



GAME DEVELOPER MAGAZINE

MARCH 2004





# GAME PLAN



LETTER FROM THE EDITOR

## Checkout Time

**W**e were all born into this world naked and helpless, and that's a bit how I felt when I first came to *Game Developer* five years ago as a young, know-nothing editorial assistant. Well, I did know some things, but trust me, they weren't about game development. What I learned from that point forward has changed my life not only by teaching me new things and new ways to think about them, but also by introducing me to a development community teeming with more gifted, thoughtful, and generous people per capita than may exist anywhere else.

Now, after five years at *Game Developer*, my time has come to move on and learn more new things, which I am hoping won't feature an unrelenting battery of monthly production deadlines. It's not been an easy decision at which to arrive, of course. I'll miss turning the brainchildren of clever minds into glossy printed pages that they can show off to Mom. My job has been pleasurable primarily due to the professionalism and generous esteem of the hundreds of writers I've dealt with over the years and the thousands of enthusiastic readers I've heard from and met.

Writing this column each month was not exactly something I looked forward to, but I always tried not to waste the space allotted to me (okay, space which I allotted to myself), and for that I've covered a lot of ground, from business to politics to publishers to matters of craft. So, at the risk of repeating myself here and there, let me touch on some of the major trends I've seen transpire in the game industry in the past five years.

For one thing, most of you have gotten a lot better at your jobs, whether spurred by rising expectations, abject fear, or just practice. While some production gaffes continue to appear in Postmortems with frustrating frequency, many other common development pitfalls have been

incrementally improved upon to where they are now—gasp!—manageable.

Also, you've gotten more confident in your professional identity. Maybe you still can't explain your job to your parents or make them understand that it even is a real job, but you're heroes to more and more young people who want to grow up to be game developers.

On the downside, after five years too many of you are still letting yourselves be marginalized or trivialized in a mainstream context. Keep demanding more professional marketing and PR activities for your games on par with other high-profile entertainment products, not promotional materials that evoke monkeys and typewriters.

Finally, apathy among game developers at regulatory efforts and political witch-hunts is at an all-time yawn. Your creativity and freedom are being threatened. Just because you create fantasy worlds doesn't mean they won't be subject to a heavy dose of reality at some point. Even though I may suffer some initial separation anxiety, rest assured that *Game Developer* remains in fine hands. The editorial team has some very exciting changes coming down the pike, but I don't want to give too much away yet, so stay tuned.

In addition to all the regular columnists who labored month after month under my cruel hand, I'd like to thank the whole advisory board for their selfless contributions, and I especially thank Jeff Lander, Jonathan Blow, Hal Barwood, and Dave Pottinger for being extra generous with insightful feedback and for each having a delightful way of delivering it. This magazine owes much to everyone who's contributed to it in ways big and small over the years.

Jennifer Olsen  
Editor-in-Chief

## Game Developer

www.gdmag.com

CMP Media, 600 Harrison St., 3rd FL, San Francisco, CA 94107 t: 415.947.6000 f: 415.947.6099

### EDITORIAL

#### Editor-in-Chief

Jennifer Olsen [jolsen@gdmag.com](mailto:jolsen@gdmag.com)

#### Managing Editor

Jamil Moledina [jmoledina@gdmag.com](mailto:jmoledina@gdmag.com)

#### Departments Editor

Kenneth Wong [kwong@gdmag.com](mailto:kwong@gdmag.com)

#### Product Review Editor

Peter Sheerin [psheerin@gdmag.com](mailto:psheerin@gdmag.com)

#### Art Director

Audrey Welch [awelch@gdmag.com](mailto:awelch@gdmag.com)

#### Contributing Editors

Jonathan Blow [jblow@gdmag.com](mailto:jblow@gdmag.com)

Noah Falstein [nfalstein@gdmag.com](mailto:nfalstein@gdmag.com)

Steve Theodore [stheodore@gdmag.com](mailto:stheodore@gdmag.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

#### Group Associate Publisher

Michele Sweeney e: [msweeney@cmp.com](mailto:msweeney@cmp.com) t: 415.947.6217

#### Senior Account Manager, Eastern Region & Europe

Afton Thatcher e: [athatcher@cmp.com](mailto:athatcher@cmp.com) t: 404.658-1415

#### Account Manager, Northern California & Midwest

Susan Kirby e: [skirby@cmp.com](mailto:skirby@cmp.com) t: 415.947.6226

#### Account Manager, Western Region & Asia

Craig Perreault e: [cperreault@cmp.com](mailto:cperreault@cmp.com) t: 415.947.6223

#### Account Manager, Target Pavilion, Education, & Recruitment

Aaron Murawski e: [amurawski@cmp.com](mailto:amurawski@cmp.com) t: 415.947.6227

### ADVERTISING PRODUCTION

#### Advertising Production Coordinator Kevin Chanel

Reprints Julie Rapp e: [jarapp@cmp.com](mailto:jarapp@cmp.com) t: 510.985.1954

### 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@balldata.com](mailto:gamedeveloper@balldata.com)

### INTERNATIONAL LICENSING INFORMATION

#### Mario Salinas

e: [msalinas@cmp.com](mailto:msalinas@cmp.com) t: 650.513.4234 f: 650.513.4482

### EDITORIAL FEEDBACK

[editors@gdmag.com](mailto: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 Falettra

#### 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 Friedenber

#### VP, Group Publisher Electronics Paul Miller

#### VP, Group Publisher Enterprise Architecture Group Fritz Nelson

#### VP, Group Publisher Software Development Media Peter Westerman

#### 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

## CMP Game Group



# INDUSTRY WATCH

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

## Nintendo releases scant details on new device.

Nintendo spilled a few specifications of its upcoming handheld Nintendo DS, scheduled to debut at E3 this year. Nintendo president Satoru Iwata's statement that Nintendo DS is "based upon a completely different concept from existing gaming devices" diminished speculations of the new device simply being an enhanced Game Boy Advance. Though Nintendo spokesperson Yasuhiro Minagawa claims that they're not trying to take on PSP, industry watchers consider the Sony PSP a serious competitor to Nintendo's handheld products. Nintendo's announcement describes a device with "two separate 3-inch TFT LCD display panels, separate processors, and semiconductor memory of up to 1GB." No photo or prototype was available at press time.

**Kevin Bachus to make the Phantom materialize.** Infinium Labs appointed Kevin Bachus, the cofounder of both Microsoft



Will games developed for the current Xbox (shown here) be playable on the new Xbox?

Xbox and Capital Entertainment Group, as its president and COO. Bachus's immediate focus is to launch the Phantom Gaming Services, the company's broadband game-rental program. Even though Bachus's arrival lends some legitimacy to the Phantom project, many are still highly skeptical of the elusive game console, whose debut at CES in January 2004 proved to be a non-operational mockup.

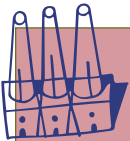
**Avid gains access to Alienbrain.** Avid Technology acquired the Munich-based

NXN Software, which offers asset and production management systems for the entertainment and computer graphics industries. The acquisition gives Avid's customers access to NXN's Alienbrain, supported by Alias, Discreet, Softimage, and other digital content-creation packages. Designed for workflow management, the Alienbrain product line fits into Avid's current strategic motto: "make, manage, and move media."

## Microsoft leaks next-gen Xbox details.

Microsoft leaked to the press some tentative specifications for its next-generation Xbox. Whereas the current Xbox features an 8GB hard disk, the new console will likely include none; users may rely on flash memory to store saved files (as with the Playstation 2). Microsoft was reluctant to say whether the new ATI-powered Xbox would be backward compatible with the current Nvidia-powered Xbox. The unofficial nature of this announcement leaves Microsoft room to reposition the new console with additional components, should competitor Sony introduce the Playstation 3 with far more advanced features. The next Xbox is set to appear in 2005, and Playstation 3 in 2006. *EW*

Send all industry and product release news to [news@gdmag.com](mailto:news@gdmag.com).



## THE TOOLBOX DEVELOPMENT SOFTWARE, HARDWARE, AND OTHER STUFF



**SN Systems delivers PS2 debugger.** SN Systems released Proview Plus for the Playstation 2 console, a postmortem debugger that works with Sony's latest development hardware (DTL-H3010\*LT). The software-based debugger communicates with the console via the FireWire port, leaving both the Ethernet and USB ports available for use by the game. It requires Windows 2000 or XP, a FireWire port, and Sony's development hardware. Proview Plus is available for \$1,000-\$7,200. [www.snsys.com/PlayStation2/ProViewPlus.htm](http://www.snsys.com/PlayStation2/ProViewPlus.htm)

**IDV plants Speedtree RT 1.6.** Interactive Data Visualization released Speedtree RT 1.6, a tree design and simulation editor/middleware. The new version includes enhanced lighting realism and trunk design, along with efficiency improvements. Other key new features

include: Bump-mapping and self-shadow lighting effects, 360-degree billboardboarding for smoother transitions, improved root/branch detail, and new library additions, bringing the total to 80 species. [www.idvinc.com/html/speedtreert.htm](http://www.idvinc.com/html/speedtreert.htm)

**Macromedia launches Director MX 2004.** Macromedia announced Director MX 2004, the latest version of its multimedia development and prototyping tool. New features include support for JAVA script in addition to Director's existing Lingo script, better integration with Flash, support for nonlinear DVD playback, and better playback of QuickTime, Windows Media, Real, and AVI files. Director MX 2004 is available as an upgrade from Director 8.5 or MX for \$399 or standalone for \$1,199. [www.macromedia.com](http://www.macromedia.com)

—Peter Sheerin

## UPCOMING EVENTS CALENDAR

### GAMER TECHNOLOGY CONF.

WESTIN SEATTLE HOTEL  
Seattle, WA.

March 11-12, 2004

Cost: \$895

[www.lawseminars.com/htmls/seminars04/04gamewa/](http://www.lawseminars.com/htmls/seminars04/04gamewa/)

### SOUTH BY SOUTHWEST: MUSIC AND MEDIA

THE AUSTIN CONVENTION CENTER  
Austin, TX.

March 12-21, 2004

Cost: \$225-\$775

[www.sxsw.com](http://www.sxsw.com)



## Borland's JBuilder Mobile Edition

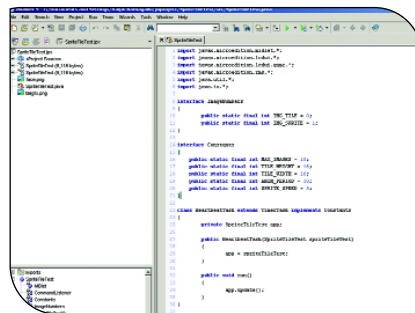
by ralph barbagallo

**M**obile game development is hot. That doesn't mean many people are actually making money in the market, but nonetheless, it's big. Major publishers such as Activision and THQ have stepped up to the plate, along with a host of newcomers, to bring content to carriers worldwide. Sun's J2ME platform is deployed on the most carriers, with thousands of developers producing a seemingly infinite array of games for handsets of all types.

Surprisingly, there isn't a thriving market for J2ME development tools. With Metrowerks' CodeWarrior Wireless Studio languishing without updates, Borland has stepped into the vacuum with an affordable J2ME IDE—JBuilder 9: Mobile Edition. Sporting the familiar JBuilder interface, Borland has put together a great package that's even cheaper than CodeWarrior, for \$399. You may have to look really hard, however, JBuilder Mobile Edition just isn't very widely available.

JBuilder has a long history as a tool for J2SE developers. Therefore, the interface of the IDE should be familiar to those who have used previous versions of JBuilder. To the uninitiated, JBuilder is far less complicated than CodeWarrior's somewhat Byzantine interface, but not as fast or slick as Microsoft's VisualStudio. The GUI is also curiously slow, with redraws and refreshes occasionally being heavily delayed even with 512MB of RAM (Borland suggests a full 1GB of RAM for optimal performance).

Yet, after reading through the brief online documentation, I was up and running



JBuilder's interface is less complicated than CodeWarrior's, but not as fast as VisualStudio's.

in no time. Creating a simple "Hello World" MIDlet was easy as pie. And converting my old Sun Wireless Toolkit projects to JBuilder was just as simple once I got the hang of it. All I had to do was create a new project and start copying files over. In addition to manually copying files and adding them to the project, JBuilder provides a number of wizards to set up default projects and simplify the task of getting started.

Speaking of wizards, JBuilder has a number of unique features that can help when developing more complicated MIDlets. One in particular is the UI designer. Even though J2ME's lcdui package is pretty bare-bones as far as GUIs are concerned, JBuilder allows you to specify the GUI components of a screen and will generate the code and classes automatically. It will also analyze the code and then display a diagram that shows you which screens link to which. This not only helps you visualize your program, but also helps you spot errors in your GUI logic that may make screens inaccessible or other-

wise unusable. This tool really simplifies the task of setting up mundane GUI code and lets you get to the real meat of development (that is, if you actually use J2ME's ugly and inflexible GUI classes).

One of the great innovations of Wireless Studio was the ability to select between multiple JDKs to support each handset's unique API. JBuilder mimics this ability with the JDK Configurator. Here you can point JBuilder at the folder in which the desired API is stored, and it will automatically find the classes, libraries, and emulators. You can then pick the SDK you want to use for this project, not to mention switching it at any time to make builds of your project for specific SDKs.

As for other J2ME-specific features, JBuilder allows for extensive editing of the JAD file. This includes creating your own custom fields and values. In addition, JBuilder provides support for third-party obfuscators, including Retroguard. Also, any third-party emulator can be used and custom command-line parameters can be set for each one. JBuilder also supports ANT, so you can create your own custom scripts for just about any purpose. This comes in handy when having to make handset-specific builds that may require the inclusion or exclusion of certain files for certain handsets (such as art assets), or omitting various classes.

JBuilder's source-level debugger is quick and responsive, not to mention full-featured, with a standard array of breakpoints, watches, and other tools. In comparison, CodeWarrior's debugger is laggy, and even buggy at times where it fails to stop at certain exceptions and has a hard time refreshing variable values. Unfortunately, CodeWarrior's trailblazing on-device debugging feature is absent from

**RALPH BARBAGALLO** | *Ralph runs FLARB ([www.flarb.com](http://www.flarb.com)), a game studio in Southern California specializing in wireless games. He is the author of Wireless Game Development in C/C++ with BREW (Wordware Publishing) and is currently working on a MIDP 2.0 book.*

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

JBuilder. Even though in the case of CodeWarrior it only worked on one or two handsets, it is an excellent feature that Borland should really look into for upcoming versions.

Much like CodeWarrior, JBuilder features in-IDE support for third-party source control programs, including the widely used open-source tool, CVS. Other big-time development features touted in the documentation include UML diagramming, although that's only available in the more expensive Enterprise Edition.

Once again, a J2ME IDE is here with not much to compete against. I heartily recommend this over CodeWarrior Wireless Studio. With JBuilder's snappy interface, a plethora of features, and a very low price, I don't think anyone will be missing Metrowerks from this space. It's unfortunate, as some advanced features such as on-device debugging are still absent from JBuilder. We'll have to

see how much continuing support Borland affords JBuilder Mobile Edition given its scarcity. If you are an existing CodeWarrior user, there may be no compelling reason to switch. However, considering that heavyweights such as Sony Ericsson and Nokia have embraced JBuilder as a prime development tool for their respective handsets, it may be time to reconsider.

### 3DConnexion's SpaceTraveler

by sean wagstaff

If you work in 3D, navigation in space probably occupies far more of your day than you realize. But just as a painter doesn't give much thought to how he positions his brush on the canvas, experienced 3D artists don't really think about moving around in three-dimensional space. Unless you're using an unfamiliar application, say, switching from Maya to 3DS Max, navigation is simply an integral part of what you do and there's not much room for improvement. Or is there?

The \$599 SpaceTraveler, which looks like a volume control knob (complete with a purple LED accent on the buttons around its rim) is designed to make 3D operations faster and more intuitive.

