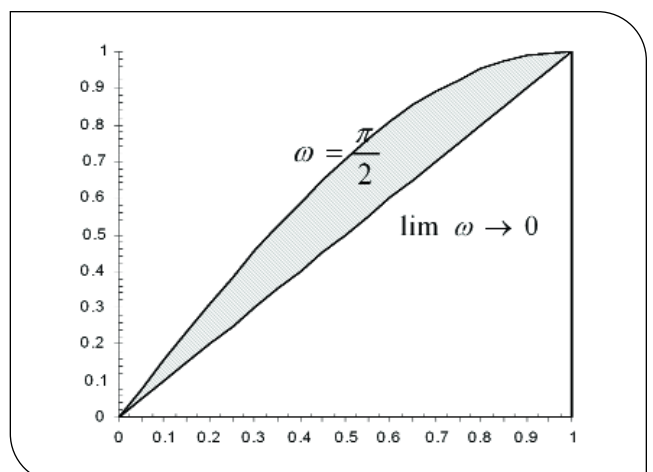


FIGURE 2. The domain covered.

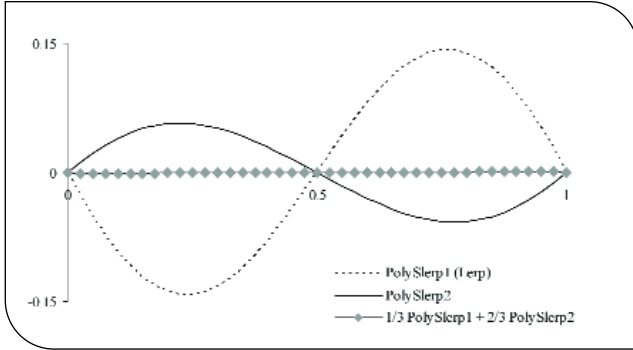


FIGURE 3. Angular error as function of t for $\omega = \pi/2$.

that coming from PolySlerp_1 but negated. We may wonder if it would be enough to replace P_2 by the linear composition $1/3P_1 + 2/3P_2$ to minimize the angular error. It seems, at first glance, to be an interesting option, as the third curve on Figure 2 suggests. It's only an intuitive ratio, for the specific case $\omega = \pi/2$, but it allows us to present our approach: using a linear composition of polynomials of degree 1 to n to minimize the error. It happens that a constant coefficient is good enough for all values of ω . The end composition being of the same degree as P_n , but with different coefficients. We define the new polynomial:

$$Q_n(t) = \sum_{i=1}^n m_{n,i} P_i(t) \quad (\text{Eq. 6})$$

where each $m_{n,i}$ is the coefficient applied on P_i .

This method can work only if we assume the result will be normalized, the simple case $1/3P_1 + 2/3P_2$ shows immediately that PolySlerp_1 substantially alters the length of the quaternion.

We renormalize the quaternion at the end of the interpolation which implies that the sum of $m_{n,i}$ can be any arbitrary value (except 0), and we add a constraint so that this sum is always one. Doing so reduces the number of $m_{n,i}$ to search by one and $m_{n,n}$ is now simply the complement to one of the sum of all the other $m_{n,i}$. Compared to the method suggested earli-

TABLE 1. Formulating $P_{n,2}$ to $P_{n,n}$.

P Coefficients	
P_2	$p_{2,2} = 2 - 4s_{2,1}$ or simplified as $p_{2,2} = 2 - \frac{2\sqrt{2}}{\sqrt{d+1}}$
P_3	$p_{3,2} = -\frac{9}{2}(5s_{3,1} - 4s_{3,2} + 1)$
	$p_{3,3} = \frac{9}{2}(3s_{3,1} - 3s_{3,2} + 1)$
P_4	$p_{4,2} = -\frac{2}{3}(104s_{4,1} - 114s_{4,2} + 56s_{4,3} - 11)$
P_4	$p_{4,3} = \frac{16}{3}(18s_{4,1} - 24s_{4,2} + 14s_{4,3} - 3)$
	$p_{4,3} = -\frac{32}{3}(4s_{4,1} - 6s_{4,2} + 4s_{4,3} - 1)$

er—that builds a polynomial for each $P_{n,i}$ and needs to the search for $(n-1)(k+1)$ parameters—we have only $n-1$ parameters to search. Remark the sum over i of the $q_{n,i}$ is 1.

After simplifying all the linear compositions, we get the new $q_{n,i}$ coefficients shown in Table 2. We just need now a “hill-climbing” style function to estimate the $m_{n,i}$ coefficients by minimizing the error. We not only want to minimize the maximum angular error ($e_{a \max}$) but the average one as well ($e_{a \text{ avg}}$). To do so, we actually minimize $e = e_{a \max} + k_a e_{a \text{ avg}}$ and update k_a during the search process to take into account the actual statistical ratio observed between the average and maximum angular error.

Implementation

We explore different approaches here. For PolySlerp_2 , and its error-compensated version, the evaluation of the coefficient $P_{2,2}$ (to define P_2) and $q_{2,2}$ (to define Q_2), can be done directly from the dot product d and therefore either in real time or stored in a lookup table:

TABLE 2. Simplified $Q_{n,i}$ Coefficients.

Q Coefficients	
Q_2	$q_{2,2} = m_{2,2}(2 - 4s_{2,1})$
Q_3	$q_{3,2} = m_{3,2}(2 - 4s_{2,1}) - \frac{9}{2}m_{3,3}(5s_{3,1} - 4s_{3,2} + 1)$
	$q_{3,3} = \frac{9}{2}m_{3,3}(3s_{3,1} - 3s_{3,2} + 1)$
Q_4	$q_{4,2} = m_{4,2}(2 - 4s_{2,1}) - \frac{9}{2}m_{4,3}(5s_{3,1} - 4s_{3,2} + 1) - \frac{2}{3}m_{4,4}(104s_{4,1} - 114s_{4,2} + 56s_{4,3} - 11)$
	$q_{4,3} = \frac{9}{2}m_{4,3}(3s_{3,1} - 3s_{3,2} + 1) + \frac{16}{3}m_{4,4}(18s_{4,1} - 24s_{4,2} + 14s_{4,3} - 3)$
	$q_{4,4} = -\frac{32}{3}m_{4,4}(4s_{4,1} - 6s_{4,2} + 4s_{4,3} - 1)$

$$p_{2,2} = 2 - \frac{2\sqrt{2}}{\sqrt{d+1}}$$

$$q_{2,2} = m_{2,2} \left(2 - \frac{2\sqrt{2}}{\sqrt{d+1}} \right)$$

Reminder: $m_{2,2}$ is a constant found by the search algorithm.

Another option is to store the dot product d and the coefficient $P_{2,2}$ (or $q_{2,2}$) along with the pair of quaternions. Finally, we can use a look-up table that holds the coefficient $P_{2,2}$ (or $q_{2,2}$). The index in this table is the dot product multiplied by the size of the table and we call this index the “Polynomial Id” or PolyId for short. For B.C. we use a table of 2,048 entries.

For **PolySlerp₃** and **PolySlerp₄**, and their error compensated versions, however, the coefficients of P_3 , Q_3 , P_4 , and Q_4 , require several sample points on s and need for that reason to be stored in lookup tables.

If q_1 and q_2 are known in advance—coming from an animation usually—we attach the PolyId to the pair of quaternions—typically in the exporter code. The dot product is needed for the renormalization (see Equation 5), but we don’t need to calculate it in real time as we can store it in the table. The normalization won’t be completely accurate in that case, but we assume it’s a reasonable trade-off.

When q_1 and q_2 are not known in advance—while blending two skeleton poses for instance—the PolyId has to be calculated

or, as explained earlier, in the case of **PolySlerp₂** it is also possible to compute the coefficient of the polynomial directly in the interpolation code.

The code provided offers, for **PolySlerp₂**, both a direct version and a version using the lookup table. We coded them mainly to be able to compare the error level and the speed of the two versions.

We implemented a standard and a SSE version of all the **PolySlerp_n**. Note that the SSE code uses the “Reciprocal Square Root Estimate” (mnemonic “rsqrtss”) which is much faster but also less accurate than the standard method to compute $1/\sqrt{x}$. This introduces a new source of discrepancies in the interpolation. We compensate it, to a large extent, by having another set of $m_{n,i}$ coefficients. The error minimizing function that computes the $m_{n,i}$ has hence to be compiled in standard or SSE mode.

