

# gd

GAME DEVELOPER MAGAZINE

FEBRUARY 2001



# GAME PLAN

LETTER FROM THE EDITOR

## Telling Stories

**T**here have been a lot of unusual discussions popping up around game developer gatherings these days. People are talking about violence in videogames, whether or not games are an art form, how games can reach a mainstream audience, and whether the industry as a whole is doomed due to its immaturity.

We're in a stage between garage game development and mainstream entertainment art form, and the path from here to there is not clear. Game developers are casting about looking for guidance, trying out new models, and attempting to learn from the mistakes of other entertainment industries. The recent hearings by the U.S. Federal Trade Commission on violence in the media were a profound wake-up call that illustrated our growing influence in society.

Though our technology is always improving, yielding stunning 3D worlds and dramatic spatialized sound environments, an emphasis on story is developing. Developers are recruiting screenwriters from Hollywood to tune up their game concepts for maximum impact. Many game engines are now available, freeing up game development to focus more on the story and art. Game modifications also enable development without the concern for technical superiority. On most development projects, the art and design teams are now larger than the programming teams.

The steep slope of technology can only take us so far in our quest to go mainstream and be recognized as an art form. At some point the environments look the same to the player no matter how many extra polygons or texture passes you add. We've reached that point — now we must turn our focus to story. A good story is much more accessible to people than snazzy technology. A look to our past shows that in every culture there are storytellers, people who pass the history and mythology from generation to generation. They didn't have the technology we have today, and yet many of those stories have endured the ages. We are creating a whole new method of storytelling, which is a very exciting place to be. But the key to creating a new art form is to focus on the art, not the canvas.

This is one reason why the motion picture industry has made a good model for us

lately. They (for the most part) solidified their technical conventions decades ago, and have generations of experience in the art of storytelling using those conventions.

Many people over the last decade have hyped the convergence of Silicon Valley and Hollywood. They were right; it is happening. Take advantage of it by enlisting the help of excellent screenwriters, knowledgeable directors of photography, and seasoned soundtrack composers for your next project.

### This Month

**I**n this issue we highlight games for kids. What does it take to make a really great and fun children's game? A good story definitely helps. What do you do when you can't judge the quality of your game based on whether you personally find it to be fun? We detail games for kids in this month's main feature, Postmortem, and Soapbox.

We also have an introduction to game modifications, better known as "mods." Mods are undoubtedly going to continue growing in popularity and influence, and with this article, we're just opening the floodgates. Check out all the mod web sites in the article's "For More Information" section and you'll see what I mean.

Jan Kautz and crew dig into the technical details of using BRDFs in your games in this second part of their two-part article. Be sure to download their demo source code from our web site at [www.gdmag.com](http://www.gdmag.com).

We at *Game Developer* are always exploring the nooks and crannies of our industry for things that you will find interesting. With this issue we're beginning a gradual shift over the next few months to tune ourselves more to the needs of our community as expressed through reader response. Watch this space for more on our transition next month.



Let us know what you think. Send e-mail to [editors@gdmag.com](mailto:editors@gdmag.com), or write to *Game Developer*, 600 Harrison St., San Francisco, CA 94107

ON THE FRONT LINE OF GAME INNOVATION  
**Game  
DEVELOPER**

600 Harrison Street, San Francisco, CA 94107  
t: 415.947.6000 f: 415.947.6090 w: [www.gdmag.com](http://www.gdmag.com)

**Publisher**  
Jennifer Pahlka [jpahlka@cmp.com](mailto:jpahlka@cmp.com)

#### EDITORIAL

**Editor-In-Chief**  
Mark DeLoura [mdeloura@cmp.com](mailto:mdeloura@cmp.com)

**Senior Editor**  
Jennifer Olsen [jolsen@cmp.com](mailto:jolsen@cmp.com)

**Managing Editor**  
Laura Huber [lhuber@cmp.com](mailto:lhuber@cmp.com)

**Production Editor**  
R.D.T. Byrd [tbyrd@cmp.com](mailto:tbyrd@cmp.com)

**Editor-At-Large**  
Chris Hecker [checker@d6.com](mailto:checker@d6.com)

**Contributing Editors**  
Daniel Huebner [dan@gamasutra.com](mailto:dan@gamasutra.com)  
Jeff Lander [jeffl@darwin3d.com](mailto:jeffl@darwin3d.com)  
Mark Peasley [mpeasley@gaspowered.com](mailto:mpeasley@gaspowered.com)

#### Advisory Board

Hal Barwood LucasArts  
Noah Falstein The Inspiracy  
Brian Hook Independent  
Susan Lee-Merrow Lucas Learning  
Mark Miller Group Process Consulting  
Paul Steed WildTangent  
Dan Teven Teven Consulting  
Rob Wyatt The Groove Alliance

#### ADVERTISING SALES

**Director of Sales & Marketing**  
Greg Kerwin [gkerwin@cmp.com](mailto:gkerwin@cmp.com) t: 415.947.6218

**National Sales Manager**  
Jennifer Orvik [jorvik@cmp.com](mailto:jorvik@cmp.com) t: 415.947.6217

**Account Manager, Western Region, Silicon Valley & Asia**  
Robert Darden [rdarden@cmp.com](mailto:rdarden@cmp.com) t: 415.947.6223

**Account Manager, Northern California**  
Susan Kirby [skirby@cmp.com](mailto:skirby@cmp.com) t: 415.947.6226

**Account Manager, Eastern Region & Europe**  
Afton Thatcher [athatcher@cmp.com](mailto:athatcher@cmp.com) t: 415.947.6224

**Sales Representative, Recruitment**  
Morgan Browning [mbrowning@cmp.com](mailto:mbrowning@cmp.com) t: 415.947.6225

#### ADVERTISING PRODUCTION

**Senior Vice President/Production** Andrew A. Mickus  
**Advertising Production Coordinator** Kevin Chanel  
Reprints Stella Valdez t: 916.983.6971

#### CMP GAME MEDIA GROUP MARKETING

**Senior MarCom Manager** Jennifer McLean  
**Strategic Marketing Manager** Darrielle Ruff  
**Marketing Coordinator** Scott Lyon  
**Audience Development Coordinator** Jessica Shultz  
**Sales Marketing Associate** Jennifer Cereghetti



*Game Developer*  
magazine is  
BPA approved

#### CIRCULATION

**Group Circulation Director** Kathy Henry  
**Director of Audience Development** Henry Fung  
**Circulation Manager** Ron Escobar  
**Circulation Assistant** Ian Hay  
**Newsstand Analyst** Pam Santoro

#### SUBSCRIPTION SERVICES

For information, order questions, and address changes  
t: 800.250.2429 or 847.647.5928 f: 847.647.5972  
e: [gamedeveloper@halldata.com](mailto:gamedeveloper@halldata.com)

#### INTERNATIONAL LICENSING INFORMATION

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

#### CMP MEDIA MANAGEMENT

**President & CEO** Gary Marshall  
**Corporate President/COO** John Russell  
CFO John Day  
**Group President, Business Technology Group** Adam K. Marder  
**Group President, Specialized Technologies Group** Regina Starr Ridley  
**Group President, Channel Group** Pam Watkins  
**Group President, Electronics Group** Steve Weitzner  
**Senior Vice President, Human Resources** Leah Landro  
**Senior Vice President, Global Sales & Marketing** Bill Howard  
**Senior Vice President, Business Development** Vittoria Borazio  
**General Counsel** Sandra Grayson  
**Vice President, Creative Technologies** Johanna Kleppe





# FRONT LINE TOOLS

WHAT'S NEW IN THE WORLD OF GAME DEVELOPMENT | *daniel huebner*

## 3DS MAX 4 ARRIVES

**3**ds Max 4, the fourth version of Discreet's flagship modeling, animation, and rendering software, has been released. The upgrade will feature a new inverse kinematics system for character animation, new subdivision surface and polygon geometries, and new rendering abilities with ActiveShade and Render Elements. Weighted constraints, angle deformers, and real-time shaded skin are all integrated into the IK architecture. 3ds Max 4 will be available in early 2001, with a suggested retail price of \$3,495. Upgrades from version 3 will be priced at \$795.