Using the SpaceTraveler is almost immediately familiar. You plug it into your USB port and install the driver software (plug-ins are provided for Maya and Max, and built into MotionBuilder, Cinema 4D, and BodyPaint 3D, but the controller doesn't work with every 3D tool). To use it, you simply push, pull, tilt, and twist the single knob. Your finger movements translate directly into 3D space—*x*, *y*, and *z* rotation and translation, often referred to as six degrees of freedom—in your application. Lift the knob and you move up in *y*, push it forward and you move forward in *z*. Twist the knob and you'll rotate in *y*; tilt it, and you'll pitch forward or back, left or right. The tricky part is learning not to translate on *z* when you pitch on *x*, and not to translate on *y* when you actually mean to roll on *z* (a temporary filter can be turned



The intuitive SpaceTraveler 3D controller.

on that blocks non-dominant movements). But with a few minutes worth of practice to get a feel for it, the SpaceTraveler becomes very natural to use, although it is quite sensitive to even fine movement. However, you'll soon find yourself tumbling a scene around as easily as you would with your standard keyboard and mouse combinations, and rotating a camera is certainly more intuitive than, say, SHIFT-CTRL-ALT-middle-mouse dragging.

Which brings us to the most obvious question about this device: who needs it? If you're already comfortable working in a 3D application, and navigation with the standard key commands and mouse actions has become second nature, why bother with yet another input device? In my experience, many 3D operations, such as architectural modeling, dynamics, and texture manipulations, simply require too much keyboard input to benefit from the SpaceTraveler at all. I need my hands on the keyboard, and mouse, and instant access to pop-ups and marking menus provided by my right mouse button, which just doesn't leave enough hands for a third input device.

On the other hand (literally) when it comes to operations that require one-handed navigation, the SpaceTraveler is a terrific idea. For example, when sculpting an organic model or painting textures on surfaces with a Wacom tablet, you can rotate and tumble the model with one hand, while painting with the other. While doing character animation, the SpaceTraveler can be used as a low-speed motion capture input device that lets you use gestures, rather than explicit rotations, to move a joint, although you'll have to set up your characters to work with this

#### JBUILDER MOBILE ED.



##### STATS

Borland Software Corp.  
Scotts Valley, Calif.  
(831) 431-1000  
[www.borland.com](http://www.borland.com)

##### PRICE

\$399

##### SYSTEM REQUIREMENTS

Intel Pentium II/400MHz or equivalent.  
256MB of main memory (512MB recommended). Windows 2000 or Windows XP. 1.9GB of free hard drive space (full install).

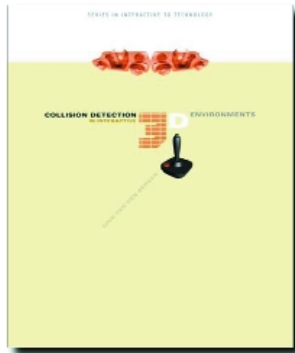
##### PROS

1. Intuitive and streamlined interface.
2. Easily switch between JDKs.
3. Innovative new tools such as the UI Designer.

##### Cons:

1. No on-device debugging.
2. Lack of UML support.
3. A very hard to find product.

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



input. The device's eight buttons can be mapped to common keyboard shortcuts, and the defaults for Maya activate the Hot Box, translate, rotate, and scale commands. However, the buttons are too small with terrible ergonomics, and I still need to use the keyboard for other commands, such as the marking menus.

The SpaceTraveler, as the name implies, is small and portable. Although on-the-road walkthroughs of real-time-3D scenes seem unlikely, I found the SpaceTraveler useful as an accessory to a high-end 3D laptop for bringing work home. My Compaq runs all my 3D applications, but the built-in trackpad is all but useless for 3D navigation, and the keyboard is cramped, with a non-standard layout, which also makes navigation clumsy. The SpaceTraveler really improves the usability of that machine on the go.

I wouldn't recommend the SpaceTraveler to everyone. After all, if you're already comfortable navigating in your predominant 3D application, you probably don't need it. However, if you do a lot of work that requires one-handed navigation, the SpaceTraveler may be a welcome arrival to your world.

★★★★★ | SpaceTraveler  
3DConnexion  
[www.3dconnexion.com](http://www.3dconnexion.com)

*Sean is a freelance 3D artist. You can reach him at [www.wagstaffs.org](http://www.wagstaffs.org).*

## Collision Detection in Interactive 3D Environments by Gino van den Bergen

*reviewed by jeremy jessup*

In *Collision Detection in Interactive 3D Environments*, (Morgan Kaufmann, November 2003), Gino van den Bergen explores the algorithms necessary to determine whether polygonal intersections occur in a real-time interactive simulation. Available for \$59.99, the book spans 277

pages through seven chapters and includes a CD-ROM containing the source code to SOLID 3.5 (Software Library for Interference Detection), a collision-detection library for interactive 3D animation.

After the first chapter's brief introduction, the second chapter details the required concepts of the text. Generally, the collision-detection algorithms presented in the book operate on convex objects. Methods are described to decompose complex shapes into various convex primitives such as spheres, triangles, and boxes. Some consideration is given to collision response, performance optimizations through frame and geometric coherence, and problems arising from floating point error in calculations. The chapter is heavy in mathematics and notation and makes for a slow and sometimes tedious read.

Chapter three introduces algorithms for various types of primitive collisions through four broad categories: spheres, axis-aligned boxes, separating axes, and polygons. Each category contains an algorithm for various primitive combinations. For example, under the sphere category the routines presented are sphere to sphere, ray to sphere, and line segment to sphere. Each algorithm is well described mathematically, then some pseudo-code is provided to illustrate the implementation. However, each category's primitive combination type presents just one algorithm. While other sources for algorithms are well-cited throughout the book, it would have been beneficial to compare multiple collision algorithms based on various scenarios to explore the topic completely. The SOLID library uses the routines chosen and presented in the text.

Chapter four is on convex objects, and Van den Bergen considers both single-shot and incremental algorithms designed to perform several types of proximity queries on polytopes. In particular, each algorithm's computational complexity is provided and references are given for additional detail. The bulk of the chapter is devoted to discussion of

the Gilbert-Johnson-Keerthi (GJK) algorithm, which is used to determine distance and collision of general convex objects. The GJK algorithm is an iterative distance routine but can also be applied to general convex objects.

Chapter five discusses data structures that reduce the scope of collision calculations during run-time. Through a combination of spatial partitioning, model partitioning, and frame coherence (an assumption that motion is generally smooth and changes per frame are small in a given scene), optimizations can be made to reduce overall computational time in calculating pair-wise collisions between the various types of polyhedra. Each section presents several partitioning methods and provides a case study regarding their performance, along with a test bed of complex objects to help highlight the performance differences.

SOLID has been under development for the past seven years, and chapter six provides the goals, an overview, design decisions, and restrictions of the library. In fact, the material presented in the book is implemented as the SOLID library. The provided SOLID source code helps contextualize the algorithms and discussion presented in the text. Finally, the last chapter describes the current limitations of collision detection and considers future research areas where further improvement might occur.

Overall, the book does an excellent job presenting the challenges and necessary considerations when designing a collision detection system but not in a manner that is approachable by everyone. Developers capable of appreciating the mathematics and theory will benefit from van den Bergen's description of his insights and experience. One drawback, though, is that his presentation is tailored toward the SOLID API implementation, rather than being a complete look at the problem in general.

★★★★★ | Collision Detection in Interactive 3D Environments | Morgan Kaufmann  
[www.mkp.com](http://www.mkp.com)

*Jeremy works for Rockstar San Diego.*

## Hard-Boiled Developer

**Luxoflux's Peter Morawiec on bringing classic story genres to life**

**A**s co-founder of Luxoflux, Peter Morawiec has heard it all before. Having broadened the vehicular combat genre with games like *VIGILANTE 8* and *STAR WARS: DEMOLITION*, he's used to fielding comparisons to pioneering titles. Well, it was déjà vu all over again, as gamers, marketers, and, yes, even journalists made connections between Luxoflux's recent *TRUE CRIME: THE STREETS OF L.A.* and the seminal sandbox title *GRAND THEFT AUTO III*. This time out, the story took an unexpected turn when author Robert Crais accused the game's developers of infringing on his novels. When Luxoflux and publisher Activision showed him the game, he saw how distinct it was from his work, and dropped the suit. Although this added an unwelcome layer of drama to the release of *TRUE CRIME*, it validated Morawiec's dogged pursuit of innovation. As project/design lead on *TRUE CRIME*, Peter delivered that innovation by integrating the story tightly into the game.

**GD: What factors led you to connect *TRUE CRIME*'s gameplay so dependently to story?**

**PM:** I'm a big movie and fiction buff, so *TRUE CRIME* was always envisioned to be a story-driven game. I was hoping we could create something akin to a videogame incarnation of an action film, where the story and gameplay blend into one another seamlessly (subject to load time limitations). We used very short, palatable cinematics to progress the story, while placing the action segments in the player's hands.

Simultaneously, I wanted to achieve a sort of hybrid active-passive experience, where the entertainment goes on no matter how badly the player does, allowing even a total newbie to fumble his or her way through an entire storyline, without repeating missions or getting stuck. In a passive medium such as a movie, whenever the hero hits a low point mid-film, the story doesn't restart; rather, the hero recovers or finds another way to go on. This is especially true in detective stories, where the protagonist tends to encounter a few dead ends before eventually connecting the dots. However, many gamers will instinctively want to replay a failed mission, so the jury's still out on this particular feature.

**GD: What is your writing process?**

**PM:** I prefer to arrive at a condensed story outline first—the general theme, the hero, the villain, their motivations, the climax, the introduction event, key branch points, some loca-



Peter Morawiec makes crime play.

tions, and story twists. At this stage, I try not to concern myself with gameplay issues much, but I never discard those considerations completely either. The next step involves detailing out the story and breaking it down into individual cinematic and gameplay components. Games tend to be considerably longer than films and most time is “action time,” so you've got to stretch the script and proliferate it with gameplay mechanics pertinent to your game. Unlike a traditional movie script, each conversation is described merely in terms of its content and tone, not the actual final dialogue, which comes last. We hire professional writers to assist us with tuning the script and developing all dialogue. One of the lessons of *TRUE CRIME* is the need to write matching

VO for both cinematics and gameplay—the game is very campy throughout, but it also features several darker moments, so the main character's generic one-liners often end up out of synch with the mood of the story.

**GD: Given *TRUE CRIME*'s connection to hard-boiled detective stories, how important is it for games to explore narrative genres?**

**PM:** As the videogame market matures, I believe it's natural for story-driven games to be crafted within established narrative genres. With the age of today's average gamer pegged at something like 29, the audience welcomes a greater thematic variety as well as deeper and more mature storylines. I believe that people will instinctively want to play the same types of genres they like to watch or read. As a matter of fact, we've already seen a number of successful games dubbed as horror, film noir, Hong Kong action, and so forth.

**GD: In light of *GRAND THEFT AUTO III*'s notoriety, is it better to market a crime game as a GTA-killer or as an original experience?**

**PM:** As a game maker, I'd clearly prefer the latter. However, from a sales standpoint, I'd imagine it is always beneficial to make bold claims (so long you can back them up). Either way, the challenge lies in managing consumers' expectations, which is an extremely tricky thing.

**GD: In retrospect, what steps can a writer and a project lead take to head off infringement claims before lawsuits are filed?**

**PM:** Games are big business, so as long as there are willing lawyers, there will be lawsuits. The best thing to do is to perform plenty of legal due diligence before you ship.

**GD: What games are you playing now?**

**PM:** *CALL OF DUTY*, *SHREK 2* (in development internally), and eagerly awaiting *HALF-LIFE 2*. 🎮

# Designing the Language Lerp: Part 3

Lately, I've been developing a programming language called Lerp. Lerp is an imperative language with some declarative extensions for data handling. The declarative statements are based on predicate logic ("Predicate Logic," December 2003), a simple way of reasoning with facts well known in the AI community.

In the past few articles I've shown how predicate logic expressions can be used to manipulate data in concise and powerful ways. However, the traditional handling of predicate expressions, in languages like Prolog, has some problems that need to be addressed. Prolog was never adopted for widespread use; I believe this is partially due to some software-engineering shortcomings that Prolog proponents were slow to acknowledge and fix.

## Software-Engineering Problems

A language with good software-engineering properties helps you keep a program from becoming too chaotic as it grows; such a language supports software-development patterns that result in fewer bugs and makes it easy to find bugs when they do happen. As an example, C++ performs type checking at compile time and link time, so many common errors (like passing the wrong argument to a procedure) are caught and fixed before you ever run the program. On the other hand, LISP doesn't have static type checking, so you can only find type errors at runtime. Those errors might lurk for a long time, if they're in code paths that are infrequently exercised. So C++ provides a definite advantage over LISP in terms of getting real work done.

On the whole, predicate logic is an error-prone method of expression. Traditionally, there's not much in the way of compile-time error-checking. The speed of your program and its correctness depend drastically on small variations in the way the predicates are written. I'll illustrate this with some examples.

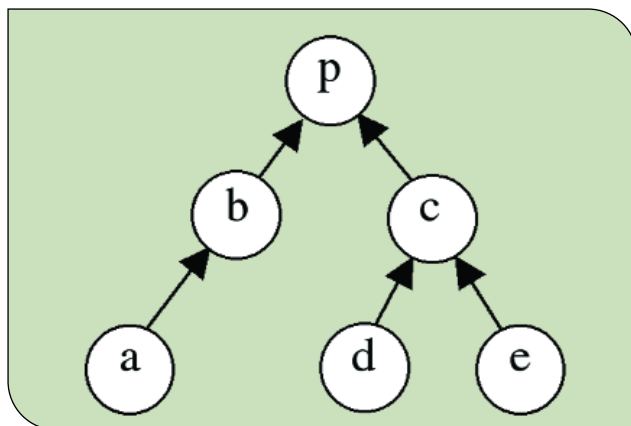


FIGURE 1: We represent this set of nodes arranged in a tree in predicate logic assertions as ['parent a b], ['parent b p], ['parent c p], ['parent d c], ['parent e c].

Suppose you have some objects arranged in a tree; there is a parent relationship that links objects together (Figure 1). We want to ask whether one object is an ancestor of another, in other words, whether it can be reached by traversing some number of parent links. Using Lerp syntax, we would define the ancestor predicate like this:

```

['ancestor ?x ?a] <- ['parent ?x ?a];
['ancestor ?x ?a] <- ['parent ?x ?p] & ['ancestor ?p ?a];
  
```



**JONATHAN BLOW** | Jonathan Blow normally has a new blurb in this box every month. This month, though, he forgot. Send recommendations or your favorite anti-senility medicine to [jblow@gdmag.com](mailto:jblow@gdmag.com).



The first line says *a* is an ancestor of *x* if *a* is the parent of *x*. The second line says *a* is an ancestor of *x* if *x* has some parent *p*, and if *a* is an ancestor of *p*. The second line performs recursion and the first line handles the simplest case; the two lines together provide a complete definition of ancestor.

One of the nice aspects of predicate logic is its declarative structure. In theory, you just state the facts, and the runtime system figures out the answers to your queries based on the stated facts.