Statistics: Error and Speed

It’s finally time to see if this method was worth the effort: the evaluation of polynomial of the fourth degree, some renormalization logic and a look-up table that may cause data cache contentions are, after all, good reasons to doubt. We measured the time to interpolate between 200,000 pairs of quaternions and evaluated the average error on the angle (in degrees) and on the length, compared to **Slerp**. We made the test on an Xbox, in standard version and with SSE extensions activated. The standard version gave us the figures shown in Table 3.

The **PolySlerp₄** seems to be quite accurate without error com-

TABLE 3. Interpolation error test results in standard version.

METHOD	VERSION	TIME	ANGULAR	LENGTH X1 000
D3DSlerp	From the standard Xbox library	216	0.00	0.00
Lerp		62	1.100	104.85
Lerp	Err. Compensated	91	"	0.00
PolySlerp4	Standard	81	0.005	0.04
	Preprocessed PolyId	70	"	"
	Preprocessed PolyId + Err. Compensated	97	"	0.03
	Err. Compensated	113	"	0.00
PolySlerp3	Standard	82	0.012	0.58
	Preprocessed PolyId	69	"	"
	Preprocessed PolyId + Err. Compensated	94	0.005	0.03
	Err. Compensated	112	"	0.00
PolySlerp2	Preprocessed PolyId	61	0.471	2.09
	Preprocessed PolyId + Err. Compensated	89	0.007	0.03
	Err. Compensated	107	"	0.00
	Err. Compensated + No lookup table	123	"	"
	Err. Compensated + Coefficients stored in animation key	85	"	"

FOR MORE INFORMATION

Ken Shoemake. "Animating Rotation with Quaternion Curves." *Computer Graphics* Vol. 19, No. 3 (July 1985).

David Eberly. "Rotation Representations and Performance Issues." Magic Software, Inc. (January 2002).

www.magic-software.com/Documentation/RotationIssues.pdf

David Eberly. "Fast Inverse Square Root." Magic Software, Inc. (January 2002). www.magic-software.com/Documentation/FastInverseSqrt.pdf

Jonathan Blow. "Hacking Quaternions." *Game Developer* magazine (March 2002).

pensation, it should be a good candidate both for interpolating between animation keyframes (the "Preprocessed PolyId" version) and a general-purpose replacement of Slerp ("Standard" version). Now, using a lookup table may be a problem: it can be very cache unfriendly and can cause discontinuities between two look-up entries. In that case the "Err. Compensated + No look-up table" version of could be used.

Table 4 shows the results from the SSE implementation. We can see the speed improvement for PolySlerp and measure the effect of using the "Reciprocal Square Root Estimate" assembler operator on the accuracy. It becomes apparent that compensating the errors on PolySlerp₄ is actually detrimental in SSE: it doubles the average error on the length. From this table, it looks like the best candidates are probably the "Preprocessed PolyId" version of PolySlerp₄ for animation key-frame interpola-

tions and the "Err. Compensated + No lookup table" version of PolySlerp₂ for the other cases. At the cost of more memory used for the animation data, the "Err. Compensated + Coefficients stored in animation key" could be used.

The Accuracy Trade-Off

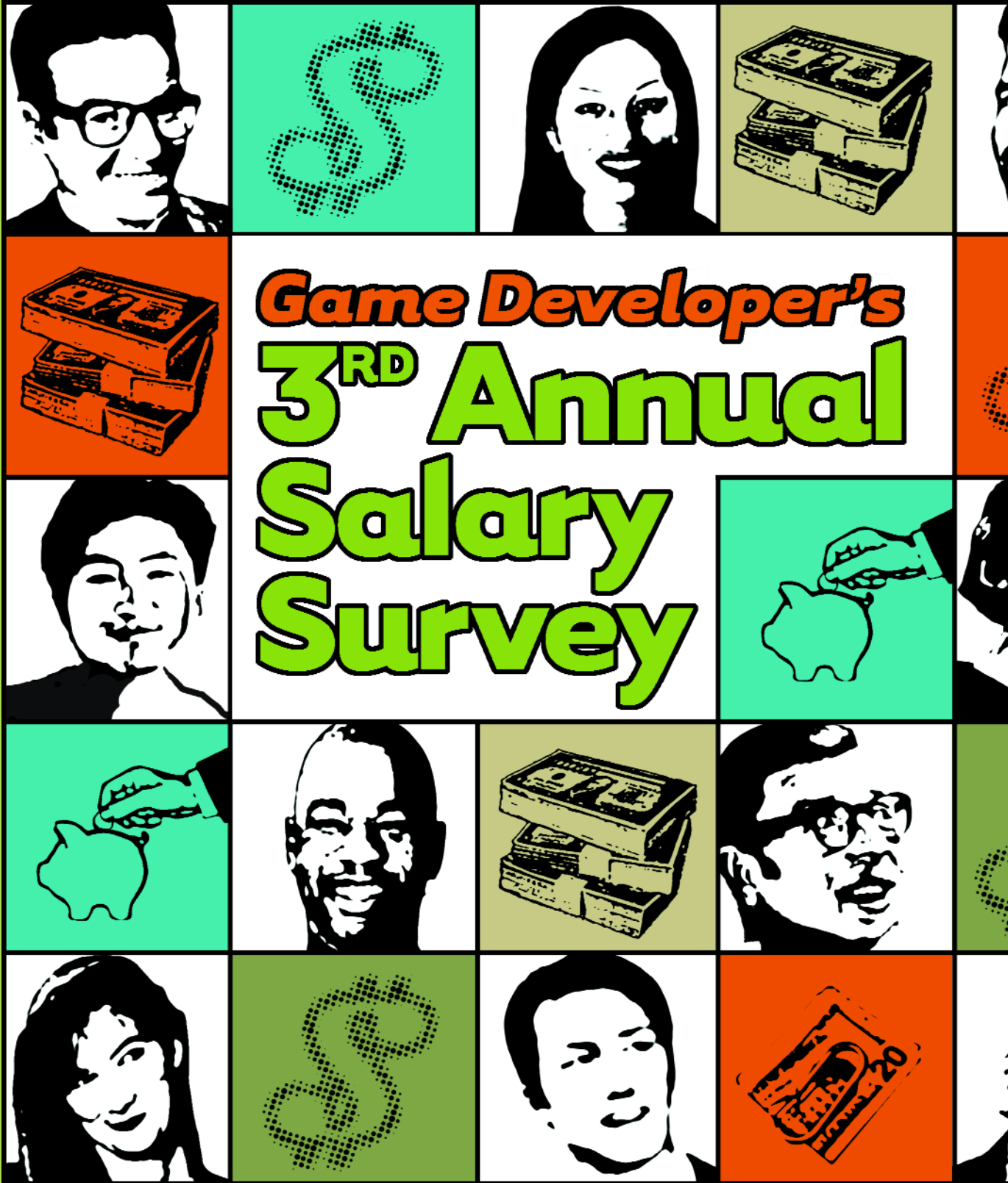
Improving animation techniques to create richer content is a constant concern in our field. Whether it is a matter of blending more animations to produce more lifelike behaviors, building more complex character skeletons to improve realism, or having more actors on the screen to create more interesting worlds, the cost of computation increases dramatically. Having a fast approximation method for animation interpolation proves itself as a useful tool so long as the memory and accuracy trade-off is good. In many cases, a Lerp function with some error correction might be accurate enough, but when the accumulation of error becomes very noticeable, a more accurate method is desirable; for that reason, PolySlerp has been useful for us and may be useful for other developers. ☺

ACKNOWLEDGEMENTS

Thanks to Malika Saulnier for years of contribution to systems analysis and algorithms design, and Andrew Vidler and Sam Martin for their constructive comments and their careful reading.