3DS MAX 4 | Discreet | [www.discreet.com](http://www.discreet.com)



3ds Max 4 heads-up display features slider-manipulators for speedy character animation with real-time rendering window below.

## ENROUTE TARGETS PLAYSTATION 2

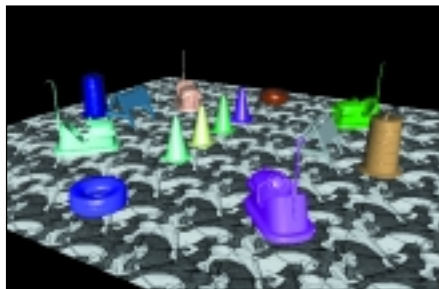
**E**nroute has joined Sony's Tools and Middleware program, authorizing them to begin offering FirstPerson toolkits to PlayStation 2 developers. Enroute's FirstPerson system combines multiple video streams captured from any outward-looking camera system and creates a self-navigated 360-degree video format for playback on PCs and entertainment consoles. Enroute claims the

FirstPerson format enables viewers to experience broadcast-quality concerts, movies, music videos, or sporting events in full motion from any angle. Enroute's SDK is available on negotiable terms, and the first game titles containing FirstPerson content are scheduled to debut in 2001.

FIRST PERSON | Enroute | [www.enroute.com](http://www.enroute.com)

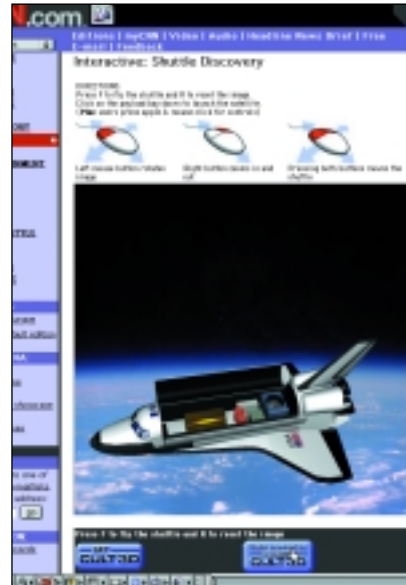
## RESEARCH GROUP RELEASES PIVOT

**T**he University of North Carolina's Chapel Hill Research Group has released PIVOT (Proximity Information from VOronoi Techniques), its software that uses multi-pass rendering to perform proximity queries between objects. These computations include collision detection, computing intersections, separation distances, penetration depth, and contact points. PIVOT balances computation between the CPU and graphics subsystems by localizing the closest features between the two objects and computing the proximity information in those localized regions. The proximity information is received from a computed distance field. PIVOT will be available in early 2001 and will be free for noncommercial use, however, commercial pricing has yet to be determined.



UNC Chapel Hill's PIVOT can turn a 3D scene in to a 2D collision problem.

PIVOT | Chapel Hill Research | [www.cs.unc.edu/~geom/PIVOT/](http://www.cs.unc.edu/~geom/PIVOT/)



The Cult 3D viewer allows users to view and move Cult3D objects on the web.

## CULT3D EXPORTER FOR MAYA

**C**ult3D Exporter for Alias|Wavefront's Maya allows users to export models created in Maya directly to Cycore Cult3D. Artists can then create interactive 3D objects for the web, Microsoft Office, and Adobe Acrobat. It is the first 3D exporter software available for Maya from a third-party developer. Cult3D runs on PC and Macintosh over low-bandwidth connections. The Cult 3D viewer works under Netscape

Navigator and Microsoft Internet Explorer, and consists of two components: the Cult3D plug-in, which allows users to view Cult3D objects on the web, and the Cult3D Designer and Exporter, which imports 3D objects and exports them to a web page, or a Microsoft Office or Adobe Acrobat document. Cult3D Exporter is available free for download on the Cycore web site.

CULT3D EXPORTER | Cycore | [www.cycore.com](http://www.cycore.com)

## TERRAPLAY SYSTEMS RELEASES SDK

**T**erraplay Systems' first SDK for PC game developers provides developers with the ability to combine Terraplay's platform with online games. Terraplay is an IP-based system designed to speed up network-based games by formatting game data for available bandwidth regardless of whether clients are using modems, broadband connections, or wireless connections. Terraplay's development kit is available free on its web site for downloading.

TERRAPLAY SDK | Terraplay Systems | [www.terrappay.com](http://www.terrappay.com)



**Financial results.** Major game publishers are continuing to feel a financial pinch. Eidos reported a \$116.4 million loss for the six months ending September 30, a figure that more than doubles the losses reported by Eidos for the same period one year prior. Though much of Eidos's poor performance can be attributed to a \$51 million write-off related to the company's failed experiment with online game retailer Express.com, Eidos failed to meet sales expectations for many of its mainstay products.

The situation is much the same at Acclaim Entertainment, which posted a major loss for its fiscal 2000, which ended August 31. Acclaim managed revenues of just \$188.6 million, resulting in a net loss of \$131.7 million. In contrast, Acclaim reached revenues of \$431 million and net earnings of \$36 million the previous year. Acclaim blamed the decline on poor Nintendo 64 game sales, which had been Acclaim's main focus, as well as delays in the introduction of new titles. Though Acclaim's operating expenses have been negatively impacted by the research cost related to next-generation consoles, the company believes that the new platforms will return Acclaim to profitability as early as next quarter.

**Hook exits.** Brian Hook has left Verant Interactive. Hook, formerly with 3dfx and id, had been working on next-generation graphics technologies for the company. Although the departure was amicable, Hook did reveal in an interview with game news site Voodoo Extreme that Verant had become too corporate for his tastes. Hook plans to take some time off from the game business before returning to open his own studio sometime this year.

**Nintendo profits.** Nintendo managed to beat many analysts' expectations by announcing strong half-year results. A 26 percent drop in Nintendo 64 game sales wasn't enough to keep Nintendo from posting a pretax profit of \$470.6 million for the April to September period. Much of this gain, however, is attributable to an increase in the appraised value of the company's assets and the dollar's relative strength against the yen. Game Boy hardware and software continued to be a bright point,



ROLLER COASTER TYCOON, one of Hasbro Interactive's titles now owned by Infogrames.

with sales of 10.2 million Game Boy units worldwide in a six month period despite component shortages. Nintendo expects to sell 23 million Game Boys for the full year, including one million Game Boy Advance handhelds following its March launch in Japan. The company did, however, cut its N64 forecast to 3.05 million units from an earlier projection of 3.5 million.