However, in reality, it's just not that simple, because facts can't get up and solve problems by themselves. In

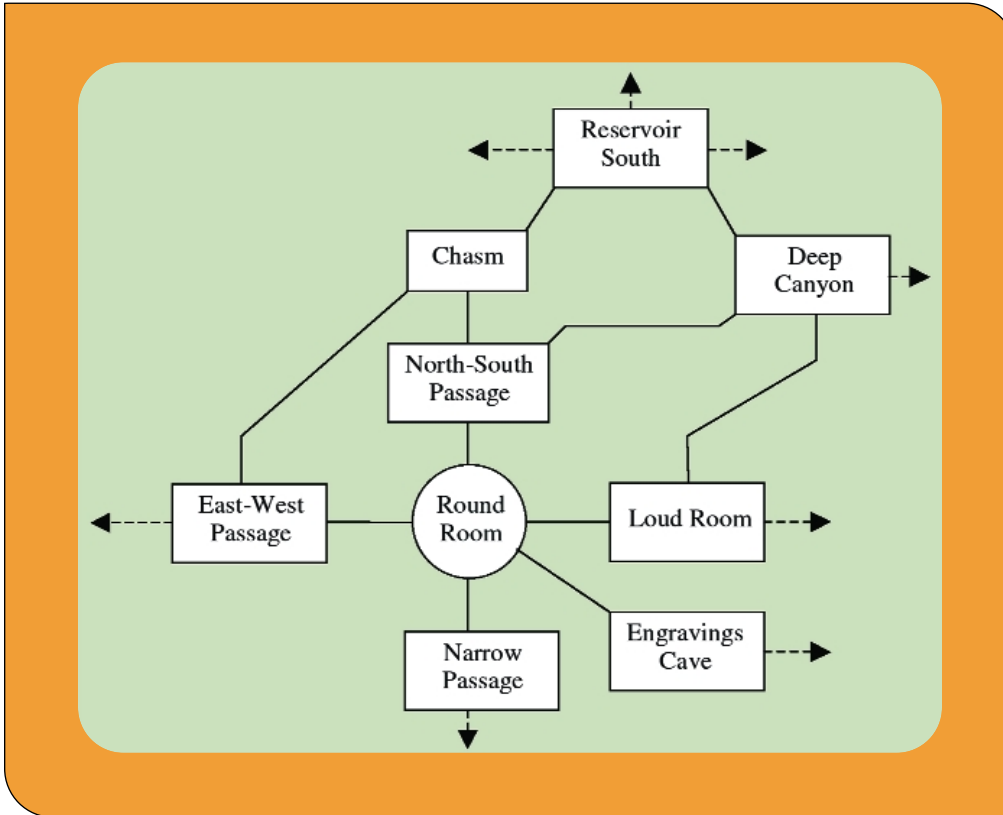
predicate logic systems, there's a solver algorithm that tries to put facts together to eliminate unknowns. The problem with traditional predicate logic systems is that, due to concerns over

performance and correctness, we end up spending most of our time thinking about the course of action this solver algorithm will take to put our facts together, when we're supposed to be thinking about just the facts themselves.

*We end up spending most of our time thinking about the course of action the solver algorithm will take in order to put our facts together, when we're supposed to be thinking about just the facts themselves.*

For example, Prolog's solver will consider the facts one by one in the order you list them (for details, see the sources listed under For More Information on page 19), so the ancestor relation as written above would be an efficient way to program. But suppose you switch the order of the statements like this:

```
[^ancestor ?x ?a] <- [^parent ?x ?p] & [^ancestor ?p ?a];
[^ancestor ?x ?a] <- [^parent ?x ?a];
```



**FIGURE 2:** A graph that contains cycles will cause problems for a Prolog solver. In this scheme, each arc between the nodes represents an assertion of the form of [`'neighbor ?a ?b`]. To ask whether a path exists between rooms, we can define a primitive `connected` that traverses these neighbor facts.

an arbitrary graph (Figure 2). Let's say we have a fact, `neighbor`, which tells us whether two nodes have an arc between them, and we want to define a relation, `connected`, which tells us whether two nodes are some number of neighbor-steps away. Well, a Prolog solver will just go into an infinite loop on this problem, period. To fix this, we need to add lots of extra data and

Now you have a program that is extremely inefficient. It will always perform a recursive search first, climbing all the way to the root of the tree and whizzing right past the answer, only finding the answer as it backtracks.

The situation can get worse if, for instance, you switch the order of the conjunction in the first rule:

```
[ancestor ?x ?a] <- [ancestor ?p ?a] & [parent ?x ?p];
[ancestor ?x ?a] <- [parent ?x ?a];
```

Now the solver will just loop infinitely. An attempt to answer any ancestor question will cause the solver to immediately pose another ancestor question. As long as our examples remain this simple, we can work around them without too much strife. Maybe we can add some compile-time checking to spot simple loops like the one above. But, as with any language, the situation gets murkier when the program gets bigger, due to additional relations that reference one another recursively in various ways. As the program grows, we must work harder to manage the solver's data processing. Writing a large-scale software project like this will not be as much about logic as one would hope.

To top it all off, there are a lot of places this kind of solver just can't reach. Imagine we want to make the solver navigate

relations to manage the solver's progress, and we end up doing the same kind of work we must do in imperative language, except in a more Byzantine manner. Recent variants of Prolog, and other logic programming languages, apply numerous band-aid approaches to this problem, but I have not seen one that solves the problem to my satisfaction.

Flow control is implicit in such declarative languages, so it's very hard to manage if you need to actually steer the process. This is not nearly as nice as in an imperative language, where the programs are lists of statements that say, "Do this, then this, then this."

## Control the Solver

This straightforwardness of imperative languages is why I designed the core of Lerp to be imperative, but so far there's an extreme asymmetry in the language design. The imperative part can scale, with reasonable confidence that the program will do what you hope. The logic part, though, is only useful for simple tasks; as facts in the logic system become more complicated, the program will probably start running slowly or loop infinitely—who knows? I don't feel I can produce reliable programs with acceptable efficiency when things get large on the logic side.

Maybe that's not so bad—if the imperative side is good enough, maybe the language philosophy should be that logic is only used for simple things. Certainly, the matrix and vector examples previously shown (“Designing the Language Lerp: Part 2,” February 2004) are interesting and useful, and we could just accept those (along with the `carrying` example) as the limit of what the logic in Lerp will do. But I want to push it further.

## Simplified Ancestor

Throughout the language design process, I have been keeping in mind the most effective features from other languages, like Perl, and looking for opportunities where they can be gainfully employed. In this case, I found a way to reframe Perl's regular expression handling. Let's go back and look at that definition of ancestor:

```
[`ancestor ?x ?a] <- [`parent ?x ?a];
[`ancestor ?x ?a] <- [`parent ?x ?p] & [`ancestor ?p ?a];
```

What we're trying to say here is, “An ancestor is anyone you

can reach by following a parent relation one or more times.” This is interesting because “one or more times” is a common, primitive concept in regular expression-pattern matching; it's denoted with a `+`. In fact, if we treat `parent` as a single symbol, then the idea “one or more parent relations” would just be written in regex form, as `parent+`. With that in mind, we can develop an alternate syntax where `ancestor` is defined like this:

```
[`ancestor ?x ?a] <- /{ ?x `parent+ ?a };
```

This definition treats the database facts of Figure 1 like a graph. The `/{}`  are just syntactic markers, saying that the braces contain instructions about how to walk the graph. This particular graph-walking expression says, “Start at `x`, then follow one or more `parent` arcs until you arrive at `a`.” The symbol `parent` is still assumed to be a binary operator like in the previous definition. Think of `{ ?x `parent+ ?a }` as expanding as follows:

```
[`parent ?x ?tmp_1] & [`parent ?tmp_1 ?tmp_2] & ... & [`parent
?tmp_n ?a].
```

If it helps reduce confusion, we can reformat the parent facts in the database so the identifier `'parent` is infix rather than prefix, so the facts look like `[node1 'parent node6]` instead of `['parent node1 node6]`. This is just a cosmetic change, and clearly we can do a graph-walk among facts stored in either format.

## Complex Graph Walks

**W**e can compose longer graph-walking expressions. Suppose we want to find whether `x` and `y` have a common parent. There are several ways to form this query. We could say `{ /{ ?x 'parent+ ?p } & /{ ?y 'parent+ ?p },` meaning, “Is there some common `p` reachable by parent-steps from both `x` and `y`?” Or we define the relation `child` as `['child ?a ?b] <- ['parent ?b ?a]`; then we can string together one expression that looks like this: `{ /{ ?x 'parent+ ?p 'child+ ?y }.`

That may be easier to think about, since it talks about one continuous path, from `x` up to `p` down to `y`. But the point of `child` is just to define a transition that goes in the direction opposite of `parent`. Requiring a separate definition for that is a little cumbersome; instead, we can add some notation to the graph-walking expressions to say, “Switch the order of the arguments before searching for this fact in the database.” Let’s use the `~` character for this, with the little wave symbolizing the swapping of two things. Then the query becomes `{ /{ ?x 'parent+ ?p ~'parent+ ?y };` whether you prefer this is a matter of taste, but it does make for a simpler program. Another way to think of `~` is that it means, “Go backward along this graph edge instead of forward.”

Let’s apply this to a concrete game situation. Suppose you’re making a fantasy game, and a dragon has just breathed on a player, whom we will call `victim`. We want to apply `damage_amount` points of damage to everything flammable the player is carrying. We know an object is flammable if it has a member variable `flammable` set to true.

The catch is, we need to search for items that are not in the player’s top-level inventory. If the `victim` is carrying a bag, and there’s a `scroll` in the bag, then we will have facts `['carrying victim bag]` and `['carrying bag scroll]`, but probably no `['carrying victim scroll]`, since such deep linking would make data handling cumbersome and expensive. So when it comes time to burn the stuff, we need to perform a recursive search through the player’s inventory to make sure we find everything. Supposing we have some function `apply_damage`, we can invoke it on all relevant entities like this:

```
apply_damage(damage_amount, each /{ victim 'carrying+ ??.flammable });
```

In this graph-walking expression, the `carrying+` finds all the entities starting from `victim`, the `??` indicates the node in that slot should be the return value of the expression and the `.flammable` looks up the member variable `flammable` on whatever

value we reach. If that member variable doesn’t exist or is false, the traversal fails for that particular path and returns nothing; otherwise, it returns the node in the `??` slot. Recall that the `each` as a function argument causes `apply_damage` to be called once for each entity that satisfies the query.

That’s pretty good. It’s different from what you’d do in many other languages, and it’s certainly simpler. You can imagine ways of adding further regexp syntax to handle fire-proof containers. It works out to be pretty simple. In fact most of the convenient features of regular expression syntax can be adopted directly into this graph-walking scheme (parentheses to join patterns into larger groups, and so forth).

## What this Notation Does

**T**his graph-walking syntax helps us phrase queries in ways that are more straightforward and compact than with traditional predicate logic. But it also does something more important—it gives us a way to avoid the recursion and termination pitfalls. When evaluating a term like `carrying+`, the graph-walking engine can just assume some things that are difficult to deduce in a general predicate logic environment.

Going back to the non-hierarchical situation of Figure 2, a graph walker can easily solve the query with no risk of infinite looping, without requiring the programmer to add ancillary data. It can just mark the nodes it has visited and never try them again, since there’s no reason to visit a node twice when evaluating a pattern option like `+`. This is valuable, since it widens the scope of queries that the programmer can make while feeling confident and safe. This month’s sample code (available at [www.gdmag.com](http://www.gdmag.com)) implements the examples discussed here, and some others.

## Relation to SNePS

**T**he regular expression query syntax treats a predicate logic database as a series of nodes connected by arcs. Interestingly, there are some systems developed in the AI world centered around this idea of graph traversal. The most prominent one is SNePS (see For More Information), considered by some to be a helpful tool for knowledge representation in natural language processing. The semantics of SNePS are not exactly like what we have discussed here, though there are some interesting similarities. 🦄

### FOR MORE INFORMATION

Clocksin, W.F. and Mellish, C.S., *Programming in Prolog* (Springer-Verlag, New York, 1987).

SNePS Research Group  
[www.cse.buffalo.edu/sneps/](http://www.cse.buffalo.edu/sneps/)

# Subdivide and Conquer

## Part 2: Practicalities

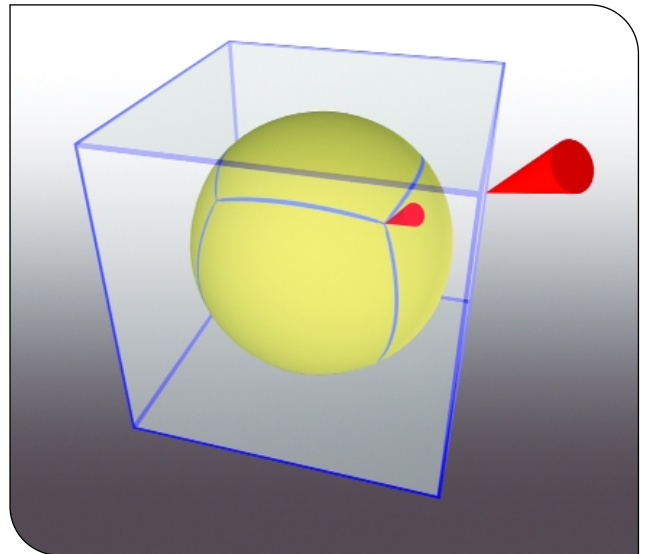
**T**his month we're going to return to the discussion of subdivision modeling we started previously (Artist's View, January 2004). Since the theoretical part is out of the way, we can now focus on some practical rules for dealing with subdivisions.

First, let's quickly recap the basics from our earlier discussion. A subdivision surface is made by recursively refining a polygon mesh. The topology of the original mesh determines the quality of the smoothing. Connected mesh edges that pass through the four-way intersections in the control mesh (edge loops) create smooth B-splines on the subdivision surface. Conversely, vertices that have more or less than four incoming edges (pole points) terminate edge loops, interrupting the flow of the surface and introducing a cusp on the surface right under the pole. Faces with more or less than four sides also create smoothing glitches when they are refined (extraordinary points), although in a typical game model these aren't too serious.

For these reasons, the ideal subdivision model is built from a grid of quads. Since all of the faces are four-sided, they subdivide smoothly. More important, since all of the internal intersections in a quad grid are four-way, every series of connected edges in the mesh is a smooth spline, so it's easy to build models with smoothly curved surfaces. Note that a mesh is only a quad grid if all of its vertices (except those on the outer borders) have exactly four incoming edges. Many meshes that are made entirely of quads don't qualify as quad grids (Figure 1).

### Advice for the Left Brain

**B**ecause of the stress subdivision modeling places on mesh topology, it tends to require more forethought than traditional polygon crunching. Bay Raitt, who did the subdivision models of Gollum in the *Lord of the Rings* movies,



**FIGURE 1.** An all-quad mesh can still have pole points, where more or less than four edges meet. Note the discontinuity of the splines on the surface of the subdivision sphere.

describes the subdivision workflow as “volume, surface, detail.” This top-down process stresses the importance of working from the largest aspects of the model to the smallest. This is usually a good practice in any case. With subdivisions,



**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 [sstheodore@gdmag.com](mailto:sstheodore@gdmag.com).

it's especially important, since an accidental pole point can distort the sweep of a large contour. Thus, in my capacity as a conscientious columnist, I am obliged to make a public pitch for careful planning.

In any case, subdivision tools implicitly favor a top-down process. Most major packages implement Hierarchical Subdivision Surfaces (HSDS), allowing you to work directly on vertices created by different iterations of the subdivision process. HSDS is invaluable for localizing details; the wrinkles in a character's forehead, for example, are best done locally at a higher subdivision level than the rest of the model. Without the ability to drill down in the subdivision hierarchy, the detail areas would have to be carefully stitched to the lower-resolution regions or they would propagate unneeded control points to the simpler areas in the way NURBS models often do. Moreover, HSDS details will move with their parent surfaces if the base control mesh is changed, so large scale edits can be made without disturbing completed detail work.