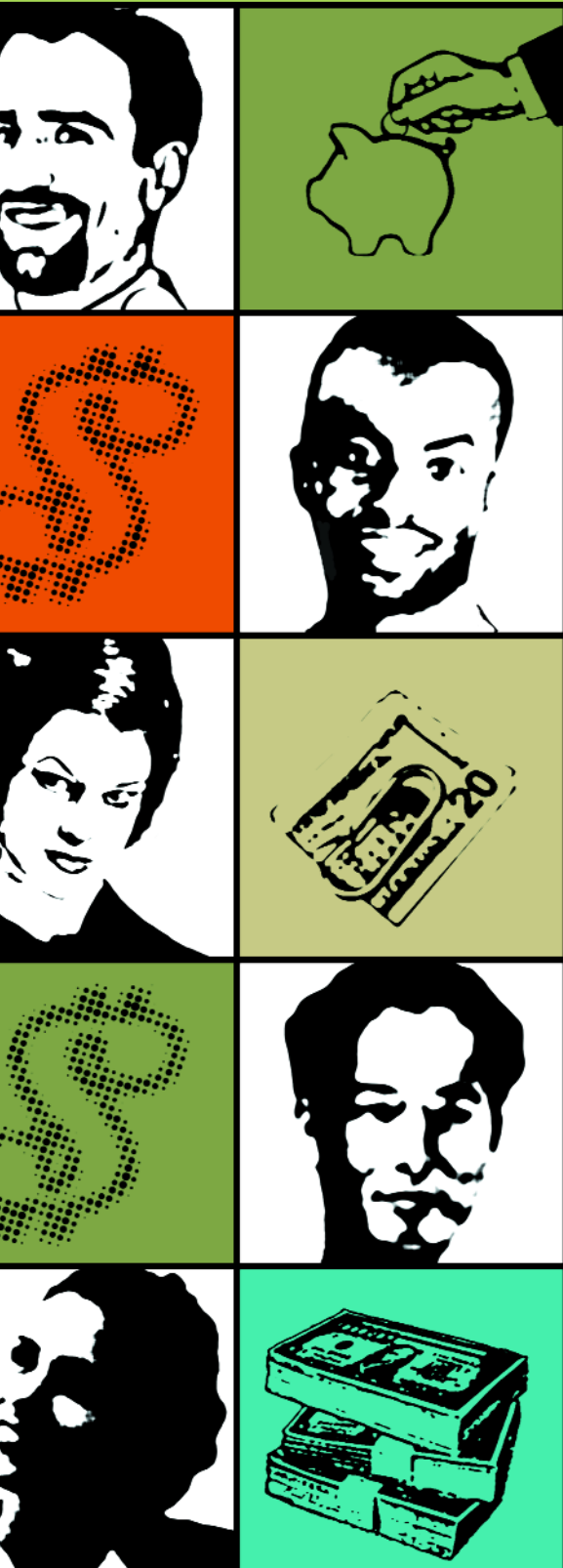
TABLE 4. Interpolation error test results with SSE implementation.

METHOD	VERSION	TIME	ANGULAR	LENGTH X1000
D3DSlerp	From the standard Xbox library	221	0.00	0.00
Lerp		60	1.100	104.85
Lerp	Err. Compensated	70	"	0.09
PolySlerp ₄	Standard	85	0.006	0.04
	Preprocessed PolyId	67	"	"
	Preprocessed PolyId + Err. Compensated	83	"	0.09
	Err. Compensated	106	"	"
PolySlerp ₃	Standard	84	0.013	0.58
	Preprocessed PolyId	68	"	"
	Preprocessed PolyId + Err. Compensated	74	0.006	0.09
	Err. Compensated	93	"	"
PolySlerp ₂	Preprocessed PolyId	55	0.471	2.09
	Preprocessed PolyId + Err. Compensated	71	0.008	0.09
	Normalized	88	"	"
	Err. Compensated + No lookup table	74	"	"
	Err. Compensated + Coefficients stored in animation key	63	"	"



Game Developer's
3RD Annual
Salary
Survey

Illustration by Audrey Welch



This year has been one of true maturation in the game industry, growing pains and all. To paraphrase Calvin Coolidge, today more than ever the business of game development is business. The gulf between game development's garage roots and Wall Street's unrelenting demands is widening. Consolidation has been rampant, bringing big paydays to some and leaving others out in the cold. Uncertainty about the future, both technological with regard to future consoles, and professional with regard to job security, has been a dominant theme.

Still, at the heart of every underpraised triumph and big-budget blockbuster alike are the individual men and women who conjure game magic from the alchemy of programming, art, design, audio, and production support. Now in its third year, Game Developer's annual salary survey examines how such efforts translate into salaries and perks for thousands of U.S. game developers.

With the help of research firm Audience Insights, we sent e-mail invitations to Game Developer magazine subscribers, Game Developers Conference 2003 attendees, and Gamasutra.com members in October 2003, asking them to participate in our annual salary survey, and we received 4,508 unique responses worldwide.

Not all respondents provided sufficient compensation information to be included in the findings. We also excluded cases where the compensation was given at less than \$10,000 or greater than \$300,000, or where there was text entered that did not readily correspond to a compensation figure. We further excluded records missing key demographic and classification information. As this article reports U.S. compensation only, we also eliminated the approximately 1,400 non-U.S. respondents, bringing the total sample reflected in the compensation data presented in the following pages to 2,740.

The sample represented in our salary survey can be projected to the game developer community with a margin of error of plus-or-minus 1.8 percent at the 95 percent confidence level. That means we can say with 95 percent certainty that the aggregate statistics reported would stay consistent, within the margin of error, across the entire population.

Every year the game industry garners more attention from fans and speculators alike. Analysts are no longer projecting the gangbusters growth rates of the past few years, but many outside the industry, from film and music especially, are looking for ways to leverage its cross-media moneymaking potential. Within the industry, some are experimenting with more Hollywood-like permutations of the game business model, including the creation of modular, discipline-centric teams of programmers, artists, or designers available for contract. How future evolution of the game business will affect the balance of power in the industry, and the compensation for developers, remains to be seen.

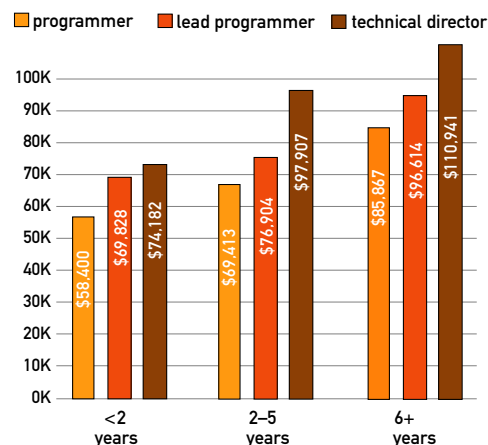


Programming

In the midst of rampant consolidation and talent-shifting in the game industry, programmers continue to enjoy high salaries relative to other development disciplines, whether they work on development tools, gameplay, animation, graphics, physics, networking, AI, or hardware engineering. But as the next generation of consoles looms, subtle shifts in the employment market are already taking place as studios cast an eye to who will carry them smoothly through the transition. Once again the existing talent pool will face an “evolve or die” prospect with new technology.

Valuable assets in programmers in addition to core technical proficiency are flexibility, an ability to see the “big picture” on a development project, and an understanding of how the business of game development affects decision-making on a project. These qualities help differentiate the top tier of technical talent that is always in demand. Battle-tested leads and technical directors are also extremely valuable, but scant availability of such positions limits the advancement prospects of many rank-and-file game programmers.

Programming salaries per years of experience and position



Art and Animation

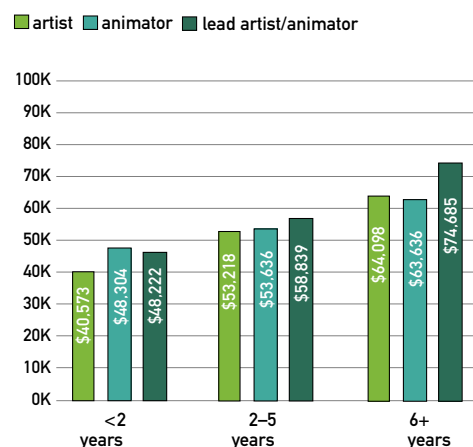
Specialization is more than ever the name of the art game. Unlike programming positions, which can often be difficult for employers to fill, a single artist opening can elicit hundreds of applications. Relative to programmers, artists’ salaries reflect the opposite extreme of a gulf between demand and supply.

The driving force in the artist market, whether for painters, modelers, or animators, has always been raw talent. Those artists and animators who can push the creative envelope while still

respecting technical parameters are most prized. As more and more artists and animators migrate to games from Hollywood, this crop of talent must come up to speed on the technical limitations a game project will place on their genius.