**Digital Anvil acquired.** Microsoft strengthened its ability to produce in-house Xbox titles with the acquisition of Digital Anvil. The terms of the deal gave Microsoft full rights to in-production titles, including Digital Anvil's long-anticipated FREELANCER and an unnamed Xbox project. Digital Anvil will operate as a part of Microsoft's internal studio structure, but unlike earlier Microsoft pick-up Bungie, Digital Anvil will not be relocated to the company's Redmond, Wash., headquarters. The deal does not include Digital Anvil founder and CEO Chris Roberts, either. Roberts is leaving the company to pursue other creative endeavors, though he will stay for the remainder of FREELANCER's development as a creative consultant.

Digital Anvil's president, Martin Davies, moved to Internet consultancy Sapien as vice president for games. Davies will provide strategy and implementation for next-generation game development at the former Human Code studio in Austin, Tex.

**Infogrames grabs Hasbro.** Infogrames has purchased 100 percent of Hasbro Interactive's common stock, and as part of the purchase gains Hasbro's Games.com web portal and a long-term exclusive license to develop and publish games

based on Hasbro properties. Infogrames' license on the Hasbro brands will run for 15 years, with an option for an additional five years based on performance. Among the properties changing hands as part of the deal are best-selling game franchises such as ROLLER COASTER TYCOON and CIVILIZATION, as well as successful toy brands such as Monopoly. The purchase price totaled \$100 million; \$95 million in Infogrames SA shares and \$5 million in cash. Subject to shareholder approval, both sides expect to wrap up the deal by the first quarter of 2001.

Infogrames made other company changes with the move of two senior European executives to North America. Rob Watson, Infogrames' senior vice president for worldwide licensing, has moved to the company's Los Angeles office, where he will oversee operations at Infogrames' I-Stars label in addition to his current duties. Jean-Philippe Agati, head of European publishing and production in France, has travelled to Infogrames' San Jose headquarters to take on the role of senior vice president and general manager for the company's operations there. 



## UPCOMING EVENTS CALENDAR

### GAME DEVELOPERS CONFERENCE 2001

SAN JOSE CONVENTION CENTER

San Jose, Calif.

March 20-24, 2001

Cost: Expo — \$145 and up

Conference — \$425 and up

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

### AMERICAN ASSOCIATION FOR ARTIFICIAL INTELLIGENCE SPRING SYMPOSIA

STANFORD UNIVERSITY

Stanford, Calif.

March 26-28, 2001

Cost: \$295 for nonmembers

[www.aaai.org/Symposia/Spring/2001/sss-01.html](http://www.aaai.org/Symposia/Spring/2001/sss-01.html)





## Dead Reckoning a.k.a. Motion Prediction

### Problem

When writing a network game, maintaining synchronization of game objects can be challenging. The problem is usually stated as a problem of “latency” — the delays that are inherent to all communication systems and that are particularly noticeable on the Internet.

Network latency is not as simple as it may appear; a brief review is in order. There are two related causes of latency: “path latency” and “queuing latency.” Path latency is the time it takes for a message to get from one place to another, while queuing latency is created when the sent data exceeds the bandwidth of the channel. A highway analogy is useful: path latency is the time it takes to drive from home to work at midnight — it is only a function of distance and any speed limits. Queuing latency is caused when the rush-hour traffic volume exceeds the road capacity. For the network programmer, the important fact is that the only way to reduce latency is to reduce data traffic.

### Solution

To reduce latency, predictable state changes are simply not transmitted. The classic example of this is linear motion: instead of sending frequent positional updates (“the ship is now at  $X$ ”), a position and velocity are sent less frequently (“the ship is at  $X$  and has velocity  $V$ ”); the resulting positions are estimated by extrapolation until another update arrives. This ancient navigational technique of estimating one’s position based on a known starting point and velocity is called “dead reckoning.” The name comes from the nautical technique of throwing a floating object overboard, rendering it “dead in the water,” and then timing its travel from bow to stern to estimate the vessel’s velocity.

The Dead Reckoning pattern is more versatile than simply predicting object motion. Any game variable whose state is predictable over time can use this tech-

nique. For example, say a tank locks on target to another tank and starts firing; the hit points may be decremented predictably on the client with few updates. Or, an even simpler form of the pattern is when the server orders an effect animation such as “explode at time  $T$ ” without any subsequent server updates.

### Issues

**Prediction algorithm.** It is often advisable to use different prediction functions for different situations. While it may be useful for an airplane to send acceleration along with velocity, this may be less useful for an avatar who can stop on a dime. A common adaptable algorithm is for the server to calculate the difference between the client’s predicted value and the actual value and broadcast a correction once the error passes a certain threshold. This threshold can be dynamically adjusted, depending on current bandwidth availability.

**Warping.** What if a game client was told that an enemy airplane was traveling at a given speed, but before the next update arrived the plane had radically altered its course? Upon the next update, the client must somehow get the plane to its true position. However, simply snapping it to the new location might look strange; for example, what if the predicted position was inside a mountain? The resolution to such problems is called “warping,” and there are too many resolution techniques to enumerate here.

**Time synchronization.** If one could ensure that the computers shared an identical clock, one could improve the performance of Dead Reckoning. For example, instead of transmitting “unit at position  $P$ , velocity  $V$ ,” one could say, “at time  $T$  in the future, unit will arrive at position  $P$ .” This helps reduce warping problems and is especially useful when the end of an action is important and one wants to ensure good synchronization. For example: “object will explode and at time  $T$ .” (For techniques, see under References.)

**Unreliable transport.** When implemented with non-reliable transport protocols such as UDP, Dead Reckoning algorithms must account for the fact that a previous update may not have arrived or may have arrived out of order. Occasional “gratuitous updates” must be broadcast, even if an object has not deviated from its predicted course, just in case a previous update did not arrive. The Dead Reckoning technique is useful under both reliable and non-reliable protocols, although the particular choice of algorithm could be substantially dependent on this choice.

### References

Considerable research has been conducted by the Department of Defense for its Distributed Interactive Simulation (DIS) protocol ([www.darpa.mil](http://www.darpa.mil)). Jesse Aronson discusses DIS and implementations of Dead Reckoning algorithms in “Dead Reckoning: Latency Hiding for Networked Games” ([www.gamasutra.com/features/special/online\\_report/dead\\_reckoning.htm](http://www.gamasutra.com/features/special/online_report/dead_reckoning.htm)). Also see Zack’s paper, “A Stream-based Time Synchronization Technique for Networked Computer Games” ([www.totempole.net/timesync.html](http://www.totempole.net/timesync.html)).

### Credits

Thanks to Joel Desjardins of Eternal Software for significant contributions to this pattern! 🍷

Look for Patterns on Gamasutra.com

This month’s pattern is the last which will appear in *Game Developer* magazine. Visit [www.gamasutra.com/patterns](http://www.gamasutra.com/patterns) to learn more about the Game Programming Patterns Database. Contribute to our ongoing list of game programming patterns and idioms by sending yours to [patterns@d6.com](mailto:patterns@d6.com).