Unfortunately HSDS has important limitations. Most importantly, the topology of the hierarchical subdivisions is determined by the topology of the original control mesh. So, for example, you can't add a wrinkle line across a forehead in a higher subdivision level unless the edges flow in the right direction on the base level (Figure 2). Moreover, in most

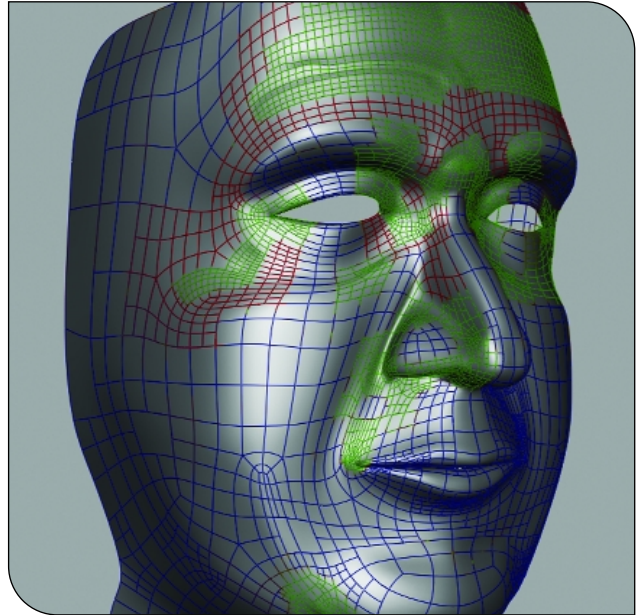


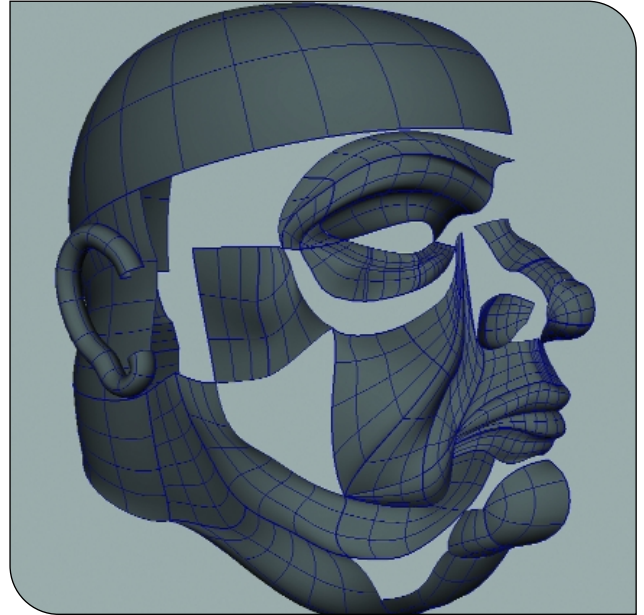
FIGURE 2. HSDS's lower levels derive their topology from the base mesh, which could be a limitation in some cases.

---

implementations, changes to the base topology can confuse the detail layers, leading to unpredictable migrations of details as the vertex indices change. Most important, HSDS doesn't change the basic behavior of subdivisions and the need for clean topology at the highest level. Thus HSDS is ideal for organized modelers, but offers relatively little to intuitive, improvisational, or just plain sloppy artists.

## Back to the Future

**N**URBS are often a good starting point for a well-planned subdivision model. Beginning your model with NURBS is also a good way to include a strong foundation of quad grids, since NURBS patches are fundamentally quad-based. More important, though, starting the model with NURBS helps to separate the volume stage of modeling from the surface stage. NURBS patches can be rebuilt on the fly to almost any level of detail. With subdivisions, on the other hand, adding or removing control points always involves some alteration in the final form. Roughing out your main forms in NURBS allows you to concentrate on the model without worrying too much about the distribution of the control mesh. Once the shapes look right, you can go back and assign density where it is needed by cutting up or rebuilding the patches.



**FIGURE 3.** Generally it's a good idea to define major forms in NURBS before starting a subdivision model.

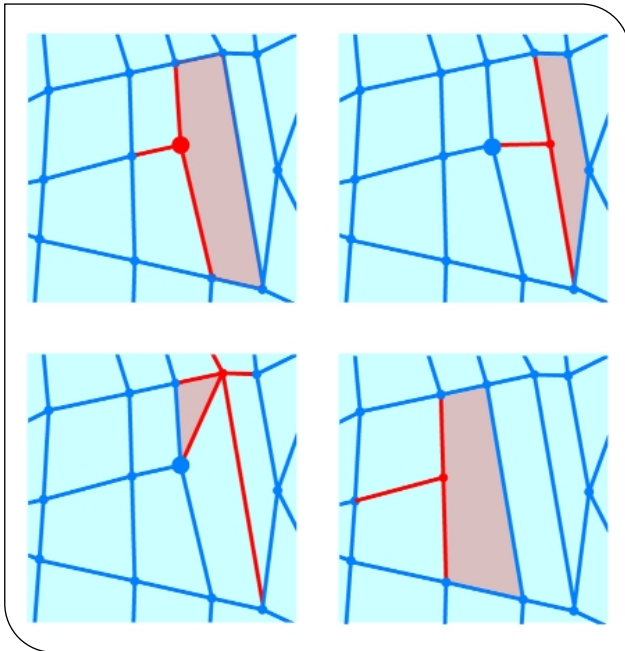


FIGURE 4. Trying to fix the pole point and  $n$ -gon (upper left) simply moves the problem around.

Integrating NURBS into a subdivision model is fairly simple. You simply convert your NURBS into polygons, which are then stitched together to serve as the basis of a subdivision-control cage. Generally, the most efficient way is to model only the important forms, leaving the connecting areas for the subdivision model (Figure 3). Subdivisions handle oddly-shaped, arbitrary blends more easily than NURBS do, so there's no need to build NURBS shapes—such as boundary surfaces, trim-blends, and multi-sided fillets—that are basically there to serve as fillers. Here's a good rule of thumb: anything that requires a lot of thought in the NURBS stage is probably better fixed in the subdivision stage. The only really finicky task worth doing in NURBS is matching up the isoparms of adjacent patches, which will reduce the time required to stitch together the subdivision cage.

The one important trick is to convert the control cage of the NURBS model rather than the final limit surface. In other words, you want to copy the NURBS control cage into your subdivision model. Maya offers an option to do this directly; however, in Max or XSI, you'll need a script that builds a polygon mesh out of NURBS CVs. Unfortunately there are subtle but important differences in the underlying math (see Artist's View, January 2004), so conversion between NURBS and subdivisions will not be perfect. For most purposes, how-

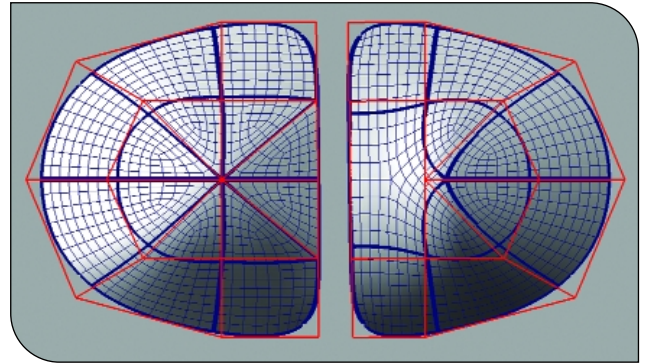


FIGURE 5. Pole points with an even number of incoming edges (Left) are more predictable than those with an odd number (right).

ever, the conversion is adequate without additional tweaks. You can improve conversion accuracy by increasing the number of NURBS isoparms before conversion.

## Now for the Right Brain

**N**ow, it's time to return to the messiness of the real world. Most of us are too busy thinking about our subject matter to worry about the topology every time we touch a mesh. Moreover, we often tend to obsess over local details that crop up, because the little details often tell us so much about the way the overall model will evolve.

The result is that most of us—especially those who learned the trade as low-polygon modelers—add or subdivide faces more or less at random when trying to capture an elusive detail or suggestive contour. It takes a lot of discipline to follow a strict top-down methodology.

Therefore, it's time to take a look at the alternative to planning: cleanup. Artistic issues aside, the ordinary give-and-take of roughing out a mesh almost always leaves behind a trail of accidental pole points and unintended  $n$ -gons, which need to get polished off before the model is truly ready for prime time. Trying to eliminate nasty little shading glitches or contours that refuse to stay put can consume a lot of time, even when a model appears to be nearly done. So let's look at some practical aspects of fixing an existing subdivision cage for optimum results.

For game modelers, the key task is really two-fold: deciding which vertices have to be fixed and which faces have to be quadrangulated. In theory you could build a complete quad grid mesh with hordes of tiny polygons taking the place of irregular intersections and faces; however, the practical problems with trying to manage such a heavy mesh generally outweigh the benefits, at least in game applications. High-resolu-



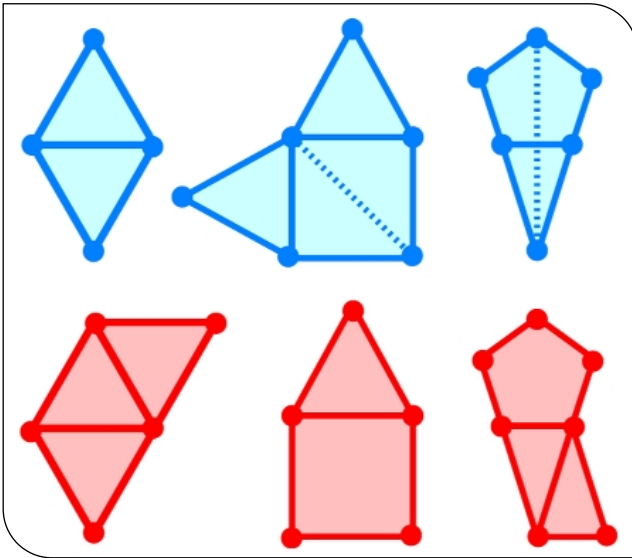


FIGURE 6. Here are some examples of possible (blue) and impossible (red) quadrangulation when adding adjacent faces.

tion modeling for films or real-world CAD applications may demand complete topological purity. For games (even game cinematics), it's easier and more realistic to accept some poles and  $n$ -gons as necessary evils.

## Sweeping Under the Carpet

Most of the time, moving topological problems into inconspicuous places is an adequate alternative to eliminating them. Usually, fixing a topology problem in one place causes a new one to crop up nearby (Figure 4). By far, the worst part of trying to regularize a subdivision mesh is the agony of spending half an hour tweaking a problem area only to realize you have reshaped the mesh back to its original configuration. Settling for simply hiding the glitches instead of eliminating them turns the frustrating task of chasing an error into a useful tactic. For this reason, keep an eye out for transitional areas that are obscure or don't have strong forms where you can offload tricky connectivity problems.

In other cases, the tradeoff is topological rather than spatial. Not all flaws are equally serious, so choose the lesser ones. For example, eliminating a pole point frequently means creating an  $n$ -gon, or vice versa. In general, if you are forced to choose between an  $n$ -gon and a pole, accept the  $n$ -gon, since a pole has two bad side effects (a shading glitch at the extraordinary point and a break in the flow of the surface spline) whereas the  $n$ -gon has only one.

It's also worth remembering that pole points with an even

number of incoming edges are also less problematic than those with odd numbers. Although an even-sided pole point still creates an extraordinary-point shading glitch, the surface flow under an even-sided pole is smoothly continuous along the opposite incoming edges (Figure 5). Odd-sided poles, by contrast, produce a clear discontinuity for all the incoming edges. The one exception to the even-odd rule is that vertices with only two incoming edges (unless they are on an open border of the mesh) are always a bad idea, since they usually produce very noticeable divots in the subdivision surface.

## Quad Squad

Because quads produce the cleanest and most predictable subdivisions, you always want to use them instead of the  $n$ -gons if you can.  $n$ -gons should be used only as connective tissues in areas without important structure or definition. When you're trying to grid up an existing irregular mesh, you'll find that any  $n$ -gon with an even number of sides can always be divided into quads without adding any vertices. Conversely  $n$ -gons with odd numbers of sides can only be quadded by splitting one of their original edges. Usually this will involve extending the split until you reach an acceptable pole point or an open edge of the mesh. When applying the even-odd rule for a group of faces, remember to subtract two from the total edge count for each shared edge. So, for example, a triangle adds one edge to a quad ( $4 + 3 - 2$ ) if it is connected by one edge, but actually subtracts one if it is connected to two edges. Thus, you can't make quads by deleting the edge between a quad and a triangle ( $4 + 3 - 2 = 5$ ), but you could do so by adding a quad and two triangles ( $4 + 3 + 3 - 4 = 6$ ) (Figure 6).

## Survival of the Fittest

Subdivisions are still in their infancy. Although they are a great step forward, they certainly don't represent the end of the line in the evolution of modeling tools. Most subdivision modelers are really just glorified polygon tools—they're great for quickly hacking around, but they don't give the artist a whole lot of leverage. Recently a lot of development energy has gone into useful but basically cosmetic improvements, like isoline displays or edge-loop selection tools. What's really needed, though, is better integration of more sophisticated functions. Tangency projection, freeform fillets, radius fillets, and—most importantly—trims would all tremendously upgrade subdivision modeling. So would native subdivision-support for tools like bi-rail lofts. Even mathematically correct NURBS-to-subdivision conversion would be a big step forward. If your software rep comes around the office asking what you want to see in the next version of your package, don't be shy—speak up! 🎨

# Laying the Smackdown on Voice Recording

**A**s an audio engineer at World Wrestling Entertainment, I record the voices of our on-air talent and wrestlers. There is an art to recording the human voice, capturing all its nuances and timbre, which requires the right mics, gear, and professional experience. This is especially true for videogames because others will rely on the quality of your recordings at a later stage of game development, and you must be right on the money.

**No gain, no game.** I record sessions flat; no EQ, no compression, and no DSP (digital signal processing). Our audio chain is quite pure: from the mic (Neumann M 149 Tube, Sennheiser HMD 25-1 headset mic) to the mic pre (Summit Audio MPE-200) to the console (Euphonix System 5 digital broadcast console), out of the console digital (AES) to a Sony PCM 7040 DAT. To have something to work with, it's critical to have a good gain structure, and a solid, strong signal.

For our latest game, *WWE SMACKDOWN! HERE COMES THE PAIN*, I decided to go with the Sennheiser HMD 25-1 headset mics because these are the mics that are used for the broadcast of *Smackdown* and I wanted the commentary to sound authentic, not done after the fact in audio post. This goes along with my theory of recording in general: Capture good sound from the original source and you won't need to do much to it to make it sound great. This way, when it's time to mix, half the battle of getting accurate sound is already taken care of.

**In your face.** Recording WWE wrestlers poses some unique challenges. Given the physical and psychological demands on these athletes, they are generally very intense people who have no time to waste. I have to be conscious of my gain structure. Too hot a signal will cause over modulation, which in the digital world is unforgiving and sounds awful. What I like to do is place a windscreen approximately 10 inches in front



**Balancing gain and personality:** recording WWE wrestler Mick Foley and on-air announcer Jonathan "The Coach" Coachman

of the mic capsule. People by instinct, tend to get real close to a windscreen, so by creating space (from the capsule to the windscreen) you effectively create a more favorable gain structure. Imagine if Stone Cold Steve Austin were asked to record some voice elements. He would probably be intense, just like he is in the ring. But, by having that space, I can control and adjust my mic gain (on the preamp) seamlessly, without wasting precious time, which is paramount.

**Keep it clean.** Preparing your talent is important because they need to know what is expected of them. For instance, I explain to the wrestlers how this is going to be used in a game, and that their words should be clear and deliberate. Usually, there are hundreds of phrases, exclamations that need to be "clean." Every wrestling move sound, grunt, and spoken name needs to be isolated. For example, if The Rock pounds Stone Cold Steve Austin with the "People's Elbow," then the sound of the interaction, Stone Cold's reaction, and the announcer's commentary need to be recorded sepa-

rately, so they can be used in varying combinations in real time during game play with perfect clarity.

**No conversions below the belt.** Other elements to consider are sample rate and bit rate. Our audio post rooms are outfitted with Fairlight MFX 48 digital workstations. We are running these systems at 48KHz/24-bit. The Fairlights work in tandem with the Euphonix System 5 console seamlessly. The Euphonix utilizes MADI technology so once the audio stream is converted to digital (AES to MADI), we can output to many different (AES) sources. For gaming sessions the result is a 44.1KHz 16-bit R-DAT. I output AES to a Sony PCM 7040 so there's no sample rate conversions to deal with.