While art team size may fluctuate during the course of a project, most games still get by with one lead artist, or a lead artist and a lead animator. Artists with management expertise will surely grow in demand in the next generation as content-creation needs escalate.

Art and animation salaries per years of experience and position



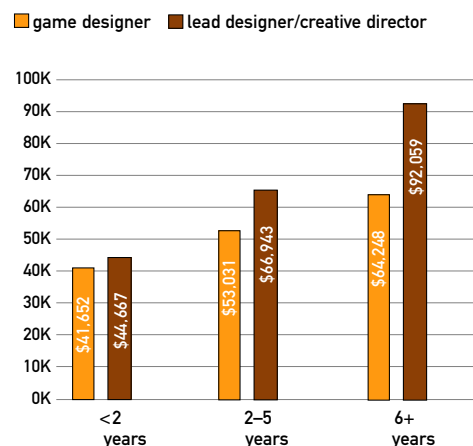
Game Design

Game design is an extremely competitive field to enter, and entry-level salaries reflect this fact. However, designers with a few blockbuster titles under their belt will find their stock rise quickly; there is a big pay gap between rookie designers and more experienced designers and leads.

In our survey, the designation of “game designer” covered game design-

ers, level designers, and writers. Writing is a hot area of design right now, receiving more attention in game budgets as consumer expectations rise for production values in games. Lead designers and creative directors generally manage others who are implementing gameplay decisions, leads governing a single title and creative directors a franchise or portfolio of titles.

Game design salaries per years of experience and position



The Employment Picture: Feast and Famine

While the overall employment picture in the U.S. improved slightly toward the end of 2003, the game industry was a sea of corporate consolidation broken by waves of layoffs, shutdowns, and very early strategic positioning for the next generation. With game production costs rising, "companies are really looking to bring on fewer and better talent," says Mark Alzahov, senior recruiter, R&D, for Vivendi Universal Games. Still, the question of whether it's an employer's or a candidate's market remains complicated, depending on what each party has to offer the other.

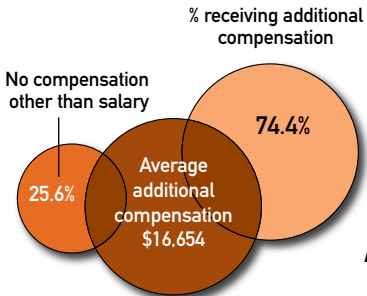
"All positions are highly competitive, and none of our clients wants to settle for less than the best-qualified candidate," says game industry recruiter Mary Margaret Walker, president of Mary-Margaret.com Recruiting and Business Services. On the other hand, "it is equally true that our candidates are not desperate, and expect a lot from a potential future employer."

So what puts a candidate in the most-qualified bracket? Understanding the business of game production with a big-picture perspective on a project is a big advantage. "Everyone wants talent that can understand a production schedule, people that are able to stick to a common goal, from programming to art to design," says Alzahov. Now that teamwork and flexibility are key assets, some companies' layoffs are opportunistic, according to Jill Zinner, president of game recruiter Premier Search. These layoffs might target people who have a lot of experience in the industry but aren't willing or able to adapt to new technologies and production models. These castaways are then having a tougher time finding new homes as the game business matures, according to Zinner. "They're going into other industries, business and edutainment industries. A lot are going into cell phones and handhelds."

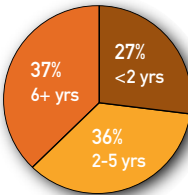
And what impact is the bumper crop of students from the growing number of game-studies specialty schools having on the market for entry-level talent? "The bulk of that impact is a few years away," says Zinner. "The general trend from employers is that they don't even want to interview these people unless they have a college degree they had before they even entered [the game-studies] school." And while a lucky few do get hired straight out of such programs, Walker is "concerned that the programs are giving [students] false hope on their ability to find a job after successful completion of the program."

As the game industry continues to mature in the next few years, the asset of adaptability and ability to mentor will serve those who remain in the industry well, as new people come in from schools and related industries, such as effects and animation. True maturity, according to Zinner, "means not being threatened by new people coming in."

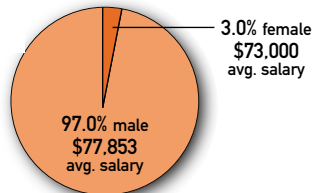
All programmers



Years experience in the industry

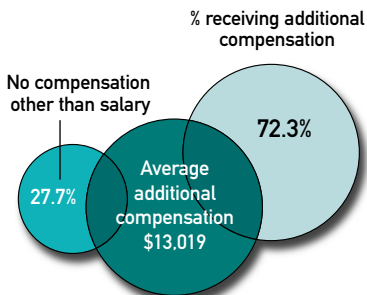


Average salary by gender

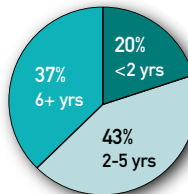


Highest salary
\$300,000

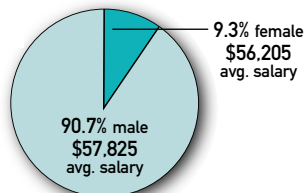
All artists and animators



Years experience in the industry

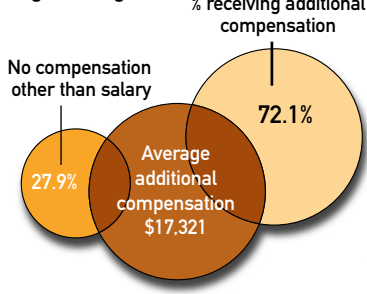


Average salary by gender

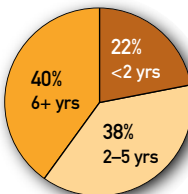


Highest salary
\$156,000

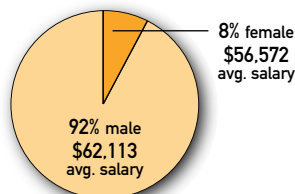
All game designers



Years experience in the industry



Average salary by gender



Highest salary
\$275,000



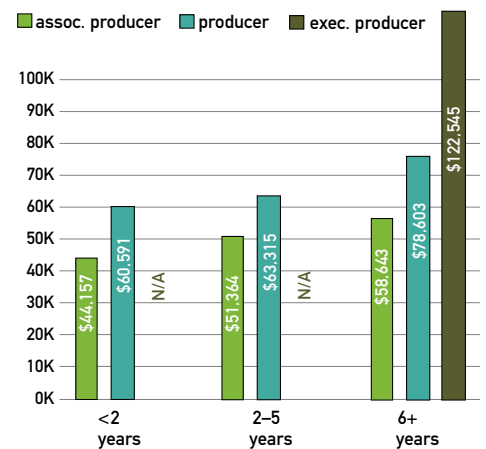
Production

Game production has undergone maturation along with the rest of the industry in the past few years. Whether working internally or externally, producers are an essential interface between the development team and the business of making games. Charged primarily with keeping a project on schedule and on budget, producers' proximity to the bottom line is reflected in higher salaries than many of the creative disciplines of development. Experienced executive producers, whose responsibilities would include management and development of a

franchise comprising multiple titles, commanded the highest salary average for disciplines reported by years of experience in our survey.

Defining and securing top production talent can be a challenge for studios. Schools and universities don't focus their educational programs on production, and the knowledge and experience needed are hard to find in those not already in the game industry. Many producers still work their way up the ladder through design and quality assurance.

Production salaries per years of experience and position



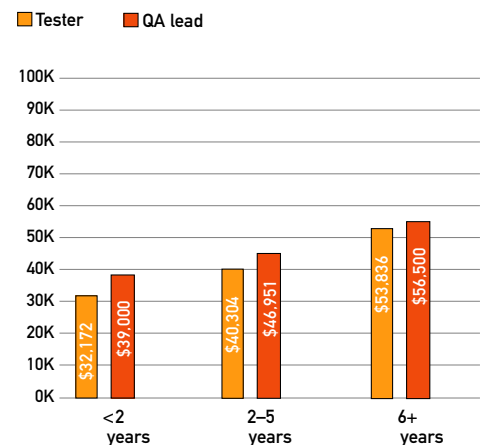
Quality Assurance

This year is the first that we've included QA salaries in our survey results, a reflection of both the role the test department plays as proving grounds for future development talent and the increasing significance of the QA function in meeting high consumer expectations and minimizing returns and support costs. No longer confined to the production domain of bug-hunting, testers are expanding into more significant territory of usability and focus-

group testing to help ensure higher customer expectations are met.