# PRODUCT REVIEW

THE SKINNY ON NEW TOOLS

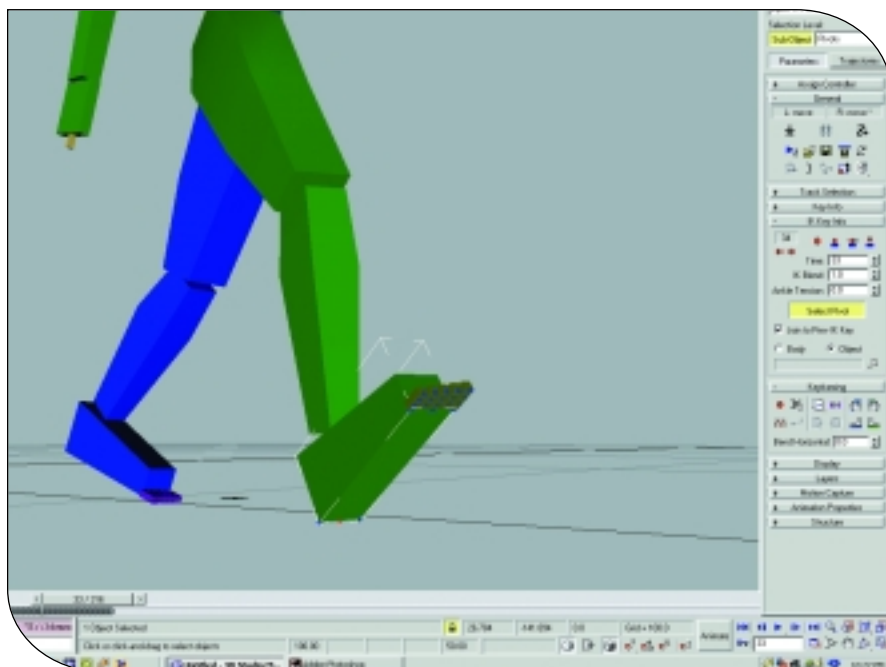


FIGURE 1. Biped's animatable pivot points (indicated in red on the foot) and new IK interface.

now much easier (the demo includes a “walking fingers” animation that would have been nearly impossible to create in Character Studio 2.x). To animators who have labored to keep a toe on the ground in the takeoff phase of the footstep, this will be a very welcome addition. A new Ankle Tension control offers a shorthand method of specifying the stiffness of the ankle (or wrist), neatly complementing the new pivot points as a tool for keeping appendages in place (see Figure 1).

Biped's user interface has also received some positive attention. Very welcome is a new option for splitting the keyframe tracks of hands and fingers (or feet and toes) from those of their parent limbs. It's no longer necessary to set a key on the entire arm to move a single finger. This makes combining complex hand gestures with larger arm motions much easier and less frustrating than before. In the biped's command panel, a new rollout makes it easier to set the IK attributes of a new key quickly, and, if desired, automatically lock it to a previous key position.

A more ambiguous improvement is an option in the keyframe clipboard to copy entire animation tracks within or between bipeds. This offers a quick way to copy animations from one biped to another, and is an invaluable aid in creating cycles.

Unfortunately, the behavior is erratic if any of the keys being copied are IK keys; IK attributes are not always copied correctly, and when the IK is pasted onto a new limb, the pasted keys point at the IK target of the original — meaning that the position of the limb may be different when pasted to another biped or an opposing limb.

## Physique

Discreet claims that the interactive performance of Physique is between three and ten times faster than in previous versions. In my tests, however, the results were less dramatic than this might suggest. A 740-polygon character playing back a simple animation went from 11.1FPS

## Discreet's Character Studio 3 *by steve theodore*

Since its debut in 1996, Character Studio (developed for Discreet by Unreal Pictures) has been an indispensable tool for animators in the game, video production, and film industries.

Character Studio 3 is the latest edition of the popular package for 3D Studio Max 3.x users. The original Character Studio consisted of two Max plug-ins: Biped, a hybrid forward/inverse kinematic skeletal system, and Physique, which performs skeletal and muscular deformation. Version 3 updates these plug-ins and adds a new set of tools (collectively known as Crowd) for applying semi-intelligent behaviors to large groups of characters and objects. Character Studio 3 also adds extensive support for

Max's scripting system, improved handling of motion capture data, and a number of workflow improvements.

### Biped Refinements

From the animator's perspective, the Biped plug-in is the heart of Character Studio, and version 3 offers a number of nicely nuanced improvements to the Biped module. The most significant improvement is the addition of user-selectable pivot points in the hands and feet. Each IK key is set with its own pivot, making it much easier to model the shifting of weight from the heel of the foot to the toe during a walk. Likewise, maintaining positional consistency as a hand moves and rotates is

STEVE THEODORE | Steve is an animator at Valve Software. He is currently working on TEAM FORTRESS 2, and can be reached at [stevet@valvesoftware.com](mailto:stevet@valvesoftware.com).

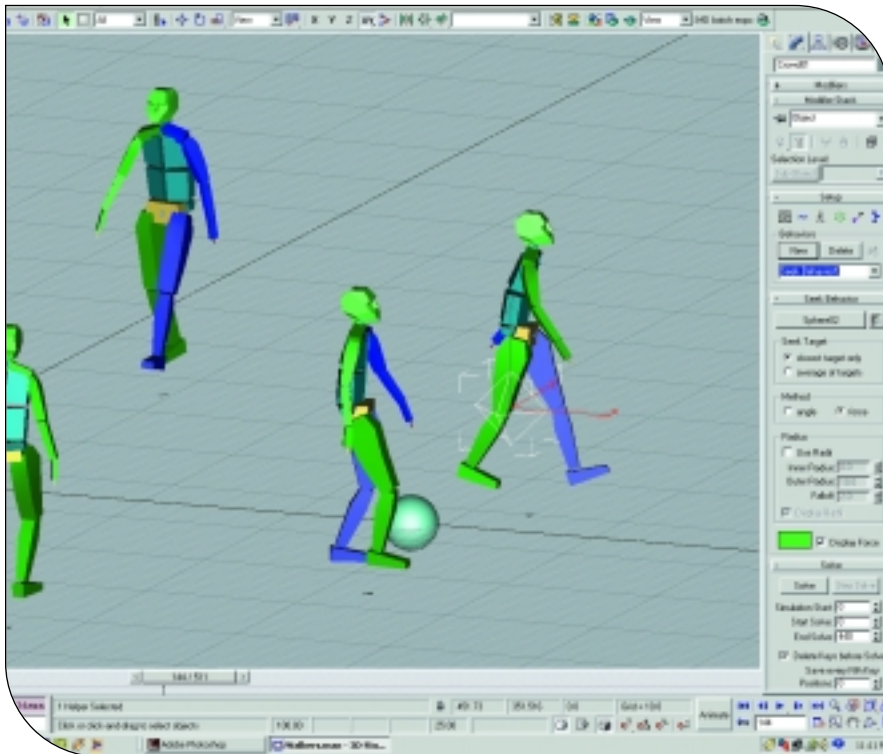


FIGURE 2. The Crowd plug-in in action, showing some of the interface.

under Character Studio 2.2 to 13.7FPS under version 3. The same file without a Physique modifier ran at 16.7FPS. These figures came from a 400MHz Pentium II with 512MB RAM and a 64MB GeForce GTS Pro; results for multi-processor systems or models with very complicated Physiques should be more impressive.

Previous versions of Character Studio required that all the bones deforming a skin be part of a single hierarchy. In version 3, the new Floating Bones option allows users to add bones that are not connected to a common root and can move independently of the other bones in the character. Perhaps more interestingly, users can add spline objects as floating bones. The splines will perform deformations along their whole length and can be animated. This can be a particularly useful tool for complicated surface deformations and facial animations too complex to handle with bones.