Sample rate conversions can get a bit dicey. Remember digital audio is a binary representation of an analog sound. The more conversions you introduce into your bit stream, the further away you are from the original analog sound. Consider the audible difference between a WAV file and a compressed MP3 of the same source. The same holds true for sample rate conversions. The more pristine the signal, the better it will sound at game time. The ideal is for a developer to be able to plug this audio directly into the game without conversions.

**Before you step out of the ring.** To eliminate "fix it in the mix" scenarios, I apply some knowledge learned from analog audio engineering. One trick is to pot up all your faders to zero, then use the mic trims to get a good strong level. The key is to actually listen to the sound you're getting. This may sound simple, but it's easy to overlook, and can make or break your in-game audio. 🎧



**TIM ROCHE** | Tim is the audio post engineer for World Wrestling Entertainment, working on WWE's TV programming and games such as *WWE SMACKDOWN! HERE COMES THE PAIN*, *WWE SMACKDOWN! SHUT YOUR MOUTH*, and *WWE SMACKDOWN! JUST BRING IT*. He also worked on a wide range of TV shows and videos.

# Beyond Entertainment?

**T**he late media analyst Marshall McLuhan once said, “Those who make a distinction between education and entertainment don’t know the first thing about either.”

When I first heard the quote many years ago, I thought he was overstating the case: there is a lot of entertainment that has nothing to do with education, and much of education is far from entertaining. But the more I think about it, particularly about the game design work I’ve done that’s intended for teaching or training, the more truth I see in his statement and its pertinence to games. Games are teaching exercises: you play them as long as there is something new to learn, whether it’s the controls of an X-Wing fighter, the right kung-fu sequence to disable an AI opponent, or the crystal rotation method required to finish a level. On the academic side, the best teachers I’ve had are those who made learning fun and turned the process into a game, a competition, or a journey of discovery.

The 2004 Game Developers Conference includes for the first time a Serious Games Summit focusing on games with training or educational goals. I’ve experienced the growth and diversity of this field firsthand, designing games that encourage good nutrition, help kids with cancer cope with their treatment regimens, or help Shell employees understand what their colleagues in exploration and production go through to find and collect oil and natural gas. The well-publicized area of educational games based on academic subjects is just a subset of this branch of game development. The Serious Games Summit allows the less-publicized and burgeoning areas that focus on adult learning and training to come to the forefront.

Inspired by the upcoming summit, I invited two experts in this field—Ben Sawyer, cofounder of Digital Mill and designer of Virtual U, and Marc Prensky, founder of Games2Train and author of



**If ROGUE SQUADRON can teach you to fly an X-Wing, can games teach you to eat right?**

*Digital Game-Based Learning* (McGraw-Hill, 2000)—to suggest some relevant game design rules. They provided enough material for several articles. Here’s a summary of some of their key points.

Ben Sawyer counsels developers to focus on a mission, a desired outcome that transcends the usual industry goal of selling lots of copies. This focus reminds us the game “is usually a subset of a larger project with greater goals for the client.” He also points out that, with this sort of game, we are designing for more than just the player and must address the needs of the instructors, teachers, or trainers who might want to customize it and monitor the participants’ progress.

Finally, he suggests we remember to apply the numerous strengths of our industry and not simply the most obvious selling point that “good games are fun.” For example, games can be easily customized by teachers and trainers and can adapt to the individual. At the same

time, game developers are experts in the practical application of AI, 3D graphics, and simulation. Ben believes we need to demonstrate these comparative advantages to potential serious game sponsors to promote the use of game industry techniques in training.

Marc Prensky shares my uneasiness about the term “Serious Games,” since this can erroneously imply that educational games are not fun or that traditional games with no purpose other than entertainment are somehow not serious business. Making a game fun is a great challenge in itself; providing game content that teaches a player something real is even harder. His first rule: “Content is important, but fun has to trump content—don’t suck the fun out!” He suggests posing a few questions to validate the effectiveness of these learning games: “Is this game fun enough that someone who is not in its target audience would want to play it (and would learn from it)? Do people using it think of themselves as players rather than students or trainees? Is the experience addictive? Does it encourage reflection about what it teaches?”

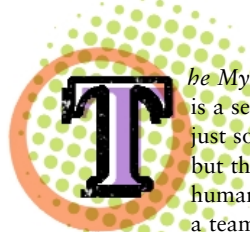
Both Sawyer and Prensky discuss at length ways to convince potential clients that games and game industry techniques have merits as educational and training tools. My experience suggests that as people who grew up with computer games mature and move into authoritative positions in traditional industries, we’ll see a huge increase in the acceptance of games as a fun and effective way to train people. I’ll follow up in a future column with lessons learned at the summit. 🍷



**NOAH FALSTEIN** | Noah is a 24-year veteran of the game industry. His web site, [www.theinspiracy.com](http://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 [nfalstein@gdmag.com](mailto:nfalstein@gdmag.com).

# The Secret's in the Schedule

## Bending *The Mythical Man-Month*



**T**he *Mythical Man-Month* is a seminal work on not just software engineering but the psychology of human interaction inside a team environment.

Nearly everyone in our industry is familiar with the name of the book and most have a cursory understanding of its central tenets. However, significantly fewer people have ever actually read the book or have a deep understanding of it beyond some well-known quotes that have become platitudes. This lack of deep understanding leads to two problems. First, people don't prevent actions that go against the very pretense of the book, which means they make mistakes that were identified as mistakes nearly 30 years ago. The second problem is people rely on an overly simplified understanding of the book and therefore are unable to conceive how its fundamental rules can be bent and twisted to allow that which seems impossible.

Obviously I recommend everyone put this book in his or her personal reading queue and get to it soon. Rather than rehashing the book's contents, the focus of this article is on ways to bend and stretch *The Mythical Man-Month* to its extremes; to help reduce restrictions of progress due to communication breakdowns and interdependencies.

The author of *The Mythical Man-Month*, Frederick P. Brooks, Jr., puts forth an idea that probably isn't too shocking to most of us. He believes software engi-

neering is radically different from all other forms of engineering primarily because there is no physical representation for our product. He argues that while studying the organization of other engineering fields can be useful, it cannot completely govern our particular craft. I'll extend this idea even further with another not-so-ground-breaking declaration: computer game engineering is by far the most esoteric of all software engineering, because at its essence is the difficulty of developing the all-important fun-factor.

### About Schedules

**T**he two primary ways to control the inherent limitations imposed by *The Mythical Man-Month* are through schedule and process. While the scope of this article is on scheduling, a complementary feature covering the process side appears on Gamasutra.com. *The Mythical Man-Month* states that adding people to a project experiences a law of diminishing returns until a point where growing the team actually creates a net loss in progress. The essence of this idea lies in the complexity of software engineering, where each team member works on a virtual object with a significant amount of attachment to other pieces. All of these separate pieces must line up correctly for the final product to succeed, so communication among people becomes an exponentially tightening bottleneck as the team expands. Translation: more of your day is being spent in meetings. This

is especially true as the project gets closer to the end and the volume of history knowledge reaches its peak. However, while this general tenet is true, the rate of decline and the transition where the new person added causes a net loss in productivity is variable. The best way to control these variables is through knowledge of your project's scope and resources, especially the team's manpower. And this understanding must go beyond simple head counts and reach a true understanding that your team is made up of very different people who need to be managed in very different ways. An effective management strategy including schedule and process can pull significantly more productivity out of a given team and thus stretch *The Mythical Man-Month* to its limit.

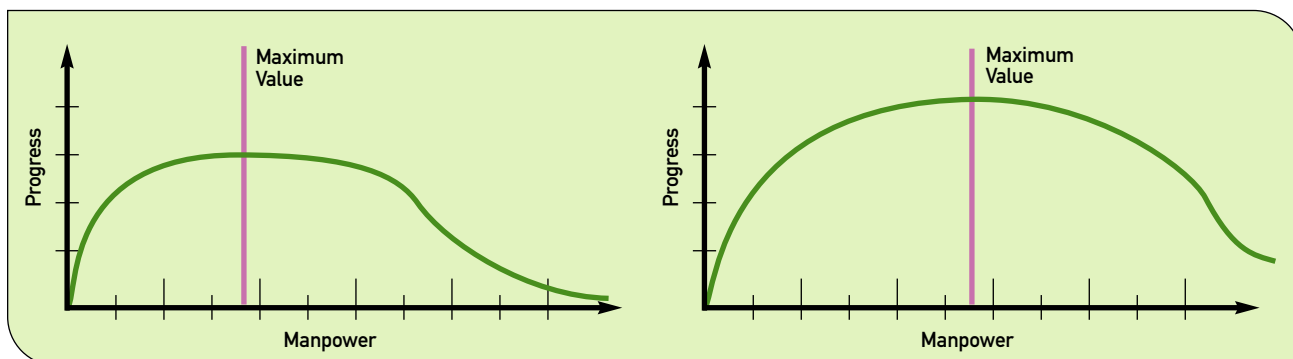
All successful projects must start with a schedule to understand the scope of the undertaking, no matter their timeframe. If you believe you don't have time to schedule, then I counter you don't have time not to. A schedule is your first line of defense against the communication breakdowns that kill a project's deadlines.

Amazingly, I still encounter teams with extremely poor or even nonexistent schedules. In my early years in the business, I worked on three games called ESOTERIA, TUBE RACER, and BENEATH. Never heard of them? Exactly. The first was barely released and probably only sold a couple hundred units and the next two were both canceled after numerous schedule overruns. All suffered endlessly

Michael Saladino | Michael just finished a year-long tour of duty at Microsoft Game Studios Publishing where he helped ship COUNTER-STRIKE for Xbox, MIDTOWN MADNESS 3, and TETRIS WORLDS LIVE. He's now at EA in Los Angeles where he's a development director, putting his money where his mouth is. Contact him at msaladino@gdmag.com.

*Start*





The figure on the left shows how just adding people to the project does not add value and can actually diminish progress due to overloaded communications. The figure on the right shows how improved scheduling can pull the maximum value line out and increase progress.

from a lack of proper scheduling. Since then, I have shipped every game I have led on time, mainly due to great schedules which gave me the information I needed to mutate my team when new problems arose. In my last year at Microsoft publishing, I've seen many games ship and many games killed: a constant theme running through them all is poor scheduling leads to canceled projects.

## Project Life Stages

To understand scheduling a game you must first understand the phases of existence that all projects go through. While the absolute time of each phase fluctuates based on the overall project length, there are common ratios of time between them. The first stage is concept development. You are in a creative period when blue sky brainstorming is the goal. At this point, you have no need for real schedules beyond simply setting a deadline for finishing these initial meetings. Upon completion of this phase, you should have a well-defined design document with discussion of not just the gameplay but also art and engineering-centric sections as well. This should take approximately 1/8 of the total timeline or about three months from a 24-month project.

The next stage is your prototype where you prove out the major unknowns, the foremost being the fun factor. This section is scheduled much like the whole game, only with an abbrevi-

ated timeframe, normally around 1/8 of the total project but sometimes growing depending on initial technology. Obviously, the more stable the tool set and engine you start with, the more likely the prototype would fall within the three month estimate. If the prototype is successful, you should leave this phase with a strong understanding of what will make the game fun and an example program that conveys this idea. Along with this will also be a system-level task list with estimated time allocations for designing these new systems. In other words, all the major systems might not be done but you should know what they are and have a rough guess as to the manpower needed to complete their first pass. This is the phase when independent developers should begin shopping around to publishers. While some development houses are lucky enough to get signed with only paper presentations, showing up with a working prototype goes significantly further when trying to secure a deal.

Your next stage is preproduction, which is when the team tackles the remaining uncompleted systems through at least the first implementation. For example, your new experimental lighting system with dynamic shadows should be up and running along with the decision to continue work on it or cut it due to problems. You should also have your first couple of levels completed to alpha quality to extrapolate the effort needed to complete the remaining levels. The first level

always takes the longest and usually ends up being the worst because your team is getting used to the process and the new game you're making. So you must complete two or three levels so the team begins approaching what the real world manpower per level will be. Once you've done this, you'll be able to map out the rest of the time frame and know if you need to reduce the level count immediately. This stage should take about six months from a 24-month project or 1/4 of the total project.

Now you enter full production, which is normally when your team hits its maximum size (without the extras needed for test, which normally comes on later). At this stage, the game should be fun and its entirety should be known. This is where your system level task list is constantly being refined into smaller and smaller resolutions. The process is becoming mechanical now as opposed to the freeform conceptual stages in the beginning. Your schedule is now the end-all-be-all of the project's existence. You should expect this time to last about nine months or a full 3/8 of the project making it the longest stage. This section of development ends with code and content complete meaning everything is in the game the way it was originally planned. It doesn't mean the game has to be completely locked down as final changes can still be made well into alpha as long as proposed changes significantly improve the overall quality of the game with limited risk.

And now we're brought to the end, the final stage. It's here that you've reached alpha and beta, also known respectively at Microsoft as code/content complete and zero-bug release (ZBR). These stages are mostly defined by long crunch times. Programming is focused on bug fixing, the art department on final polish, and the test team ramps up to full capacity. The schedule is becoming less structured and instead the team is being driven by daily and hourly updates derived from the test team along with oversight from the departmental leads and producer. Your bug-tracking software essentially becomes your schedule as you make your final sprint, which normally lasts about three months out of 24 or 1/8 the total.

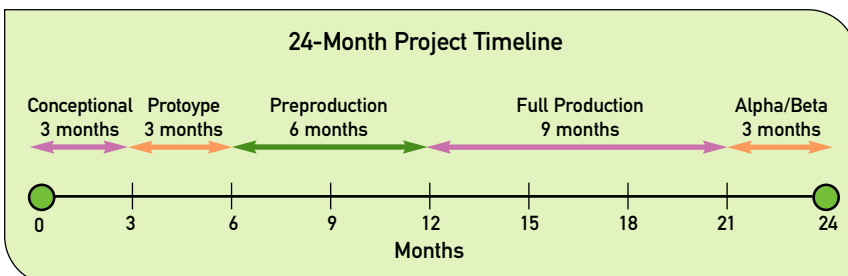
Once you understand these stages of development, you'll be better able to schedule them. One important point is that these fractions are meant to be guidelines, not strict rules. All projects expand and shrink these stages as they need them. While one project needs more time for prototyping, another needs less for production. Or perhaps your project is a simultaneous three-console release that will probably increase your absolute time spent in the final bug push. However, these guidelines should help you identify earlier when a particular stage is grossly beyond expectations. For instance, if you worked on a prototype for nine months, you shouldn't expect to complete the game in the next year. Your team obviously created lots of new technology and gameplay if it took nine months for a prototype, so ramp-up time alone when building the full game will prevent your

team from finishing in the coming year. Working at Microsoft publishing has shown these time ratios to work successfully for many projects including COUNTER-STRIKE for Xbox, which was essentially a five-month project, and WHACKED, which was a two-year one. These two projects had completely different absolute times but similar ratios between segments.