A scant 14 percent of our survey respondents reported being in QA more than six years, the smallest proportion at this experience level by far of any discipline. On one hand this figure underscores QA's role as a springboard for other development careers, while on the other hand it points to a dearth of substantial experience in this increasingly vital function.

QA salaries per years of experience and position



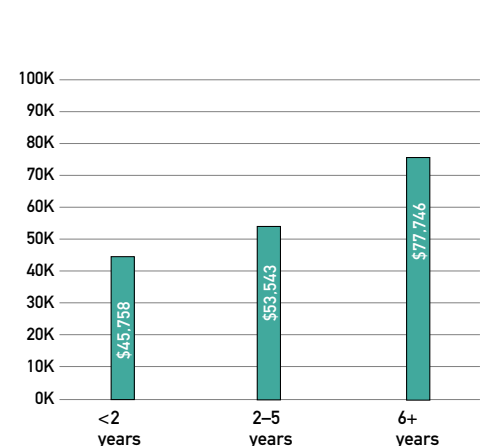
Audio

The current generation of consoles have given the audio community some of what they've been asking for for years: processor time, some storage space, and most of all, respect. The skyrocketing popularity of home theater has quickly catapulted game audio delivery from tin-can PC speakers rattling on a desktop to digital surround inundating players' living rooms. Dolby and DTS technologies are now big selling points for games, and even THX began offering

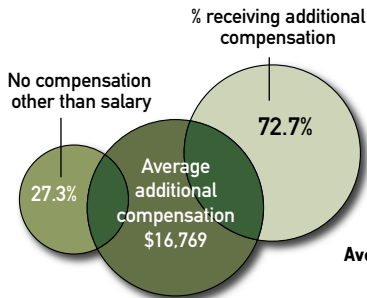
audio certification for games this year.

Industry consolidation has enabled sizeable audio departments to be established at some of the larger studios, but much of the game audio workforce remains in gun-for-hire form. It's a fiercely competitive business, but half our survey respondents have been at it for six or more years, the highest percentage of any development discipline. Obviously there is some payoff for persistence in the audio game.

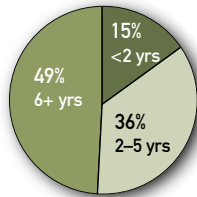
Audio salaries per years of experience and position



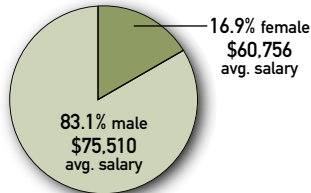
All production



Years experience in the industry



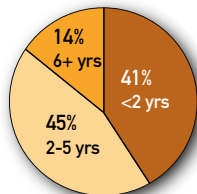
Average salary by gender



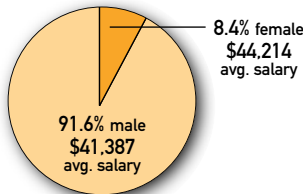
Highest salary
\$240,000

All QA

Years experience in the industry

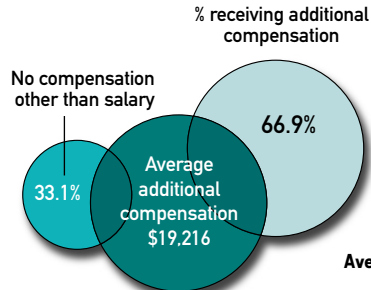


Average salary by gender

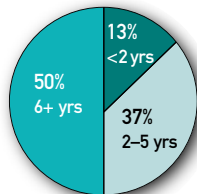


Highest salary
\$120,000

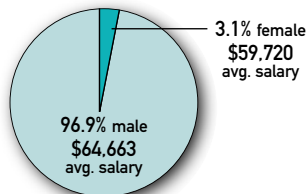
All audio



Years experience in the industry



Average salary by gender



Highest salary
\$250,000

General Trends

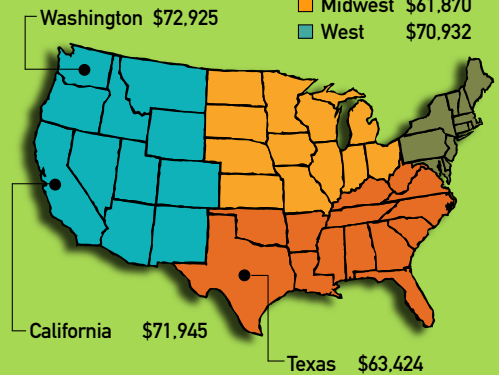
This year's overall salary picture includes QA personnel for the first time, which complicates making direct comparisons to previous years' average salary figures across all respondents; however, some relational data highlights interesting trends.

Women respondents made up 7 percent of the total, a slight increase from prior years. However, their salaries on average continue to lag behind their male counterparts', at 87.4 cents on the dollar, a slight dip from the 89 cents on the dollar reported in our 2002 survey. Conversely, the U.S. Labor Department's Bureau of Labor Statistics reported the national male-female wage gap narrowed slightly in 2002 to 78 cents on the dollar, up from 76 cents in 2000.

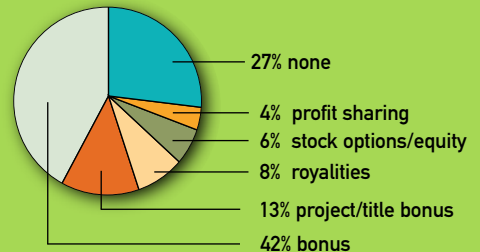
The West Coast continues to be a hotbed of game development, with employer competition driving up salaries relative to other regions. The top five states represented in our survey were, in order: California, Washington, Texas, Illinois, and Massachusetts.

Salary averages by region

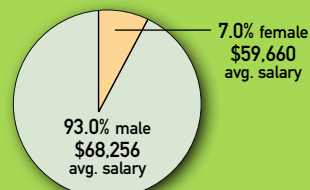
East	\$62,827
South	\$62,447
Midwest	\$61,870
West	\$70,932



Additional compensation: all developers



Overall average salary by gender

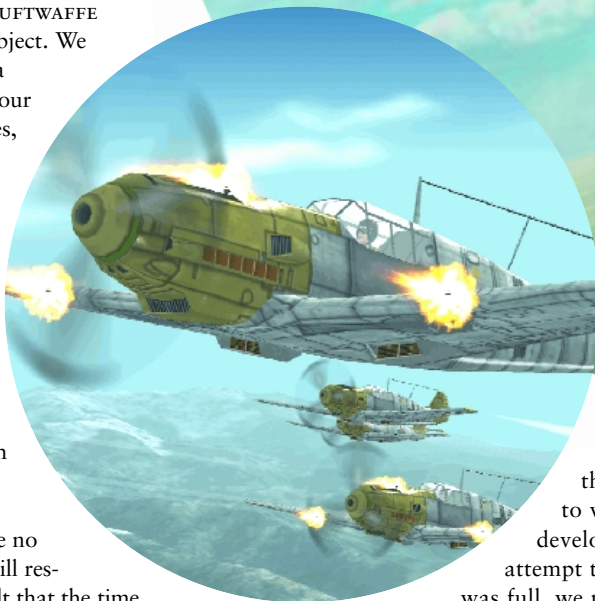


Totally Games' SECRET WEAPONS OVER NORMANDY

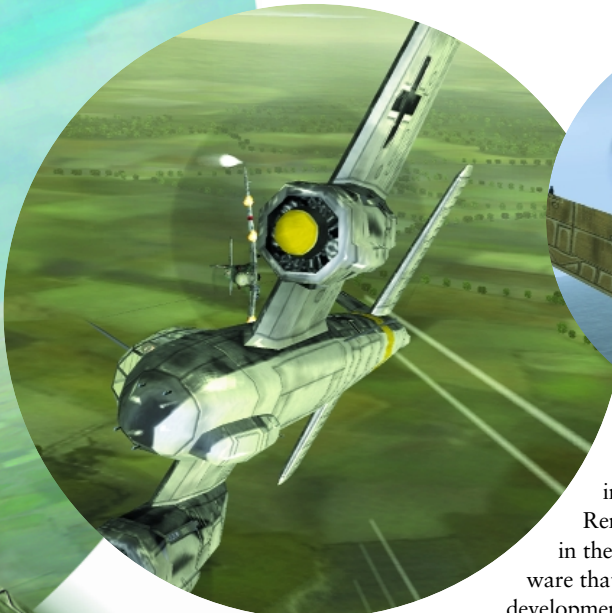
Totally Games has a proud association with the largest conflict the world has ever seen, World War II. *BATTLEHAWKS*, *THEIR FINEST HOUR*, and the acclaimed *SECRET WEAPONS OF THE LUFTWAFFE* represent our previous forays into the subject. We have also been known to dally about in a galaxy far, far away as demonstrated by our *X-WING* and *TIE FIGHTER* series of games, as well as to boldly go where several have gone before with *STAR TREK: BRIDGE COMMANDER*. But when you get down to it, there is just something infinitely more satisfying in dealing with actual historical events.