### 3's a Crowd

The real innovation in Character Studio 3 is the Crowd plug-in, which is

intended to aid the animation of large groups of characters. In essence, Crowd works like a smart particle system, creating groups of dummy "delegate" objects that perform a variety of movements ranging from random walks to target homing. The delegate objects can be replaced with animating characters or objects. If the characters are Character Studio bipeds, they can intelligently choose appropriate animations from a list provided by the user and smoothly transition between the animations as they follow their programmatic path.

The amount of control that Crowd offers is quite impressive and a little intimidating. The basic movements of the delegates can be modified with a variety of avoidance and collision behaviors, and can be constrained to operate within volumes or on object surfaces. Complex behaviors can also be scripted directly using MaxScript. The relative priority and effect of behaviors can be keyframed so that a crowd of delegates could converge on a target until a given frame and then turn and wander away from it.

Related behaviors can also be assigned based on MaxScript conditionals.

The interface attempts to manage all this complexity with mixed success (see Figure 2). To help manage large scenes, delegates can be grouped into "teams" which can be controlled together. Cloning and random placement tools also help generate large groups quickly. However, the cramped confines of the Max interface mean a lot of functionality is jammed into one 218-pixel-wide rollout. Fortunately, Character Studio's manual is comprehensive and contains a series of tutorials to help users grasp the underlying logic of the process. Very few users will be able to get much out of the Crowd system without at least a glance through the manual.

When all of the various behaviors and conditions have been assigned, the movement of the entire system is computed, or "solved," much like a Max dynamics system. Like dynamics, crowd simulations can be computationally expensive and may take quite a while to calculate. When complete, however, the scenes run at the same speed as a normal Max scene of comparable complexity. Scenes with random elements can be reinitialized and rerun with new seed values to generate multiple animations that satisfy the same set of conditions.

### Conclusions


Overall, version 3 is a significant step forward for Character Studio. How exciting it seems to you depends largely on which features will be involved in your work. The upgrades in Biped and Physique are useful improvements for a worthy product, though they are less than revolutionary. For cinematic animators, Crowd may be an enormous time-saver and open a lot of new creative doors. Animators who don't have much call for crowd simulations (one suspects most real-time game animators fall into this category) may still find Crowd useful for creating particle-system-like effects animations, or even for easily generating idle animations.

Critics of Character Studio will note that some long-standing gripes about the



package — the inability to animate scaling of limbs, the limitation of bipeds to no more than four IK-enabled limbs, and Physique's difficulties in dealing with topology changes in a mesh, to name a few — have not been addressed. Nevertheless, particularly with the new pivot point mechanism and the interesting possibilities of spline deformers, the package is an increasingly powerful tool for character animation. Equally important is the fact that Character Studio remains the only game in town for character animators on the Max platform. This dominance will be somewhat shaken with the release of Max 4 (if we can believe the pre-release publicity surrounding the new IK tools, at any rate), but for the foreseeable future Character Studio is and should be the Max-based character animator's tool of choice.

Is it time to upgrade? Character Studio 3 is unfortunately true to Max's lamentable tradition of file-format incompatibility.

Max files made with earlier versions of Character Studio will have to be resaved, and in most cases, Physique modifiers in those files will have to be manually reinitialized from their initial skeletal poses before they can be used with version 3. Although the new MaxScript extensions make it possible that this annoying task could be automated, it seems a needless irritant and will undoubtedly lead to version control problems for teams with a large content base. Also in the worst tradition of the Max platform, Character Studio 3 has no mechanism for saving files in older formats, so any team that wants to switch will have to switch en masse — there is no way for version 3 users to make files available for users of earlier versions. Unless users have an immediate need for one of the new features, they may consider waiting until they upgrade to Max 4 to upgrade their Character Studio seats in order to take all of the file-format hits at the same time. 

### CHARACTER STUDIO 3

#### STATS

##### DISCREET

Montreal, Quebec, Canada  
(800) 869-3504 or (514) 393-1616  
[www.discreet.com](http://www.discreet.com)

##### PRICE

\$495 upgrade, \$1495 new

##### SOFTWARE REQUIREMENTS

Windows 98/ME/NT/2000 with 3D Studio Max 3.1

##### HARDWARE REQUIREMENTS

128MB RAM with 350MB disk space, minimum 1024×768×32 display

#### PROS

1. Flexible crowd animation system.
2. Improved IK; animatable pivot points.
3. Faster skeletal deformation with new control options.

#### CONS

1. Cumbersome file-format update procedures.
2. New functions difficult to access.
3. No backward compatibility with earlier versions.



*“And when they battle in a puddle, it’s a tweedle beetle puddle battle.” — Dr. Seuss*



FIGURE 1. The Puddle Battle.

# The Battle Rages On

**W**hen I left you last time, I had a battle beetle built. To this, I added a skeletal system and defined bounding spheres at the joints. These spheres will be used for collision detection. During the last month, the beetles have scrounged up a couple of paddles to make the battle more brutal. However, before any kind of battle can actually begin, I need to find a way to make the beetles budge. The goal is to create a battle scene like you see in Figure 1.

With a traditional skeleton-based animation system, I would use animation data to set the orientation of each joint in the character directly. However, there are two problems with that approach. I want the characters to behave like soft objects. When they are hit, they should squash and stretch. For this to happen, the “bones” in the characters cannot be as truly rigid as they are in a normal skeletal animation system. I need to devise a system that will allow the animation to guide the bone position, but not be limited by that.

The second problem is that I would like the characters’ bodies to react to hits somewhat realistically. When a character is struck in the arm, the arm should react. In many games this is handled by creating a series of reaction animations that handle situations such as a hit to the head or a body blow. I am sure everyone remembers the great piece of reaction artwork in the “vintage” arcade game PUNCH OUT. When an opponent was hit in the belly, the character let out an audible “ooof” and his eyes bugged out — great stuff. Good as it was, however, the character was not truly reacting, it just responded to a hit by showing you what a canned response looked like. I would like my character

**JEFF LANDER** | Presently, Jeff is probably pondering perplexing problems with polygons. Jot Jeff a jaunty jingle at [jeffl@darwin3d.com](mailto:jeffl@darwin3d.com).

response to be physically based. The bodies should react directly to the forces of the hit.

### In This Corner, Rubber Man

In order to accomplish these goals, I need the skeletal system to be driven by a dynamic simulation. I could treat each bone in the skeleton as a chain of rigid links and then use rigid body dynamics to simulate the characters. However, this would not allow me to achieve the goal of making the characters soft. The mass-and-spring models that I have used in the past to create soft objects are closer to the feel I am trying to achieve.

I have the skeleton of the character, which is effectively a series of root points connected in a hierarchy. I can make the root of each character bone a 3D particle, and then use a spring to connect each of these particles to each of the particles representing the children of that bone. In Figure 2, you can see the character and the underlying particle-and-spring skeleton.

The simulation will have a couple of differences from simulations I have created in the past. I am not going to apply gravity to the individual particle nodes. Since the character should be able to support its own arms, this should not be a problem. I also want to keep the characters largely grounded and upright. If the bottom of the character were weighted normally, it might be easy to knock it down. To solve this, I can make the root particle of the character, which is the top node of the hierarchy positioned on the ground between the feet, more massive and thus more difficult to move. This makes my character like those big inflatable punching clowns that you can knock over, but they pop right back up.

Now I have a character composed of a rubber skeleton tied together with springs. The next step is to figure out how to move it.