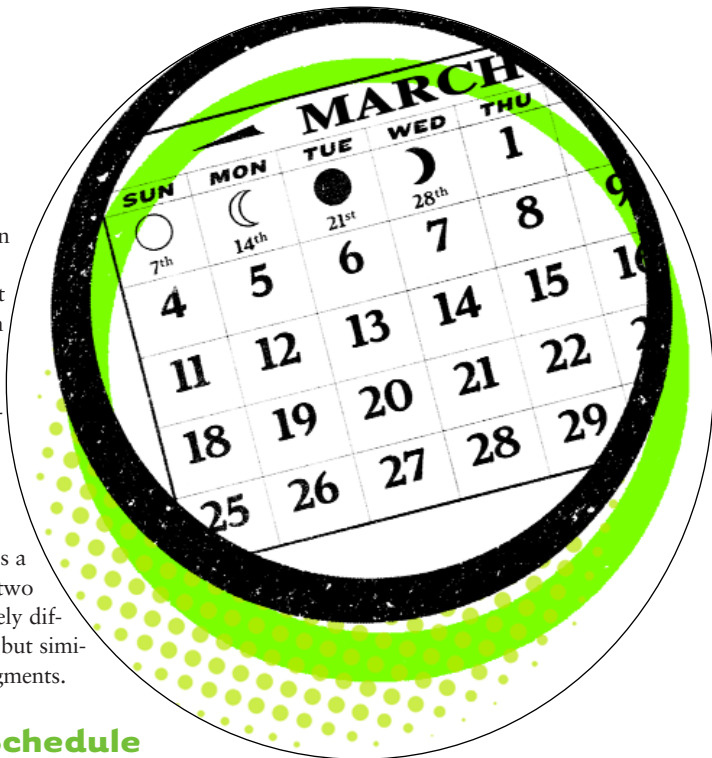
## Building a Schedule

Traditional scheduling in the form of a Microsoft Project document or a simpler Excel spreadsheet is most critical during the prototype, pre-production, and production phases of development. Concept development is too freeform and the final bug push is too reactionary; neither will benefit much from a schedule. The prototype phase is really just an abbreviated form of pre-production and production, so to understand those is to understand prototyping. Therefore, let's focus on pre-production, production, and the differences between them.

Creating a schedule first requires building a task list, which is in fact an expression of your design document. Are you building a racing game? Do you need a four-point suspension system? Do you need a flight control model? Is water surface dynamics critical to the boat level?



A sample timeline describing the phases of development. As actual project lengths and subjects vary, you can scale the proportions to fit your game.



Do you need a custom lightmapper? These game features need to be filtered into independent engineering, art, and level-construction tasks. The resolution of these tasks is dependent on where you are in the timeline of your project. We begin with system-level tasks during pre-production and constantly refine these as milestones are started, progressed, and completed. By the time you reach production with the major unknowns resolved, you should be able to make a valiant effort to refine the entire schedule down to days, but don't. Tasks should initially be timed at a resolution of about a week. Anything that needs to be completed in the next two months should be resolved down to days. Refining the resolution too much, too soon will be work wasted since the dynamic nature of a schedule will lead you to redo it anyway.

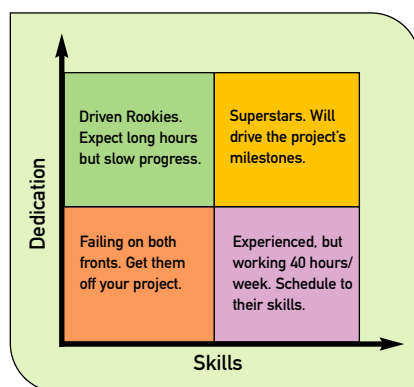
## Understand the Team

The next stage is understanding your team's manpower. You should begin by identifying the type of developer (any team member including programmers, artists, level designers, and so forth) each person on your team is. One simple way to do this is to analyze how they fall into four basic types, based on combinations of skill and dedication. These types can be represented on a simple 2x2 matrix with their skill level on one axis and their dedi-

cation on the other. Developers near the low end of both axes, those with little skill and poor morale, should be removed from the project after attempts are made to improve their dedication. Skill takes significant time to improve so don't expect that to rise over the course of one project, but a person's morale certainly can. However, if you can't promote improvement in a reasonable time, say 1/4 of the total project length, remove them from either the project or the company. Too many leads allow poor performers to stay in positions where they drop the ball and bring down the morale of those around them. Removing a poor performer can quite often create a net gain for the entire team even with the loss of the head. My previous work at Presto Studios showed me first-hand the damage one or two low-end members can do to the overall workload and morale of a team.

The next type of developers are the highly skilled persons who are poorly motivated. Maybe they don't like the game. Maybe they're angry because they didn't get the lead positions. Maybe they just came off a horrible crunch and just aren't ready to dedicate long hours again. Whatever the reason, they simply want to work their 40 hours, do their job effectively, and then go home. But since they are still highly skilled they are valuable in their own way. They will be most useful when working on exactly what they want to work on. If they're masters of networking, then that's where they go. They will put up with the least amount of unpleasant work when compared to the other developer types. Due to their skill level, they will require less hand holding but their strict hours mean keeping them on their schedule will be most critical. Luckily, their own sense of pride will often be their strongest motivation for getting the job done right and on time.

Then we have the young kids straight out of college with little skill but miles of desire. These developers want to prove themselves but they can get over their heads very quickly. Give them the systems with the most design in place. Put them in positions where they have the most number of seasoned developers working with them. They should be given non-perform-



A developer productivity matrix.

ance-critical code such as UI. They should be watched over by more experienced mentors and be expected to make up for their inevitable schedule overruns with long hours. At this point in their career, they are paying their dues. And if their rookie mistakes are kept to a minimum by monitoring them closely, their ambitious attitude can help light a fire underneath everyone in the group.

And finally, we are brought to the most important people on the team: the superstars. They are the highly skilled, highly motivated crew that makes the impossible possible. This is the white-hot fiery core of the sun that will drive your project when things get tough. They will work weekends without even being asked. They will bring in sleeping bags when necessary and work all night if needed to get back on schedule. Work for a 1:3 ratio between superstars and the other two types of developers. (Remember, you should have already fired the first group so we're only left with three total groups.) Your superstars are your first and last line of defense against missing your milestones.

## Assign the Team

Once you understand your team, you can successfully begin to assign people from your pool of talent to the task list you have built. Assign the highly skilled but poorly motivated people first. Give them the systems their skills match and they are interested in building. If you run into conflicts such as too many physics programmers all wanting to own

the system, try to exchange these resources with other teams. You do not want highly-skilled, poorly-motivated people working on systems they don't want to. If you do this they will most likely start dragging their feet or sending out resumes. Next, layer in the superstars who most likely can do almost any system you assign to them. These people have written graphics, physics, sound, AI, and most everything else so skill matching should be less important. After they have been placed, the first two groups of developers should cover every major system, leaving the eager rookies to round out the corners and fill in the gaps.

With people assigned to the task list, the lead should take the first pass at assigning time to each job. Be liberal with your estimates; optimism is a swift killer of any schedule, so assume mistakes will be made. Go ahead and assume your difficult, product-defining systems will need to be rewritten two or three times. Once completed, bring your team together and as a group discuss your estimates, gather contrary opinions, and negotiate final estimates for the schedule. Remember, even though you're the lead, it's the persons you've assigned to the task who have to complete it in the scheduled time. Give them the final word if you can't come to a consensus.

You now have a full task list complete with people and time, however it's still not a schedule. I've seen many projects over the years stop at this point without truly turning the task list into a schedule by serializing the pieces. This process begins by determining dependencies among the tasks. You can't texture a level until it's been modeled. You can't animate a character until it's been skinned. You can't program the front-end UI screens until a base GUI system is finished. This is what creates a timeline that shows people what they should work on on any given day. This is also how you can identify the communication dependencies that lead to one of the central tenets of *The Mythical Man-Month*. It's at this point that you can see just how interconnected your different systems and the programmers building them will have to be to complete the project.



We now must lay this data into our schedule software. Put each person down for six hours a day for five days a week. Even if you expect a death march, don't start by scheduling for it. That's a sure bet at blowing your deadlines down the road. The six-hour day covers the time during an eight-hour day the person will be in meetings, taking lunch, or hanging out in the kitchen eating a muffin with the team. Remember to put in holidays. (I laugh every time I see a schedule with someone completing a task on Christmas Day.) And as a general rule of thumb, assume each developer will take one personal week of time every four months.

Now you might look down at your schedule and think something has gone horribly wrong. One or two people in the schedule will have way too much work that puts them months or even years beyond the rest of the team. Workload balancing is required to fix this problem. Move tasks to other developers, starting with the easiest, but remember as tasks move away from their ideal owners they may need additional time padding for someone not as familiar with the system. This is also when the idea of additional developers should be first introduced. If you have a hard date

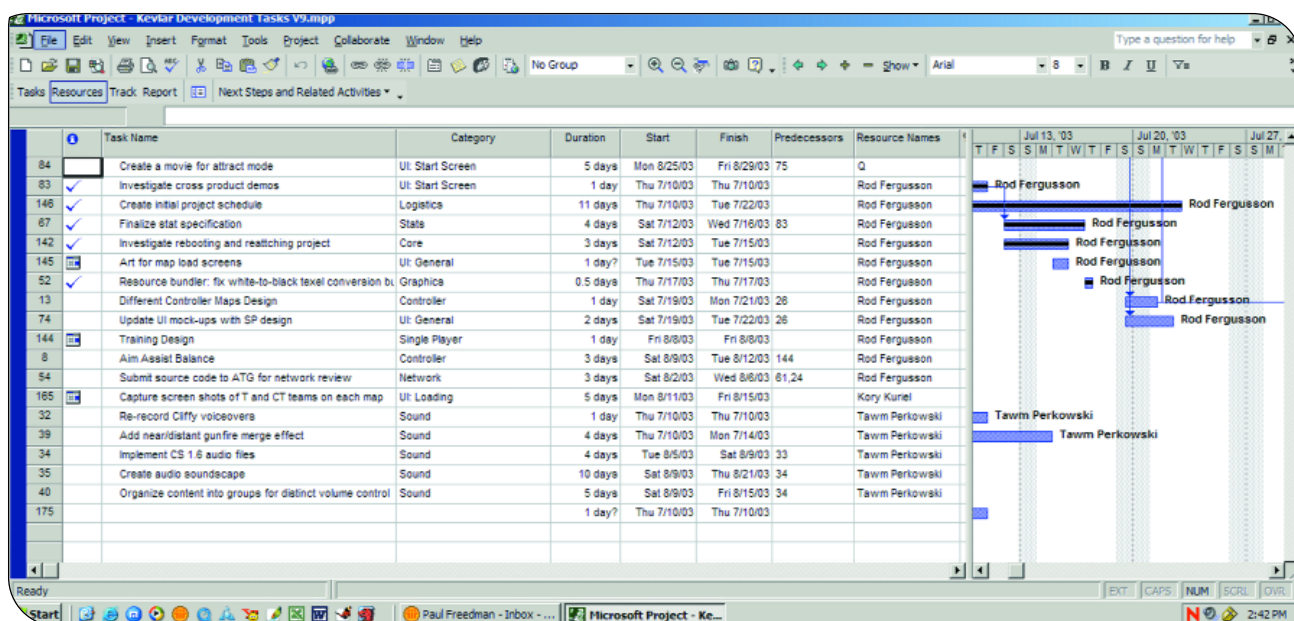
for delivery and the tasks just don't add up, start padding your team with TBD (to be decided) members. This true and honest schedule will be your ammunition to request more support from your boss. Fight the urge to simply schedule for a crunch this early in the process. If you can't get the people you need on this schedule, then make it clear cuts must begin now. By following these steps, you can start making these hard decisions early instead of in the final months.

After days or weeks of working out the tasks and timeframes, you now have a schedule. You should know the size of your team, the tasks they perform, the order in which they will perform them, and how long the total project should take. Of course, it's all a big educated guess at this point but it's infinitely better than nothing. And remember that the schedule is a living document that should be updated daily by the owner. Don't be afraid to change it and make sure people know when their sections do change or when a major overhaul is needed to bring the schedule back into reality. If any one person falls behind his or her milestone deliverable time by more than 10 percent, the problem should be addressed that day. Maybe the person

just needs to talk the problem out with a mentor. Maybe he or she needs assistance with the volume of work. Maybe someone else can take one item off the plate to give the person a couple of extra days. Be honest with the schedule. The more truthful the schedule is, the more flexible your team will be.

A schedule is knowledge. Yes, it's a human construct and therefore ultimately flawed, but it's still the best way project managers have to understand what lies before them. It's your map through the jungle and you'll be glad you have it even if a couple of the trails are mislabeled. With this information you'll be able to stretch your team into shapes and forms everyone around you will claim are impossible. And even if you do miss a milestone, a well-built schedule will help everyone believe you're only missing one instead of the constant slide most doomed projects face.

My next article (available at [www.gamasutra.com](http://www.gamasutra.com)) discusses process and useful ways to implement your new schedule, so your team can continue breaking new ground in game development while you keep the communication nightmare off their backs. Now get back to work, you have a schedule to write!



A segment of the Microsoft Project schedule used for developing COUNTER-STRIKE for Xbox.



## Game Developers Conference

# 2004 PREVIEW

### The Set-Up

**T**hrow together more than 10,000 attendees, 400 speakers, 200 exhibitors, hundreds of journalists, and a doomed San Jose downtown resigned to being overrun for a week, and a Game Developers Conference is born. The GDC (which is produced by the CMP Media Game Group, which also publishes *Game Developer*) is now in its 18th incarnation, facing the continuing challenge of offering up structure to the uninitiated along with fresh, unexpected experiences for the wizened industry veteran.

Recent forays into specialized mini-conferences, such as GDC Mobile (returning this year March 25 and 26) and the IGDA Academic Summit, have attracted new faces to the event, but not to be lost in the whirlwind of activity is people's fundamental desire to connect meaningfully with like-minded colleagues. Bonds forged in months and often years of newsgroup and e-mail correspondence might culminate in face time meticulously planned or happily spontaneous, while new connections help attendees move forward.

Whether you're one of the old-school GDC alumni or an incoming freshman, let our guide point the way to some of the interesting people and events at the 2004 Game Developers Conference.

## GDC TOP FIVE

### 1. The 4th Annual Game Developers Choice Awards

Every year sees more game awards doled out, from a wider variety of sources. Still, the GDC Awards, presented by the IGDA, remain the only major venue where developers are honored by their peers and get to address them as such. The posthumous presentation of last year's Lifetime Achievement Award to the family of Nintendo's Gunpei Yokoi left nary a dry eye in the house. Don't miss the presentation of the 6th Annual Independent Games Festival as well. **March 24, 6:30–9:30 p.m.**

### 2. Experimental Gameplay Workshop

The EGW started out two years ago as an unknown side session, but the room quickly overflowed, proving that despite rampant sequelitis there are plenty of people in the industry who care about innovating within the interactive medium. Organized by *Game Developer's* own "Inner Product" columnist, Jonathan Blow, the EGW has quickly become a must-see for wet-eared newbies and hoary veterans alike. **March 25, 3–6 p.m.**

### 3. Gamehotel: Games and Digital Pop Culture

Gamehotel is a brand-new event to GDC, as its organizers, Paris-based TNC Network, further their self-stated goal of "pointing the direction to the future of innovative and diverse interactive entertainment" by going straight to the creatives at GDC. Combining presentations and demos from such industry darlings as Tetsuya Mizuguchi (*Rez*) and Masaya Matsuura (*MOJIBRIBON*), emerging-platform game designers, and a menagerie of wacky Asian action-figure makers, the event promises an eye-opening experience. **March 25, 6–8 p.m.**



### 4. John Gaeta Visual Arts Keynote

Anytime a visionary, artistic speaker promises a "stream of consciousness-style discussion" on any number of multiple topics, you know you're in for either one of the best or worst talks of your life. Let's hope the plot of his presentation, which may or may not include "creative empowerment for the average Joe," "schooling your grandparents on a Playstation 3," "how to properly set fire to hard drives," "the death of *The Matrix*," and "some other visual effects stuff" holds together better than the trilogy's. **March 26, 12–1 p.m.**

### 5. Developer Business Summit: An IGDA Think-Tank

The days of ham-handed business management are numbered in today's high-stakes game development industry, yet proven, replicable strategies remain elusive. This in-depth two-day summit brings business leaders from development studios big and small together with publishing insiders to help plot a course for business success in game development. **March 22 and 23, 10 a.m.–6 p.m.**

## Who's New

**M**any interesting speakers are making their first GDC presentations this year, including:

**John Carmack.** Though his programming-track keynote topic remained at press time as elusive as id's next *DOOM* installment, thousands will no doubt turn out to see what this game development icon has to say. As befits this coding wizard-cum-aerospace dabbler, prepare for some actual rocket science.