SECRET WEAPONS OVER NORMANDY (SWON) began with Totally Games' and LucasArts' desire to revisit World War II as a setting. Movies such as *Saving Private Ryan*, television shows such as *Band of Brothers* (based on Stephen Ambrose's superb book), and videogames such as the *MEDAL OF HONOR* series leave no question that over 50 years later, WWII still resonates with many of us. Totally Games felt that the time was right to bring our style of air combat back to the WWII era. Also we were very excited about bringing our style of game to the console audience, a first for both our company and the console market. Once details were lined up, we jumped into development.

This article endeavors to discuss a few of the issues pertinent to cross-platform development. The simultaneous release on the Playstation 2, Xbox, and PC was by far the most unique new challenge SWON presented. With our release date set in



stone, we quickly defined the box within which we had to work. Being typical game developers, we made every attempt to fill said box, and when it was full, we pulled and stretched it to fit a few more things. We then weighed all decisions concerning gameplay, technology features, and other miscellany against the time available. There is always risk when one is too cautious with scope. Go too far and you end up missing dates risking never seeing the finished product on the shelves. Don't go far enough and you end up with a mediocre title. In the majority of our early discussions, as well as many right up to the end, we centered on this issue. In the end, it came down to a gamble.



GAME DATA

PUBLISHER: LucasArts
NUMBER OF FULL TIME DEVELOPERS: 24 full-time developers, 4-5 part time developers as needed, plus the use of LucasArts' sound, voice, localization, and QA departments
CONTRACTORS: art, writing, and voice actors
LENGTH OF DEVELOPMENT: 18 months
RELEASE DATE: November 18, 2003
TARGET PLATFORM: Playstation 2, Xbox, and PC
DEVELOPMENT HARDWARE: Pentium 1-2.4GHz machines with 256-1024MB RAM and various graphics cards
DEVELOPMENT SOFTWARE: Renderware, CRI, 3DS Max, Photoshop, Code Warrior, Microsoft Visual Studio, Various proprietary in-house tools
PROJECT SIZE: Files: 30,284; Code: 246,513 + ~85K actual lines of code



were throwing graphics up on the screen, along with rudimentary physics and AI, but the whole package was bootable on CD, across all platforms. Of course no middle-ware solution will make your game for you. Serious

work was done on the effects system, terrain rendering, physics, and numerous other aspects of Renderware in order to realize our vision of SWON. But in the end it was the initial foundation provided by Renderware that allowed us to proceed with confidence through the development cycle. The faster you can get your game up and running, the faster you can prove your design theories, or identify rough spots that require correcting. Going with middleware bought us the time to handle these issues as they sprang up.

What Went Right

1. Middleware. From the start, given our schedule, we knew that creating an entirely new engine was not an option. We simply could not afford pausing active development while our engineers worked on an engine. Therefore it was middleware to the rescue. After looking into the then current crop of cross-platform middleware solutions, we decided upon Criterion's Renderware. This goes down in the Totally Games history books as "a good call."

Beyond having a renderer and various support systems, the choice to go with the middleware solution also gave us the foundation of an asset chain and pipeline. This is often an overlooked aspect of purchasing your engine off the shelf. This initial leg-up helped us avoid the initial stumbling inherent in many projects. In what seemed like record time, we not only

2. The team. Truly the most important part of the game's development was the team. Our internal development team worked like a well-oiled machine throughout the entire development process. The size of Totally Games was also an asset. Having such a small team broke down the traditional separations and divisions sometimes found at game companies. Programmers would talk openly with designers, artists with programmers. Generally, you cannot expect to create a situation where everyone on the team gets along. You can, however, do everything possible to foster a cooperative environment,

MORGAN W. GRAY | Morgan has been with Totally Games for nearly five years. On SWON, Morgan served as project coordinator. Prior to that he served as a mission builder/level designer on X-WING VS. TIE FIGHTER: BALANCE OF POWER, X-WING ALLIANCE, and STAR TREK: BRIDGE COMMANDER, and as lead game designer for ROBIN HOOD: DEFENDER OF THE CROWN by Cinemaware/Capcom.

where problems, issues, and ideas will be circulated throughout the team and respected. In addition to the internal team, we fostered a similar relationship with our external partners. We relied on our publisher, LucasArts, for sound design, voice support, additional art resources, localization support, and quality assurance. All were treated as members of the collective, which in the end helped us make a cohesive game. In fact, although there were various heated discussions, they were all based on a passion to improve the game. Unproductive arguments or personality clashes were essentially absent from the project. All in all, it was a wonderful environment in which to create a game.

3. Design focus. From the early days of the project, the entire team had a clear idea of the game we were creating. We were setting out to make a seat-of-your-pants, historically inspired WWII air-combat action title. That was the goal. Having everyone on the same page creatively cannot be overlooked. Although there were stumbles when it came to technology, process, and general development, everyone on the team had a clear understanding of what the final game should look and play like. With this in mind, there was no grasping for gameplay or risky experimentation on ideas that would derail us from that goal. With a well-defined box in which to explore, we were able to focus everyone’s creative energies toward defining and enhancing how air combat action should play.

4. Strong preproduction. Being Totally Games’ first strong foray into the console market, and the first time we had ever attempted simultaneous releases across three platforms, we knew that having a solid plan and a realistic schedule would be the only way we could make everything come together. Scheduling was done mostly in Microsoft Project. We laid out the start and stop dates, as well as the various milestones, and quickly had a skeleton of the time frame we had to work with. We encouraged the team leads (tech, art, design) to

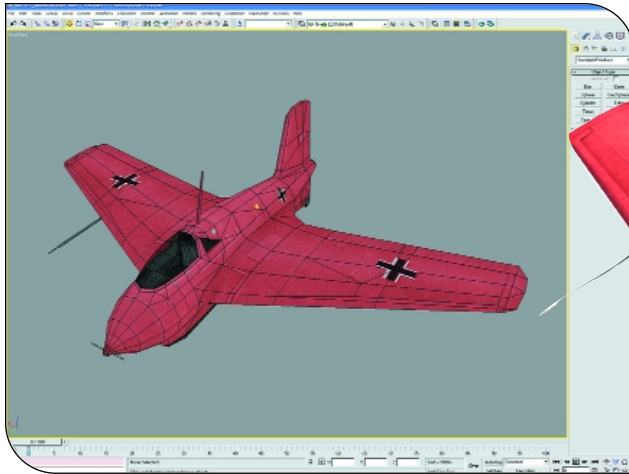
create their own team schedules, which were then incorporated into the master schedule. Allowing the leads to use a method that was comfortable to them was far more effective than trying to force them into using an imposed standardized format. Game development is an odd creature that seems to resist most known types of scheduling, mostly due to its organic nature. In spite of this, our method worked well for us, although it did demand a high level of communication. Having one person with the “grand vision” as well as an eye on the deadlines allowed all of the pieces to come together.

We also used our preproduction period to hammer out the details of our pipelines and tools. By creating various tests we began stressing our systems. During active development they would be put to the true test, and having some time up front to find the obvious issues greatly increased our overall productivity (the exception being our art tools, which we will discuss later on). We also used the time to begin testing various technology pieces, most notably our terrain systems. Finally the lion’s share of the design was documented, which provided a strong road map for the flow of the game. In addition, having the design at a mature state allowed us to begin generating asset lists, AI needs, and other miscellaneous game components that further refined our overall schedule.