### Animated Rubber

As I mentioned earlier, in order to animate a character with a skeletal system I would need to take the animation data for each joint and set the position and orientation directly. However, to drive the animation physically, I need to apply forces to animate the character. If it were a hierarchical rigid body, I would use a controller, such as a proportional integral derivative controller, to change the orientation of each link. Chris Hecker described this method in depth in his article on physical controllers ("Physical Controllers: Re-inventing Game Animation," April 1999). However, since I do not have rigid bodies, these controllers will not work.

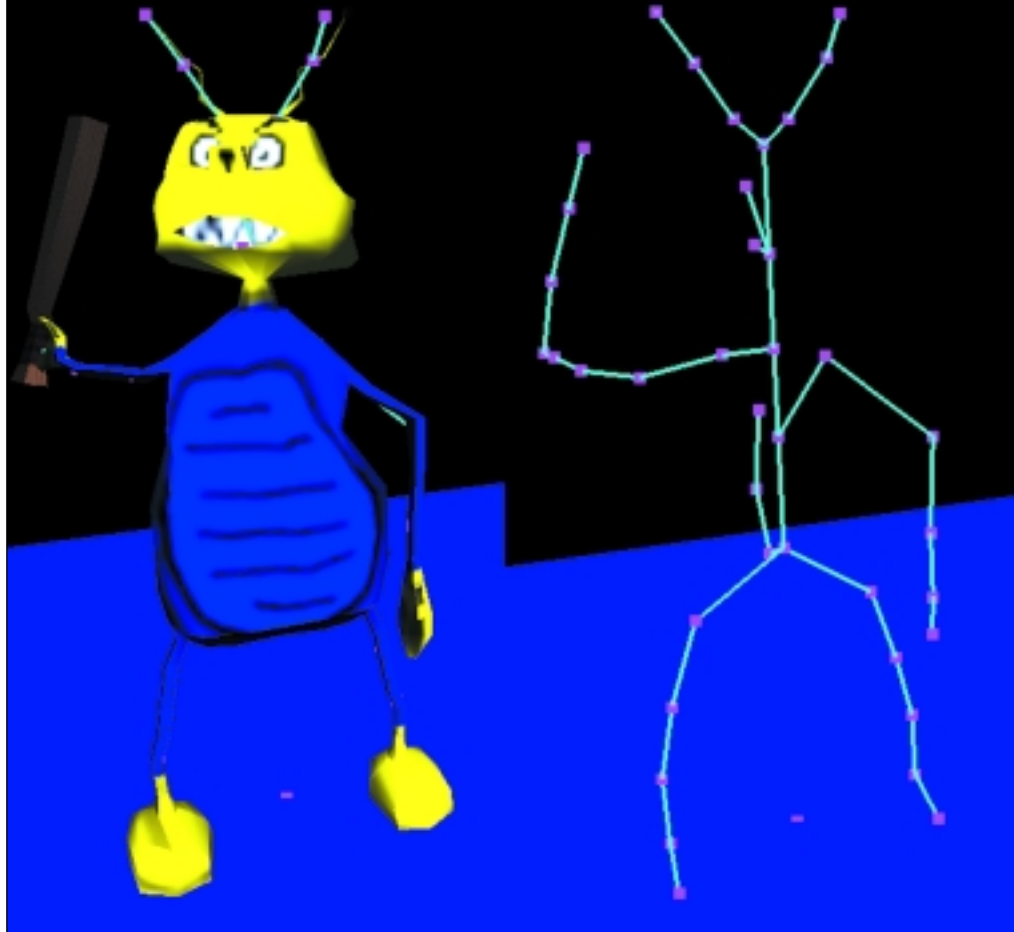


FIGURE 2. The battling beetle and its skeleton.

Fortunately, the solution is very easy. The animation system generates a set of values for the joints given the desired animation targets. This information can be converted easily to world space locations for the root of each bone. The way I accomplish this is to build the transformation matrices for each bone in the skeleton, and then grab the world space coordinates directly out of those matrices. This efficiently provides me with target positions for my physical particles.

To move the simulation particles toward those animation targets, I once again turn to the ever-useful spring. The spring is attached to each simulation particle and to the animation target position. This creates a force on the particle which, when integrated forward in time, will move the particle to the target. Using this system, I can make the dynamic control mesh assume any pose.

This system achieves some of my main goals. I get an animated character that moves in a flexible manner and can respond to dynamic forces. As an extra benefit, the animating particles gain and lose momentum as forces are applied. This causes hits to actually have some "weight" behind them. For example, if you have the character swing its arm around in a windup followed by a punch, the impact will have more force behind it than a punch with no windup. This is a level of realism that is difficult to achieve with canned animated sequences.

Once the simulation particles have been moved to the desired positions, I need to make the changes back in the 3D object so we can see the results. That is basically as easy as you would expect. I set each of the actual mesh bones to the positions of the simula-

tion particles and render. The one thing that the simulation does not address is the orientation of the bones. Since I am dealing strictly with point masses, I do not have any orientation data, as I would in a rigid body simulation. For now we can just use the orientations that come out of the animation system. This may lead to visual problems where the bone is bending but not rotating, but let's leave that alone for now.

## Taking a Pounding

The simulation now has particles flying all over the place. Hopefully, some of those particles will be hitting my opponent. In order for anything to actually happen when this occurs, I need to add some collision detection and response.

Earlier we attached bounding spheres to the base of each bone in the skeleton. These are the boundary spheres that I will use for the particles. So, in order to handle collisions between the characters, I need to add handling for sphere-to-sphere collision. Fortunately, I covered this back in my column on billiard physics ("Physics on the Back of a Cocktail Napkin," September 1999). To simulate the collision between two spheres, I first need to determine whether the two spheres are colliding by checking their positions and boundaries. I also want to check the relative velocity of the two spheres. If they are moving toward each other, they are colliding. However, if they are moving away from each other, I shouldn't do anything. You can see the situation in Figure 3 where sphere A is traveling with a velocity vector  $V(A)$  when it strikes sphere B.

Once I have determined that they are colliding, Newton's third law of motion takes effect: The forces exerted by two particles on each other are equal in magnitude and opposite in direction.

I calculate the direction of collision between the two spheres and use that to determine the normal velocity vector,  $n$ , and tangent velocity vector,  $t$ . The normal velocity is negated and scaled by the coefficient of restitution,  $K_r$ , and applied as an impulse to the particle. The opposite impulse, scaled by  $1 - K_r$ , is applied to the other particle.

Another issue I ran into was that since the character mesh had quite a few boundary spheres in it, several of them always overlap. This really messes up the simulator, since it continually believes a collision is happening. My solution was to create an exclusion table that ignored collisions between spheres that were initially determined to be overlapping.

The characters are now able to knock each other's bones around using big sticks. However, I now need to create a control system for the characters.

## Maintaining Control

I need a control scheme for allowing the beetles to battle. I could attach animation poses to keyboard keys and allow the system to handle the in-betweens, but that wouldn't look very nice. It also would not give the user the amount of control I would like.

The solution is to create "moves" that the user can select from. Each move will have a starting and ending pose. After the user selects the move, pressing the mouse button initiates the move.