**Eiji Aonuma.** The designer/director of numerous *LEGEND OF ZELDA* titles, including the most recent *WIND WAKER*, arrives on GDC's shores to discuss "The Evolution of a Franchise: *THE LEGEND OF ZELDA*." Anyone trying to move a mature franchise forward in today's competitive marketplace can learn something from the venerable *ZELDA* series.

**Masahiro Sakurai.** Late of HAL Laboratories, Sakurai (*KIRBY'S DREAMLAND*, *SUPER SMASH BROS.*) is now taking his show on the road, by developing for multiple platforms and talking openly about his design work in "Game Design: Risk and Reward." Risk: Do players really want to beat the snot out of cute little Pikachu? Reward: Yes, they most certainly do.

**Yannis Mallat.** Can scrappy young teams of can-doers really still give their all and turn out amazing games? Ubi Soft's Montreal-based *PRINCE OF PERSIA: THE SANDS OF TIME* team says *oui*. Executive producer Mallat reveals how the magic came together in his lecture, "Reawakening a Classic: *PRINCE OF PERSIA* Case Study."

**Kenji Kaido and Fumito Ueda.** *ICO*'s producer (Kaido) and director/designer (Ueda) reveal how they set out to make a very different game and bring never-before-seen attention to detail to the medium. With their backgrounds in game production and fine arts, they show, in "Game Design Method of *ICO*," how methodology and artistry can work together to create unforgettable games.

## Our GDC Picks

The entire *Game Developer* editorial staff will be at the show, and we're all putting together our own agendas. Although session times and dates were not final at press time, here are some of the goings on that Managing Editor Jamil Moledina, Departments Editor Kenneth Wong, and Product Review Editor Peter Sheerin are planning for.

### JAMIL'S GDC PICKS

#### How to Write an Unforgettable Story

*John McLean-Foreman*

Evocative storytelling is essential to memorable entertainment. As a moonlighting novel writer and film critic myself, both professional and personal interests compel me to attend this one-day tutorial.

#### A Peek Behind the Shoji: The Japanese Videogame Market Today

*Ryochi Hasegawa*

For years, Japan was the Mecca of game innovation. While rumors of its game market's demise may be premature, I wonder what it takes to chart a game through the shifting generalizations.

#### Interfacing with Hollywood: Challenges and Opportunities

*Keith Boesky, Leonard Grossi, Jason Rubin, Larry Shapiro*

Unfortunately, the movie industry has no USB port. But they do have representatives on this panel. I'm hoping they'll offer some insight into creating mutually beneficial licenses.

#### The Secret of PAC-MAN's Success: Making Fun First

*Toru Iwatani*

Why couldn't millions around the world, including myself, stop playing this game? This session doubles as therapy.

#### The History of Animation

*Phil Tippett*

Salvador Dali traveling through time to talk about surreal game art would be slightly less interesting.

### KENNETH'S GDC PICKS

#### What to Do When All Goes to Hell

*James Gwentzman*

Considering the erratic nature of game development, it's never too cautious to be prepared for the worst-case scenario.

#### IGDA Women's Group Gathering

*Jessica Lewis*

This is a great opportunity for me to learn more about the gripes and concerns of an underserved sector of developers. Plus, I'm single.

#### Developing a Massively Multiplayer Game

*Raph Koster, Rich Vogel, Gordon Walton*

It'll help me understand what it takes to efficiently manage the massively multi-developer team (100+) required to create such a game.

#### Immigration for Foreign Games Professionals in the Age of Homeland Security

*Ron Rose*

Rose's tips on compliance issues concerning hiring and contracting foreign talent should give me a better picture of what international game development teams face today.

#### Acting for Animators

*Ed Hooks*

I'm interested to see how Hooks distinguishes animation invested with emotion from animation devoid of motive.



### PETER'S GDC PICKS

#### Real World Multi-Threading in PC Games

*Aaron Coday, William Damon, Maxim Perminov*

While the potential for enhanced performance is quite real, so is the potential for diminished performance if done poorly or mismatched with all the CPU features.

#### Developing Wireless Location-Based Games

*Jay Aguilar*

This could create a new genre of videogames, and I'm curious to see if its use here is better than the horrid experience I've had with location-based WAP content.

#### Advanced OpenGL Tutorial

*Cass Everitt, Simon Green, Evan Hart, Bill Licea-Kane, Rob Mace, John Spitzer*

Not all games revolve around Direct3D, and the coming of the OpenGL shading language is definitely something to keep an eye on.

#### Advanced Visual Effects with Direct3D

*David Gosselin, Jeff Grills, Shawn Hargreaves, Richard Huddy, Gary McTaggart, Jason Mitchell*

This all-day tutorial from the leading experts on the Direct3D technologies in DX9 should teach you about the latest techniques for eye-popping imagery.

#### Government Simulation in 3DS Max

*Brian Blau, Stephen Langmead, Mike Rasmussen, Douglas Whatley*

The use of videogame technology in government applications is an opportunity for growth, and seeing how the rapid rate of technological change in games mixes with the government's pace may be illuminating.



## ROUNDTABLE ROUNDUP

Roundtable sessions are where the massive scale of GDC gets back to more intimate, grassroots discussions. Many attendees carry away their best GDC memories from the spontaneous but profound conversations around roundtables. The outcome depends on who shows up, making some sessions inimitable sources of inspiration, and some of them poorly attended duds. Here are a few sessions that ought to provoke lively debate.

### Quality of Life: The Next Step.

If moderator François Dominic Laramée successfully moves the discussion beyond cathartic renditions of crunch-mode horror stories, developers can share ideas on how the IGDA should steer its efforts to promote better balance between a game development career and life outside the office.

### User Created Content: Is It Worth It?

As project director of BioWare's NEVERWINTER NIGHTS, moderator Trent Oster should have a good perspective on this contentious issue.

Players want to get more, developers have less to give. But is putting tools and creative power in your customers' hands good for developers and their prized brands?

**By the Books: Solid Software Engineering for Games.** Now in their third year, these popular sessions (one on each conference day) attempt to tackle game programming's \$64,000 questions. Moderator Noel Llopis enables participants to compare strategies for language use, documentation, testing, tools, and more. Newer methodologies such as extreme programming are also discussed in a game development-specific context.

**The Publisher's "Rules of Acquisition."** Tantalized by the idea of a juicy sellout? Just want to be prepared on the off chance that a publisher shows up at your doorstep with sacks of gold-pressed latinum for you and your hardworking partners? Attorney Tom Buscaglia hosts a free-for-all where non-Ferengi types can discuss how to successfully jump through the business hoops of acquisition without tripping up.



## GDC at Work: Deploying an Open Interactive Audio File Format

Imagine where 3D graphics development would be without the advent of OpenGL and you can understand the impetus to create a nonproprietary, royalty-free interactive audio standard. Members of the Interactive Audio Special Interest Group (IA-SIG), in tandem with the MIDI Manufacturers Association, are pursuing just such a goal in the ongoing development of Interactive XMF, the eXtensible Music Format for interactive audio, which they presented at last fall's Audio Engineering Society (AES) Convention.

Chris Grigg, George Sanger, and Martin Wilde will host an hour-long session, "Cross Platform Audio Using Interactive XMF," to present the current status and goals of the effort, offer opportunity for community input, and drum up support from developers who they hope are as tired of reinventing the wheel as they are. Now that audio is becoming more of a selling point for games, reducing inefficiencies and rework in audio develop-

ment can benefit many developers, from the creative visionaries to the bean-counters. For more information, see Linda Law's Gamasutra article on the subject, at [www.gamasutra.com/resource\\_guide/20030528/law\\_01.shtml](http://www.gamasutra.com/resource_guide/20030528/law_01.shtml).

## PSP Peek

Well before Sony is expected to unveil PSP plans at E3 formally in May, SCEA's David Coombes and Peter Young will attempt to demystify Sony's new machine for its new and hopeful developers in their presentation, "Programming for PSP." No doubt they'll go over high (and perhaps some low) points of the specs in detail, offer development resource scenarios, and outline Sony's developer support plans for the device. They're also planning to address how the machine's graphical capabilities might affect handheld game design, making this session a prime stopping point for game designers and producers as well. Won't you throw in some free samples, guys? *EF*



# Bizarre Creations' PROJECT GOTHAM RACING 2

**GARRETT YOUNG, MARIO RODRIGUEZ, AND CHRIS PICKFORD |**

*Garrett's eight-plus Microsoft years span from testing NBA FULL COURT PRESS to exec. producing the PGR series with Bizarre. Mario began testing games at Microsoft as a 2002 University of Miami grad, recently working on PGR2 and RALLISPORT CHALLENGE. Chris has been designing and testing at Bizarre for four years, shipping the PGR series and FUR FIGHTERS.*



## GAME DATA

**PUBLISHER:** Microsoft Game Studios

**FULL-TIME CORE TEAM:** 40

**MAX TEAM SIZE (including test, licensing, localization, and other support resources):** 102

**LENGTH OF DEVELOPMENT:** 2 years

**RELEASE DATE:** Nov 18, 2003

**PLATFORM:** Xbox

**DEVELOPMENT HARDWARE:** Pentium 600MHz–2.4GHz machines with 256–1024MB RAM, GeForce 2–4 series and ATI Radeon 9700 Pro video boards.

**DEVELOPMENT SOFTWARE:** Microsoft Visual Studio .NET, Microsoft SourceSafe, Alienbrain, built-in 3D Editor, Softimage XSI, Araxis Merge 2001, SoundForge

**PROJECT SIZE:** 37,174 files, 219,538 lines of code, 41GB of data

**P**ROJECT GOTHAM RACING 2 is the sequel to the best-selling Xbox racing game, developed by Bizarre

Creations (Liverpool,

England) with production and publishing support from Microsoft Game Studios.

Our two teams rolled directly into production after finishing international versions of PGR in February 2002. We significantly expanded the scope and quality of the combined team, bringing on new artists, programmers, and testers. Our ultimate goal for this project was to create a AAA PGR title for the 2003 holiday season built upon the fundamental strengths of the PGR franchise and innovate in our use of the Xbox's online system, Xbox Live. Given our on-time delivery and the game's 92.4 percent average score from over 70 reviews (referenced from [www.gamerankings.com](http://www.gamerankings.com)), we feel that we

achieved our goals.

We owe that success to the strength of our people and the clarity of our challenge. Smart, effective, hard-working people are critical to achieving any worthwhile goal, and our two teams had that in spades. Though there were some disagreements and late changes in tactical direction, everyone on the team was always working toward a clear overall strategic vision for the project. A key ingredient to our ultimate success was the strong relationship between developer and publisher: matching Bizarre's design, technical, and artistic strengths with Microsoft's strengths in testing, licensing, usability, creative writing, and production management. Without an extremely high level of trust, we would not have been able to maximize the efforts of each team, and PGR 2 would not have been as strong.

Our hope is that this postmortem can



provide some insight into how we worked together to build PROJECT GOTHAM RACING 2—the things that worked well, and the things we would do differently if we had to do it all over again. With luck, we hope other teams will be able to apply these lessons to improve their processes and avoid some of our pitfalls.

## What Went Right

**1. Strong early vision for innovation.** In our efforts to build on the market success of PGR, we knew it was critical to stay true to and build on a formula gamers loved. We decided to greatly expand the number and diversity

of cars (from 25 to 100) and cities (from four to 11). Our artists invested significant time in researching routes through cities we felt were interesting, recognizable, and fun to drive. As we broadened the scope of the car list, we added new car categories like classics, muscle cars, supercars, and even SUVs. The designers also expanded the Kudos system to increase the reward for skillful driving, adding rewards for taking a “good line” through a corner, drafting, and navigating track sections cleanly (without hitting walls).

But as a new PGR game, we knew we had to continue to push that spirit of innovation. Though Xbox Live was an unproven and unknown technology during our initial design planning, we committed to pushing the online frontier in

PGR 2. We bet gamers would love racing online against their friends, and we decided to incorporate Xbox Live Scoreboards for each race in our game. These interactive high-score rankings allow every gamer with an Xbox Live account to post a race result, and allow the top 10 racers to post their actual race ghost replay for anyone to download and watch (or race against).

There were major challenges inherent in each decision. To build all the new cars and cities, we virtually doubled the size of the original art team, which also increased the challenges in team management and communication. Relying on unfinished technology from external teams created a large bottleneck in our production schedule, as we awaited their





**ABOVE.** A Porsche Carrera GT edges out a Saleen S7 on the Sydney waterfront.

**LEFT.** A classic Porsche Carrera purrs loudly on a bridge in Yokohama.

**PREVIOUS PAGES.** A Ferrari Challenge Stradale scares off tourists at the Duomo in Florence.

deliverables. We chose to accept these challenges head-on, given our vision to expand the scope of the game beyond the original title.

## **2. Reducing worldwide management.**

The Bizarre Creations team included all of our core developers, artists, game and sound designers, and production staff. The Microsoft team in Redmond, Washington included many production and design support staff, licensing managers, a full test team, and the marketing team. Given the importance of our international release, we had localization staff working full-time in Japan, Korea, Taiwan, and Ireland. We also employed 3D art vendors in Australia and England, and translation

vendors in France.

During production, members of the team visited locations all around the world to gather research, reference material, and recordings. We shot thousands of photos and hundreds of hours of video in each city. We recorded real DJs in Moscow and Tokyo. We even made a special trip to the Promised Land, visiting the Ferrari plant in Maranello, Italy to photograph and record engine audio of the Ferrari Enzo before it was released to the public. To borrow an old British saying, the sun never set on the PGR 2 team.

Managing such a global team created many problems in communication and schedule management. Our approach to solving these problems was to actually reduce, rather than expand the amount of management. We built up strong communication channels between all members of the team, removing the communication bottleneck that can occur at the producer level on game projects. All members of the production support staff at Microsoft were empowered to interact directly with all of their peers and other members of the Bizarre team. The Redmond testers and Liverpool developers interacted directly through the bug

database, e-mail, and phone calls. The Liverpool art team worked closely with the Redmond licensing managers on approvals and change requests from vehicle manufacturers and other external licensors. All members of our international localization teams interacted directly with our UI developer.

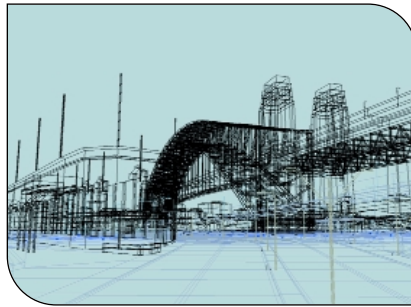
As most teams do, we also planned goals and deliverables for each milestone over the life of the production schedule. However, it was the people we had in place and our open communication channel team-wide that were the greatest contributors to our ability to resolve issues quickly and hit our aggressive holiday release schedule.

## **3. Proving stable online game-play early.**

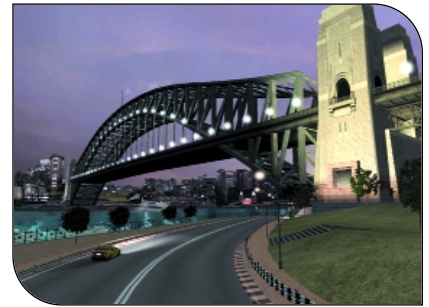
Online multiplayer is frequently the highest risk area for any game, since multiplayer features can be the hardest to implement and require significant optimization and tuning. We addressed this problem early by implementing the bulk of our network code, physics optimizations for interpolation of car speed and trajectory across all boxes, and support for all Xbox Live features by early April, 2003, five months before release. Our overall multiplayer execu-



**LEFT.** A photograph of a bridge in Sydney.



**MIDDLE.** A wireframe version of that bridge.



**RIGHT.** An in-game screenshot of the bridge.

tion was relatively smooth, with only performance and voice issues to resolve during our final code optimization phase.