5. Build process. Like most game developers, we believed that one of the cornerstones of productive development was nailing down our tool and pipeline chain. Although not as flashy as a model exporter or mission-scripting utility, our build process was the workhorse that allowed all of our hard work to be realized through the glory and majesty of a bootable version. Our process was an “automated” one, in that various configuration and output styles could be selected by the user, and the process begun by hitting what was known as “The Big Button.” Unfortunately I have had to place the word “automated” in quotation marks, since during the course



The simultaneous release of a title on multiple platforms requires a militaristic adherence to timing, milestones, and team cooperation—or else risk never seeing the finished product on store shelves.



ABOVE. An Me 163 Komet on a 3DS Max 6 drawing board.
RIGHT. The secret rocket-propelled fighter reborn to shoot across the skies of Europe.

of the project we never reached a place where anyone on the team could handle making a build. This wasn't because of the tool itself, which worked well, but because we didn't have a "normal build" until near the project's end. Something inevitably was broken in code or in our asset banks that required specialized build knowledge to correct. Our main build keeper held the keys to the kingdom, with a small handful of us capable of tackling most of the day-to-day issues. Like many aspects of game production, this specialized knowledge should be spread throughout the team. Having any one member of the team be the only one able to handle a particular task leaves you vulnerable to losing time if that person falls ill, takes vacation, or is hit by a bus.

At the start of the project our builds would take close to seven hours to complete. This represented almost an entire workday's delay before updates and fixes could be verified. Thanks to the work of several team members, we eventually reduced our overall build times to around two and half hours. This allowed for the rapid creation and distribution of the game to development team members and quality assurance. With several machines dedicated to the build process we could now turn around versions of the game, across all three platforms, along with localized versions, in under six hours. It was this speed that allowed us to hit our dates. So, despite early difficulties and the sometimes arcane knowledge needed, our build process figuratively and practically made everything work in the end.

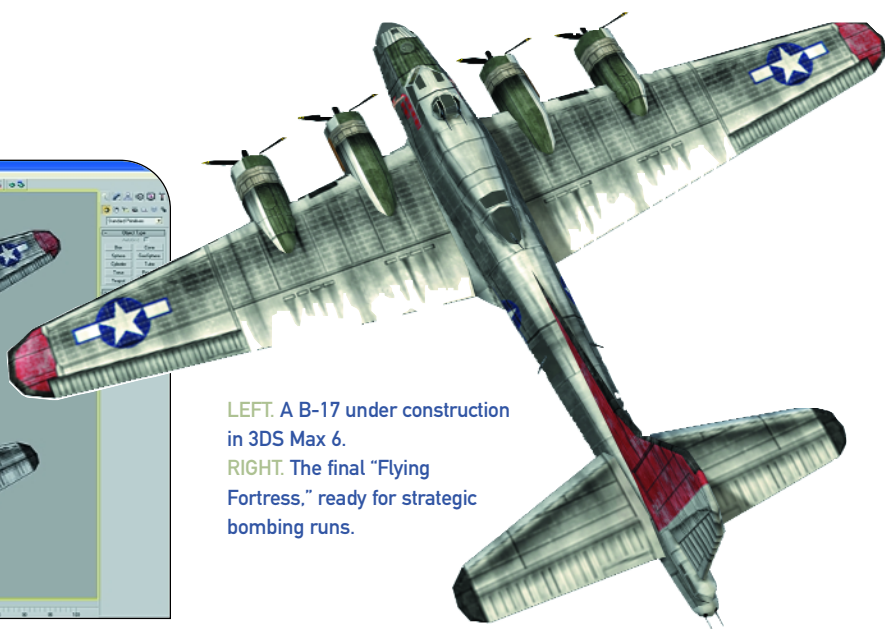
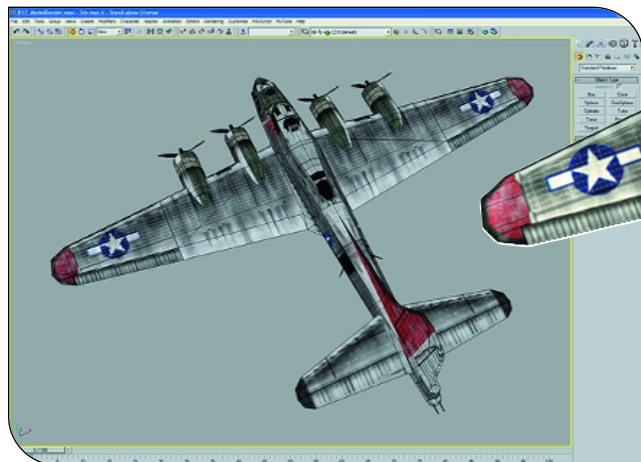
What Went Wrong

1. Platform balance. Understandably, having never released a title across three platforms, there were problems associated with various details that we completely underestimated, if not overlooked. When our QA team came on board, the Playstation 2 version was what they jumped on. We were fortunate in that many on the QA side had previous Playstation 2 testing experience. Quickly we fell into a good rhythm—we would provide new versions of the game on a timely basis, and QA would do their part in shredding them into

individual entries in our bug database. When we had stabilized the Playstation 2 version we began to turn our eye towards the Xbox version. It was then that the problems began. Having focused entirely on the Playstation 2, the Xbox version was lacking many of the features of other versions. This version was also very unstable. On its own, this situation is not unique in game development, but as we were also finalizing the Playstation 2 SKU, we faced a critical manpower shortage.

Bugs were reported on the Xbox and left to languish, as the people responsible for fixing them focused on Playstation 2 issues deemed more critical. This resulted in a slower turnaround, giving QA less time to dig deep into the Xbox version. This situation was slightly alleviated after we submitted the final Playstation 2 version to Sony, and all attention was given to the Xbox. We quickly brought the Xbox and then the PC versions up to a shippable state, but if we had the opportunity to do it all over again we would balance out our version priority, giving each one more time for testing TLC. The fact is simple: more testing time results in more bugs found, and rapidly acting on these bugs results in a more polished and enjoyable final product.

2. Endurance. Many on the team were multiple-title veterans, well accustomed to the frantic pace of crunch time. However, if releasing a title on a single platform is a sprint, releasing on three platforms is a triathlon. We had gone through the alpha, beta, and submission milestones on the Playstation 2 version, but were beginning to lose steam. However, we had to go through it all again for the Xbox version. Due to the short amount of time remaining, the lines between alpha and beta became blurred. Formal schedules were rapidly replaced with daily spreadsheet tracking and an aggressive monitoring of the bug database. Although a slightly haphazard approach to the development, it was the only way that we could stay on top of the sheer volume of things to do. Once again, we finished a component of the project, only to face another: the PC version. As with the other two versions, it was a frantic race to the end, and our assumption that the Xbox and



LEFT: A B-17 under construction in 3DS Max 6.
RIGHT: The final "Flying Fortress," ready for strategic bombing runs.

PC would be very similar paid off. Other than platform-specific issues (mostly hardware compatibility-related), the PC version was finished and sent off for duplication on time.

In retrospect, our focus on Playstation 2 as the primary platform ended up being a double-edged sword. It had allowed us to deal with the challenges of developing on what is generally a more difficult platform, and it further gave us a version that was in a presentable state far earlier than the other platforms. However, this focus forced us to let the other two versions slip too far behind. In a more ideal development process we would have maintained all versions at a near-parallel level of maturity. Although we made some level of progress on all platforms each month, in the end we basically ended up rapidly porting features to the other platforms in order to finish on time. All versions were completed on time and a level of quality achieved, but the price paid by the team was higher than it should have been. As much as possible, future schedules will be created with an eye toward standardizing the development of all versions, taking milestones, demos, and personnel resources into account.

3 Internationalization. With the growing world game market and the escalating costs of game development, getting your game out to as many players as possible is crucial. We had made the typical allowances for translation; no embedded text in graphics, an interface tool that made layout and adjustment of the UI manageable, and a story presentation that was sensitive to international sensibilities (given that our game takes place during WWII, special consideration was given to the German and Japanese markets). The one place we had overlooked was our method of storing text strings, and in turn exporting them into game-ready files.