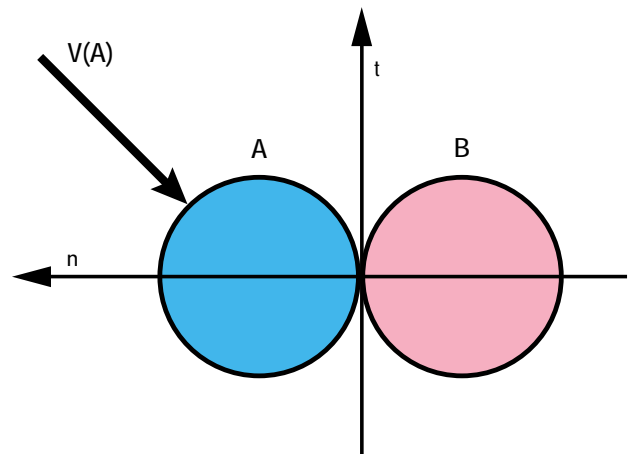


FIGURE 3. Two spheres colliding.

By moving the mouse, users can interpolate between the two poses as fast or slow as they desire.

The interpolation is handled using a quaternion spherical linear interpolation (SLERP) on each bone. This allows smooth steps between each pose. With my smooth control system, users can hit as hard or soft as they want.

## No Pain, No Gain

I now have a system where the two battling beetles can strike each other. In order to determine what kind of pain they are inflicting, I need to create a damage model of some sort. In the battle with paddles, the greatest amount of pain comes when the paddle connects with a more fleshy part of the opposing beetle.

Fortunately, this is exactly the sort of information that is already generated by my collision response system. From this system, I know when the boundary spheres in the opposing characters collide. I can then check to see, for example, if the paddle of one beetle connects with a head sphere of the other. That would be a hit likely to cause damage in most players' minds. But how much damage has occurred?

As I discussed above, the collision response system calculates that a collision has occurred and also calculates physical values that are needed to resolve it. The system calculates the collision normal as well as the velocity at which the two spheres collided. I can take that collision velocity and multiply it by the mass of the paddle, giving me the momentum of the collision between the paddle and the head. This value can then be scaled by some scoring system that ensures blows to the head are more damaging than blows to the arms. The result is then deducted from the damaged beetle's health. When a beetle is too injured to continue, a winner is declared.

This damage model provides a fairly standard battle system for a fighting game. However, by basing the system on an underlying physical simulation, I can crank the realism up a bit more. Beetle action poses are achieved through the use of controlling springs. The strength of those individual control springs regulates how quickly and accurately an individual move is achieved. It is probably obvious to many that these spring strengths could be dynamically changed. If the spring strength is reduced proportionally to

the damage that an individual control bone has taken, that control bone will have difficulty achieving a particular pose. This reaction simulates the way that a battered boxer has a tougher time swinging at his opponent as the fight goes on. A traditional animation system based on the playback of motion capture data does not have an underlying physical model controlling it. For that reason, dynamic modification of the animation system is difficult to simulate using these traditional methods. That is one of the reasons that you don't see that level of realism in many fighting games. In most fighting games, the system may track a stamina variable which determines if and when you can make a move. However, once the move is made, it always comes out the exact same perfect way. With a physically based animation system, dynamic changes to the fighter's abilities are much easier.



FIGURE 4. Beetle in a puddle.

## Environmental Issues

I wanted to create a puddle for the beetles to battle in. Once again drawing from the archives, I turn to a column I wrote in December 1999 (“A Clean Start: Washing Away the Millennium”) describing a well-known algorithm for generating rippled water. The system uses an array of height fields that represent the water level. To create a ripple, a drop is generated at a position in the array by changing the height. The system then applies a filtering process to advance the system and make the ripples animate.

I can use this system to generate the texture for my puddle. Since I know the point where the beetle's feet are standing, I can generate drops at those positions. The water height-field array is then updated, and every frame this texture generated by the array is uploaded to the 3D graphics card for use as the floor texture. You can see the results in Figure 4.

Since the texture upload does take time, it may not be necessary to update it every frame. However, in this application it doesn't seem to affect things too much. To make the environment even more realistic, the water texture can be used to displace the ground grid. This kind of software displacement mapping is fairly computationally expensive, but for a small grid it may not be too bad.

## Bring on the Battle

So at the end of the day, what have I created? By pulling together a physically based simulation with a skeletal animation system we have the makings of a dynamic fighting game. The decision to use point masses connected by springs is the first item up for review. It does result in the squash and stretch that you would expect when two cartoon beetles do battle. However, they

may be a bit too squishy. I certainly want the characters' bodies to squash a bit, but it is unclear if parts such as the paddle should appear more rigid and less like a rubber mallet. I can adjust this quite a bit by playing with the control springs and making some dependent links rigid. However, for less cartoonlike characters, this method would probably not work as well.

For more realistic characters composed of bones that may not stretch, a more rigid approach would be more appropriate. I will take a closer look at linked rigid characters next month.

These characters are currently unable to fight without user control. For a two-player or Internet game, this may be sufficient. However, most players like to have a computer opponent at least to practice against. A simple finite state machine can be used to create a decent opponent. In a finite state machine, I create a number of states for the computer-controlled character, and set rules that govern when the states change.

For my simple fighting character, two states will probably do the trick. When the character is healthy and ready to fight, the character goes into attack state. In the attack state, the character stands in a ready pose and then initiates various attacks. Whether the attack is an overhead, sideways slash, or body blow is determined randomly. Once the attack is finished, the character returns to the ready pose. If, while in ready pose, the character detects that his opponent will attack, a block move is initiated. If the opponent is not within reach, the character moves closer.

If the health of the character goes below a specified level, the defense mode is set. In defense mode, the character does not initiate any attacks and simply stands in a defensive posture and attempts to block any strikes. If the health is particularly low, the character can be made to take a step away from the battle.

This simple system is not nearly enough to make a world-class artificial fighter. However, it is good enough to provide an amusing battle. The system could be greatly improved by creating more states and perhaps including a learning system that keeps track of the opponent's strategy and refines moves that were successful in the past. But I will leave those improvements up to you. 🐞

### FOR MORE INFORMATION

Hecker, Chris. “Physical Controllers: Re-Inventing Game Animation” (*Game Developer*, April 1999).

Discuss this article in Gamasutra's Connection!  
[www.gamasutra.com/discuss/gdmag](http://www.gamasutra.com/discuss/gdmag)





# Interface Design for Games

**T**oday's game development artist is faced with a vast array of tasks. With more and more sophistication going into the development of games, users' expectations and level of acceptance are pushed ever higher. This is especially true in the area of graphical user interface designs. The amount of effort that can be devoted to the design and implementation of a game's interface alone can be staggering. Over the years, the software industry as a whole has evolved an awareness of the science behind interface design and how it can make or break a product. How people interact with computers, extract information, and utilize this knowledge is becoming a critical element in the development of software.

In the game development industry, developers are devoting more time and more sophistication to this critical link between the programming code and the player. It usually falls upon the game artist to create this interface between man and machine.

If you are like most artists on a project, you wear different hats on different days. One day you might be texturing, while on another, you are called upon as an animator, modeler, or designer. As team sizes grow, specialization is becoming more common, but it still seems as though the

game artist is a bit of a generalist. This is especially true if the project is smaller in scale than the triple-A, multi-million-dollar megatitle that includes everyone and their cousin in the development cycle. Or perhaps you are now an art lead or art director. Whatever you call yourself, the chances are good that you will eventually be called upon to design and create a GUI for a game.

This can be a daunting task for the uninitiated. Perhaps you've never considered yourself much of a graphic designer, or maybe you haven't really given it much thought at all. After all, you did design that web site for your Aunt Gertrude and it wasn't that hard — what can be all that hard about designing a GUI for a game? You slap some shapes, text, and colors down, add a drop shadow or two, and you have yourself an interface — right?