We were able to achieve positive results by hiring a strong network programmer, working closely with the Xbox Live team to fully understand the new technologies as they were being built, and synchronizing our implementation and test processes. Implementing a major set of Xbox Live features to the requirements of the Xbox certification team required educating our team immediately. This investment paid off later by allowing us to quickly implement new APIs as Xbox Live 2.0 features such as stats with attachments (to support upload of ghost replays) were completed late in our project schedule.

Testing's early involvement quickly identified lag and interpolation problems, and allowed us to troubleshoot new Xbox Live features such as friends, voice, and Scoreboards during initial implementation, which allowed the development team to stabilize new features earlier.

As a result of proving multiplayer stability early on, we were able to eliminate a major risk to our production schedule and focus our efforts on other gameplay, tuning, and polishing issues during the endgame.

**4. Driving user feedback into product design.** Designing a game under tremendous time pressure can often create a myopic approach to interface design and play balance. The team is so close to the game during development that objective evaluation of the inherent challenge and usability becomes difficult if not impossible. Schedule requirements limit valuable iteration time, and members of the team cannot represent the diversity of skill across

the spectrum of all end users. The theory “a user is only a ‘new user’ once” presumes everyone in your audience will be willing to struggle through a confusing interface and unbalanced difficulty levels to experience and enjoy your game vision.

We had success in addressing these challenges on PGR 2, but not without serious investment of people and iteration time over the last few months before release. We spent literally hundreds of hours hand-tuning each car, to balance realistic handling with ease of use. We used the Microsoft usability and consumer playtest labs extensively to seek out feedback from racing gamers, and see first-hand their initial experience and difficulty navigating our interface, understanding the Kudos system, and achieving the micro-goals of each race mode.

Aside from finding and fixing all functionality and content bugs, our greatest efforts during the endgame were put towards balancing our gameplay difficulty. Almost everyone across the teams in both Liverpool and Redmond—and many others not already on the PGR 2 team—spent time providing play-balance feedback on specific challenges, each of the five difficulty levels, and the overall rate for unlocking cars. Considering the scope of the game, the size of the audience, and the concerns we had about exposing “golden paths” (shortcut cheats to high scores) in the scores and ghosts posted to our Live Scoreboards, we were very happy with the results achieved during our intense and collaborative tuning period.

**5. Effective licensing management.** Real-world authenticity is a core characteristic of the PGR franchise: real cars, real cities, real radio stations with real DJs, and real music from real bands. Unlike movie makers, game mak-

ers are required to get contractual approval with the owners of each logo and likeness before releasing it in a game.

With 11 cities (including a real-world race track), over 100 cars, unique DJs for each of our 33 radio stations, and over 300 songs, this was a monstrous task. We employed a team of five licensing managers over the course of the entire project to own and execute on this task, establishing contacts and maintaining relationships with all appropriate parties, facilitating review of all in-game assets, and working closely with legal counsel to close down each contract.

The licensing team's contributions were vital. Not only did they enable our artists to fully realize the authenticity of each city, they also secured the appropriate rights, without which we would not have been able to include Ferraris, Porsches, BMWs, or any of the other real cars in PGR 2.

## What Went Wrong

**1. Synchronizing production deliverables worldwide.** As mentioned previously, managing the contributions of our worldwide team was a great challenge, and we feel that process went well overall. However, as with any huge challenge, there were major problems that surfaced in some core areas of production.

Our original plan for addressing the problem of recording, processing, and implementing source material from our worldwide car list was to distribute the workload and take advantage of our global resources. This ended up causing more problems than it solved. Crucial implementation and tuning time for each car was sacrificed, as all cars were not



A Porsche 911 GT1 burns rubber on the streets of Chicago.

recorded to identical specifications and many cars were recorded very late. Additionally, an essential member of the sound design team moved to California and attempted to continue to fill the role part-time, adding friction to an already weak process. A related problem also occurred with the DJ scripts we recorded in each local city, as long contract and recording schedules delayed implementation beyond our desired dates.

This failure taught us two lessons. First, we needed to better synchronize content creation and implementation in the future. Second, we were reminded of a challenge all Microsoft-published projects face: synchronization of the implementation and test teams.

Though all the developers, artists, and designers at Bizarre were working in real time within the same bug database as our Redmond testers, licensing managers, and international localization teams, we did experience many setbacks. At 4GB, our transfer time for builds was substantial, and we were forced to change file transfer tools three months before release. Some obvious bugs in Redmond were difficult to reproduce in Liverpool, as the testers verified builds that were at least a full day

behind development. The 5,000-mile distance between teams greatly increased the risk associated with any last-minute file changes at the end of the project.

Though we made some great strides to better synchronize implementation and testing, our challenge moving forward will be to hit our functionality and content deadlines more effectively, increase the stability of the build process, and to increase the scope of smoke tests in Liverpool before builds are sent over to the Redmond test team.

## 2. Design took too long.

Incomplete design had the single largest impact on the slide of deliverables throughout our project schedule. Though we all understood and agreed upon the fundamental vision for the franchise, we started our PGR 2 design document very late, and we were still updating this document after feature freeze and E3 as new ideas cemented. We ended up overhauling many elements of our design several times, such as the user interface, Kudos reward values, and overall game structure.

There were many critical factors that extended our design phase. Our designers were spread too thin early on, and we

pursued many different paths and game modes before hitting on a concrete plan. Martyn Chudley, the creator of the PGR franchise and head of Bizarre Creations, played a critical part in stepping in to narrow the focus of the overall game design in early 2003.

Iteration on the handling characteristics of each vehicle proved incredibly challenging, and was completed very late in the schedule. As with any simulation-quality racing game, each change in vehicle handling has a significant knock-on effect on each car's usability, the performance of the AI, the class and competitive categorization of each car, and the difficulty-level setting for each race.

Quadrupling the number of cars from PGR to PGR 2 more than quadrupled our vehicle-tuning time, as the broader scope of vehicle content required far more diligence, testing, and tuning between individual vehicles and classes. We underestimated the initial scope of this effort.

To combat these problems we expanded the size of the test team, pulled in additional designers from other projects, and hired a small team to execute specifically on the play-balancing task. Moving forward onto future projects we will seek out additional



**LEFT.** The Bizarre Creations PGR 2 development team. **RIGHT.** The Microsoft Game Studios PGR 2 production and publishing team.

solutions, such as fleshing out key sections of the design document earlier, prototyping new gameplay elements in offline builds, and proactively scheduling our extensive play-balance efforts. Iteration is a natural part of game development and we're pleased with the results, but getting there was no easy task.

**3 • Relying on new technologies from external teams.** PGR 2's design called for online features to be integrated within almost every area of the game. These features required APIs that were still in development by the Xbox Live team as our schedule progressed, raising many significant obstacles and time constraints.

While our Scoreboards were an extension of the system shown in the 2002 Xbox Live starter kit's MotoGP demo, we utilized them to a far greater extent than on other titles. Coupled with new Xbox Live features, such as stats with attachments (such as ghosts), and the high volume of Live users we anticipated, we knew that we had placed a big bet on bringing these emerging technologies to a usable and stable state. If any piece failed, the system would have failed. Implementing the new attachments APIs for uploadable and downloadable ghosts presented us with unpredictable problems, as we were pioneers in this space. The Xbox ATG team was extremely supportive, but they were also pushing hard to complete features for their own deadline.

One example of a problem we should have been able to avoid was optimization of our interaction with the Live Scoreboards. During research done by the Xbox Live team late in our schedule, they found our code was making far too many calls to their Scoreboard servers, a capacity problem their servers would not have been able to handle after release. Though we fixed this problem, it raised significant production fears at the end of our schedule.

In relying on critical technology from external teams, we have learned the importance of allocating an adequate schedule buffer to accommodate unforeseen problems, maintaining strong communication with all dependent parties, and gaining a deep understanding of the technology.

**4 • Stability of the ghost replay system.** In supporting a feature where any Xbox Live user worldwide could

upload a ghost replay, we needed to be able to guarantee each ghost would be a perfect replica of the actual race result. The critical nature of this feature caused us to dedicate significant attention from our test team.

The replay subsystem served as the underlying framework for recording the ghost data. This legacy system was both complex and difficult to consistently debug. One late night two days before RTC (release to certification), our testers were in heated competition, challenging each other's high score ghosts on one



A BMW M3 shines through a rainy night in Edinburgh.

of the arcade cone challenges, with several lead changes over the course of a few hours. As one score became very difficult to beat, one of the testers noticed the total Kudos shown in the ghost replay did not match the value on the Scoreboard. Despite all the months of non-stop testing, a strong game only hours from shipping still had a very critical bug somewhere in the replay system.

Competitive gameplay is a very valuable part of the testing process during the endgame—this was how gamers were going to be playing our game! Although we fixed it, we could have found that bug earlier, and in the future we will also push to create more automation for critical areas such as this, including hooks for specific test scripts, boundary, and stress conditions. We'll never be able to perfectly emulate the gameplay of millions of gamers, but by prioritizing our focus, expanding our automation suite, and increasing the size and scope of our endgame “bug bash” efforts to broader internal groups and teams, we'll be better armed to find and fix all show-stopping bugs before release.

**5. Build process and source control.** We had a substantial number of assets to manage during content creation at Bizarre Creations, including over 100 cars with 3D model and dynamics/handling files, 11 constantly evolving

city models, and over 8,000 audio content files. This caused tremendous confusion at the end of the project, as our processes were not originally planned to handle this scope of code and content.

At the beginning of the project we had multiple teams uploading to one game image. This led to significant content incompatibility problems in the build, such as cars using incorrect engine audio, city tracks without track-side barriers, and old bugs re-appearing as old content over-written as new. We planned to solve this by splitting up the original game image into separate images for city data, audio and radio content, car content, and all source code. We also planned to create a unique test image, where all content would be copied before release to the Redmond test team.

However, splitting the process this way ended up causing more harm than good. During the endgame, as the Bizarre team was doing builds every few days, the test image had to be manually updated frequently. To ensure consistency and speed, the team created a step-by-step process and set of batch files, to be run in a specific order each time. With a game so large and server space at a premium, we could not use Alienbrain or another file management package. We were forced to dedicate one person to this manual drag-and-drop process for creating the test image.

Disasters began to strike as the build

process began to take longer. Everyone was working very long hours, late checks were made after build smoke tests, and additional steps were added in managing retrieval of assets from multiple game images, complicating the build process. The test image would often be pulled together and posted to the secure FTP site, only for the Redmond test team to find the build would not run when they came in the next morning, losing a day of testing on the latest bits.

Moving forward, the Bizarre team will stick to two game images—one image for the team to post all code and content, and another image for all tested, shippable content. With less moving parts, we expect the process to go more smoothly on future projects.

## Final Lap

In the end, we are all very proud of the results we were able to achieve in PGR 2, and we hope gamers are too. As a team, we were able to deliver on our vision and critical priorities for the game. We were able to increase our quality bar by maintaining a strong balance between building upon the core fundamentals of PGR gameplay and breaking new ground in online multiplayer and scoreboards. We were also able to deliver the game to gamers on time.

However, no project is perfect, and we certainly had our share of hurdles to overcome, many self-imposed. We grew a lot as a combined Bizarre Creations and Microsoft Game Studios team between PGR and PGR 2. Our challenge will be to continue that growth in the future, to learn from the success and failures of our past, and to work together to overcome future problems as they arise.

Looking back, the key to our success was the team involved in bringing the game to life. PGR 2 was built by smart, hard-working people working together effectively with a high degree of trust, open communication channels, and a clear vision and goals. Easy things to say, but the magic was in the execution, as it will likely continue to be in the foreseeable future. 🏁

# Marketing: Packed in Together

**F**or many game developers, “marketing guy” ranks near the top of the necessary evil scale—right after “entertainment lawyer” and “retail buyer.” Of course, marketing is a necessary evil. Whether you’re working with the industry’s top publishers or just starting out, at some point you’re going to have to understand, embrace, and ultimately partner with that marketing beast demanding attention in the dark corner of your business aspirations.

## First Step: Understanding the Beast

**M**arketing is communication—it’s that simple. When a company puts an emphasis on marketing, it is in fact putting an emphasis on communication. Understand this, and you realize the old saying “we don’t need marketing—our game will sell itself” is tantamount to saying, “we don’t need to communicate with people—our game will do that on its own.”

It’s okay to think of marketing in simple terms, but it’s dangerous to think of it as a simple endeavor. Consumer segments have eroded and splintered into an expanding number of targets. Adopting a multi-layered campaign, utilizing well-placed snipers to support the heavy artillery, is necessary to get your message across effectively. Increasingly, the audience is turned off by sledgehammer communication tactics; today’s consumers are empowered consumers, and their patience is evaporating with every ad promoting the “most awesome,” “most advanced,” and—worst of all—“most immersive” game ever released. Today, your marketing message has to be razor sharp, it needs to be emotionally creative, and it better be intelligent enough to establish a meaningful connection with consumers, or your margin for success will be reduced to a pinhole.

## Next Step: Embracing the Beast

**P**utting an emphasis on effective marketing is a fact of life in any industry competing for consumer dollars, so why



Illustration by Greg Hargreaves

fight it? If you haven’t done so already, it’s time to embrace the beast. That doesn’t mean you should develop the marketing; it means you should treat marketing as a priority. In the film and music industries, for example, movie studios and music labels provide the actual marketing machinery, but the successful directors, actors, producers, and musicians make marketing a priority throughout their careers.

## Final Step: Partnering with the Beast

**U**nlike the movie studios, which are similar to one another in the way they integrate product development and marketing, not all game publishers are alike. Some demand interaction between development and marketing; others believe separation allows for the least amount of interference. In truth, marketers need the partnership—they need to know the product intimately if they’re going to establish effective positioning and communication. And developers need the partnership so they can create the appropriate fuel to drive the communication machine, and ultimately deliver the promise of the marketing claim.

There are three keys to a true partnership. Number one: share information. Numbers two and three: share information

*continued on page 71*

continued from page 72

early and often. That means sharing original pitch materials, concept art, napkin notes, whatever you've got. And, yes, that means sharing the design document—preferably before it's


finalized. Your marketing partners need more than a list of features; they should be in on the discussion about character traits, story arc, level objectives, and everything else from the HUD layout to the number of hours the average player needs to complete your game. In turn, you should get in on the discussion about consumer trends, competitive products, market conditions, and retail initiatives. There is no such thing as starting too early on this process.

Through this process, marketing and development should identify the vital “hooks” to your game's positioning as early as possible—to make sure the features that support the hooks don't disappear during production. Through this process, the two groups should identify product placement, promotional partnership and custom retail opportunities that are organic to your game, not just slapped on at the last moment. Through this process, the costly and unsettling disconnects that occur between the marketing message and the final game can be avoided. And

## Partnership means meeting regularly, establishing positioning, and developing goals

opportunities can be seized in a timely manner, rather than frantically chased or missed altogether.

Cooperation isn't enough. Cooperation is the marketing guy politely e-mailing a request for assets that get

incorporated into your milestone schedule. Partnership is marketing and development meeting regularly, establishing positioning for the product at the earliest possible stage and developing goals for communication initiatives to support the positioning. This shouldn't be a relay race, with each person performing his leg of the race separately. It's a bobsled run, with everyone packed together rocketing down a slippery course that demands a unified effort from start to finish. It may seem like an uncomfortable notion to some, but it's the only way to keep pace with the competition and maximize your game's chances for success. 

---

**Craig Relyea** | *Craig is Executive V.P. of Marketing for Creative Domain, a leading Hollywood entertainment marketing agency, where he runs the Interactive Entertainment Division and helps develop game campaigns for the industry's top publishers. Craig is also the former head of Marketing for DreamWorks Interactive and V.P. of Worldwide Marketing for Interplay Entertainment.*

---