We used Microsoft Word, with a custom table-based template, for storing and ID-tagging strings. This method worked well for our game designers to enter dialogue and text in a familiar, creativity-friendly format; however, it did not prove adequate for data management. This initially became an issue

during the formation of our voice recording script. Multiple documents (more than 65 individual files) had to be manually cut and pasted into a single comprehensive Microsoft Excel spreadsheet for the recording. It was at this time that we should have created an automated system that took the individual documents and formed them into a single comprehensive database. However, caught in the moment, we made the unwise decision to do this by hand as opposed to taking the time to create a proper tool.

When we came to localize the game, we once again regretted the lack of an automated process for text integration. Our full text string count topped 5,000, certainly not overly large, even by flight game standards. However, we localized the game into five languages. The translated text would come back in language-specific sets. Soon we were faced with five sets of translated documents, each with over 65 documents per set, all requiring integration into our main dataset. Attempts were made to handle this by hand for a brief period of time. Understandably, this process was prone to errors. Text mismatches and version control issues popped up all over the game. Also handling integration by hand was time consuming, which greatly slowed our turnaround time in producing new builds to be tested. As the deadline loomed, we quickly realized it was time to go through the trouble of creating an automated system; good thing, too, since we only had two months remaining in regular development. The hassle turned out to be about five hours of work for our tech lead to prepare an automated string importer/exporter. This turned a process that once took hours (per document set), into a 30-minute miracle that could happen over a lunch break. We could turn around a complete batch of localized builds, ready to boot from DVD, in about four hours. As you can imagine, this greatly increased our QA time, and allowed us to ship clean, neat, and polished versions of the game overseas.

The moral of the story here is, build tools. Time spent upfront will pay off in the end. In the future we are moving toward an online database system that will allow the transla-

tors the ability to modify the text over the web, which will then automatically be integrated into our build process.

4. Art tools. Placing this element in the “what went wrong” list pains me. We made a strong commitment to creating good tools and supporting them through their development. The importance of building good tools in a timely fashion cannot be stressed enough. All too often our art tools reach maturity at the last minute. Our ability to preview our art in a native platform environment (Playstation 2 or Xbox) was hampered for a great deal of the project. In addition, our special effects tools did not fully come online until two months before alpha. This significantly cut down our iteration and refinement time. Although the final versions of the game are beautiful, so much more could have been achieved, specifically when it came to maximizing the potential of each platform, if we had had stronger, more artist-friendly tools from the start of the project. During our preproduction period, we should have created multiple “stress cases” to truly define the scope of the feature sets we would require for active development.

5. Product messaging. The air-combat market is a complicated beast. On the PC side, your game is either a hardcore flight simulation, a space combat game, or an “arcade” shooter. On the console side, there is a bit more latitude with respect to genre definition. Totally Games’ history with the genre was both a blessing and a curse. Tying the game to our SECRET WEAPONS OF THE LUFTWAFFE gave many the false impression that we were setting off to make a hardcore flight simulation. We actively attempted to address this misconception, but somewhere the true message did not reach many of our fans. This created some confusion as to what type of game we were making. Surprisingly, our console fans (new to Totally Games’ titles) accepted and embraced what we put on the shelves, though our PC players seemed disappointed

with the direction we decided to take. Although we understood that there are distinctions between console and PC gamers, we failed to realize that each required very specific handling in terms of delivering the message of the game. One must be aware of the types of genre labels one’s game may receive depending on the platform. In retrospect, we might have done a better job at fostering a community base, and keeping those in the community base well informed of what the game was and how it was progressing.

We live in an information-rich world, and although it is easy to keep one’s head down in order to deal with daily challenges of development, we owe our fans and the game community our best efforts at reaching out and keeping them in the loop. They serve as an invaluable resource to bounce ideas off of, and as a source of inspiration. Time will tell if our current efforts at promoting and supporting the game can break some of its lingering misconceptions. In the end, as the reviews seem to support, we reached our goal of producing a fun, exciting air combat game.

Leaving the Nest

Writing these words a few weeks after SWON’s release, I feel the term “postmortem” is slightly inappropriate. If anything I think “postpartum” might be more applicable. All involved accomplished the difficult task of finishing the game across three platforms, and shipping the game off to the world on time. This was the hardest development cycle of my career, and the hardest, I’m sure, for many others. In the end, I am proud of what we accomplished. Not only did we make our first entry into console gaming, but we did it with a degree of professionalism and savvy that makes us all proud. The reviews have been kind, and all that is left to see is the reaction of our fans. Hopefully, they will enjoy the game as much as we do. Happy flying! ✈



Distinctively arcade-like game features such as dogfighting and an established legacy in simulation can confuse the intended audience.

The Zen of the Professional Artist



Illustration by Alana Machnicki / Three in a Box

Zen Buddhists have a tradition of creating beautiful, intricate artworks in sand called Mandalas. They sometimes dedicate months to perfecting these elaborate arrangements. Then, once the masterpiece is complete, they destroy it.

As an artist and a Westerner, when I first heard of this practice I felt an inexplicable knot in my stomach. Imagine your proudest achievement suddenly gone, with only the memory of its creation left to comfort you. Imagine your résumé and portfolio blank in spite of years of work experience. Now imagine that you caused this, and you did it willingly.

This goes against everything we are taught. “You reap what you sow,” so we spend our lives sowing furiously, in hopes of reaping great rewards in the future. But when do we stop sowing and start reaping? If we stop sowing today, then what happens tomorrow? No time to ponder—up the corporate ladder we scurry, around and around the hamster wheel. It’s the old rat race, and as videogame developers, even we are part of it.

One might ask why Zen monks would destroy such a beautiful piece of art. Well, consider the alternative. How on earth do you preserve a piece of artwork made from sand? The slightest vibration, even the footsteps of a passerby or a conversation in the next room, poses a catastrophic threat. With

every passing moment the sand settles and shifts. It cannot be preserved. It is beyond control. So they do not seek to control or prolong what cannot be, and therein lies their wisdom—wisdom that can be applied to our world, the world of the professional artist.

“Professional artist” is a curious term, somewhat contradictory in nature. If you define the word “professional” in relation to getting paid for a service, then it makes sense, but if you define it through association with the concept of professionalism as it pertains to etiquette, it’s a bit harder to wrap your mind around. Artists rely on their ego and a strong sense of individuality to distinguish themselves. Our creative process is often fueled by the desire for emotional satisfaction, thriving on concepts such as pride, inspiration, creativity, recognition, and congratulation. True professionals, on the other hand, must suppress their ego and forego their personal agenda in

continued on page 55

continued from page 56

order to best serve their purpose and do their job well, learning instead to draw satisfaction from payment, and taking pride in professional conduct in place of emotional gratification.

Of the many challenges that face the professional artist, most are similar in nature to the quandary faced by the Zen monk, originating from a loss of control. We were artists long before we became professional artists. Often during these earlier years, our art served as a form of self-expression, relaxation, and a great source of personal pride and self-worth. It's also one of the few aspects of our life over which we were able to exercise absolute control. Sometimes it's hard, when you have been steering your own ship for so long, to relinquish the helm and resign yourself to rowing while someone else steers. But we must—that is our job.

In years past, this bothered me, causing me at times even to question my decision to make art a career, and as a result, lose one of my favorite pastimes. But recently I have found that there is much joy to be had as a professional artist. It comes not by searching out and discovering more favorable circumstances or by altering your environment, but rather by altering your own perception of events.

**At the moment of creation,
there is no reason to do
less than what is possible.
At completion, however,
there is no reason to
expect more**

Instead of scheming over new means with which to preserve their creations, or vainly cursing their misfortune, Zen monks divorce themselves from countless hours of effort, and relinquish all emotional attachment to their Mandalas once they're complete. They do so in recognition of their powerlessness to control destiny, and the foolishness and futility of such aspiration.

What is remarkable is that, although they crave no emotional payoff for their labor, their personal detachment does not negatively affect the quality of the work. They live in the present moment, and at the moment of creation, there is no reason to do less than what is possible. At completion, however, there is no reason to expect more from a piece of art, for it is finished, static, and dead. The time has come to remove it from thought and focus on the present task, one that has not yet reached maturity, one that may still benefit from the attention. This is the Zen of the professional artist. 🙏

ERIK ASORSON | *Erik is a Southern California-based 3D artist specializing in videogame development. His work has been featured in seven videogames since 1998. He is the author of "Game Artist's Perspective," a bi-weekly column at www.cgworks.com. Contact him at erik@erikasorson.com.*