## The GUI Isn't the Game

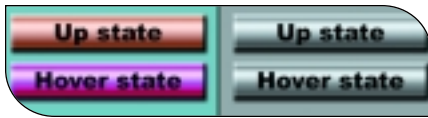
**A** whole multitude of elements come into play when designing an interface. What makes one GUI good and another great? What are some of the rules that can be broken without getting into a lot of hot water?

The first step is to understand what graphic design is and how it relates to user interface design. Essentially, graphic design is nothing more than presenting information in a strong, consistent, visually appealing format. The text and visuals are organized in such a way as to provide the viewer with an easy way to retrieve, sort, and store the information. Composition, layout, and typography are all balanced to provide the strongest visual presentation possible. However, graphic design in the traditional sense is a one-dimensional medium for conveying information. It is targeted to a noninteractive, one-way presentation.

User interface design adds many new elements to the equation. Not only are sound graphic design principles needed, but consideration has to be given to a whole new set of design requirements such as user interaction, navigation and the impacts of sound, animation, and time. Designers must consider how they will be controlling the user throughout the experience: what sort of feedback mechanisms will be in place, and how all of these elements will tie together to form a cohesive unit.

Oftentimes, the best GUI is the one that is most transparent to the user. The last thing you want is to have the gameplay elements impeded by a poorly designed interface. Depending upon the needs of the game, a minimalist approach to the user interface design might prove to be the most appealing to the player. With all of this to consider, it's easy to see why creating a well-thought-out design can have a major impact on your time.

**MARK PEASLEY** | Mark is currently the art director at Gas Powered Games. Visit his web site at [www.pixelman.com](http://www.pixelman.com) or e-mail him at [mp@pixelman.com](mailto:mp@pixelman.com).



**FIGURE 1 (above).** Using color without altering contrast might give difficulty to users with color blindness. **FIGURE 2 (right).** A flowchart will solidify design and structural ideas as well as work out errors in the interface's general functionality.

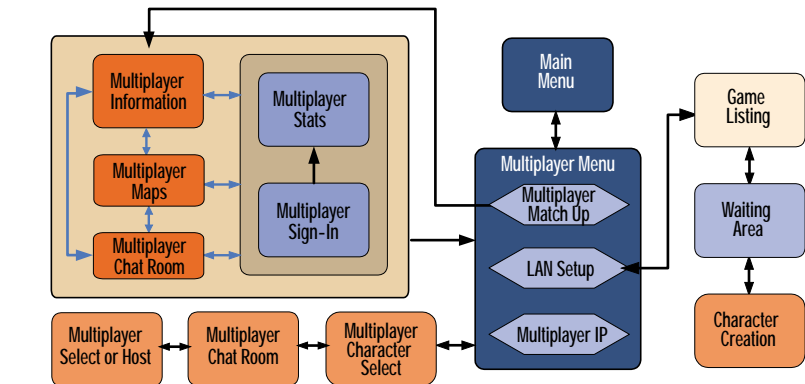
## Things to Think About

Let's first consider some basic graphic design elements, and how they might be expanded upon for use within a graphic user interface design.

**Simplicity.** The best thing to keep in mind is K.I.S.S. (Keep It Simple, Stupid). Every artist I know (including myself) has a tendency to noodle something beyond what is needed. In interface design, the simplest solutions are usually the easiest to use, and the most effective. Less is more — sometimes more information and greater impact can be gained by using fewer elements.

**Consistency.** We are creatures of habit. We learn through repetitive occurrences and are quicker to respond to events if we can predict their behavior. Once users have learned the function or placement of an interface element, they will use that knowledge on new screens in an attempt to find consistency in the structure. If the consistency isn't there, they will be frustrated by having to relearn new paradigms. Consistency also makes a design seem simpler to use. If users feel that they inherently "know" how a menu will function, then they won't view the interface as an unpleasant learning experience. By setting up consistent placement of repetitive elements, such as where to find the cancel button or how to minimize windows, you can create an environment that the user feels empowered to explore. Metaphors can also add consistency. For example, your interface might always reveal help files when the user rolls over an eyeball symbol. The user learns to associate the help function with the eyeball symbol, regardless of its location.

**Know your target user.** In the broadest strokes, this means understanding and predicting how the product will be used by the target demographic group. The



GUI you design for a kids' game will be radically different from a first-person shooter. Beyond the obvious differences, you need to consider how knowledgeable the user is, how they perceive the information presented, what sorts of feedback mechanisms will be used, and how simple the navigation needs to be. In addition, consider the cultural implications that might affect your design, particularly if the product is going to be international. In the United States, the color red might mean "stop" or "danger." However, that same color can have an entirely different meaning in another culture.

**Color usage.** Don't rely on color alone to convey critical information. Use additional feedback mechanisms. Consider that a certain percentage of the population (roughly eight percent of men) has either color blindness or color perception deficiencies (see Figure 1). Use enough contrast between the foreground and background elements, especially where text is concerned. Avoid large amounts of light text on a dark background; it is more difficult to read. As with graphical elements, use color in a consistent manner.

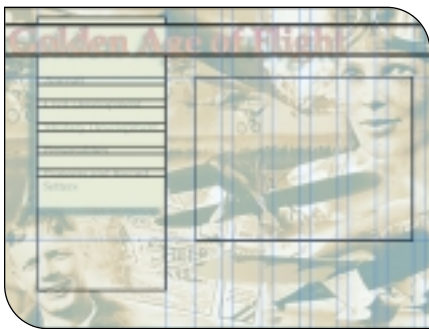
**Feedback mechanisms.** What is the visual mechanism you will provide the user to enhance the experience? A common example of this is standard buttons in most applications. They usually have a rollover state that indicates when you are over a "hot spot" of the button. This can take the form of a highlight, a special effect, an animation, a sound, or any combination thereof. Give the user visual feedback that they have accomplished a task such as pushing the button. Also, let the user know when the computer is working on something. A progress bar

can satisfy this need easily and prevent user frustration. If loading a file takes longer than five to ten seconds, it's a good idea to provide the user with some form of a progress bar or percentage feedback. Without these, load delays can often be mistaken for program crashes.

## Design Elements

**Create a flowchart of the design.** I know this seems obvious, but believe it or not, it's something a lot of designers don't get around to doing. This is especially critical when the functional design and the aesthetic design are being done by two different people. It is fairly common for the game designer to come up with the functional requirements of the menu system, while the artist comes up with how it will look. Oftentimes a flowchart will flush out flaws in the logic of the menuing system well before any time and effort have gone into creating art assets (see Figure 2). Always try to get all elements of the GUI defined as soon as possible. Without a doubt, the most difficult GUI design to create is one that needs to be open-ended enough to allow for undefined design elements. This approach often leads to having to redesign the entire GUI from the ground up.

**Make navigation simple.** Whenever possible, make the navigation as simple as it can be. Think of the user's memory as a RAM chip, with only a finite amount of space that can be used before it starts to be overwritten. Our short-term memory works essentially the same way. Ideally, users should never be more than three to five clicks away from accessing the information



FIGURES 3A and 3B. A sample interface (3A, top) and its underlying grid design (3B, bottom).

they want. Of course, this is sometimes impossible to maintain, but keeping navigation to a minimum will increase the user's comfort with menus. Grouping multiple functions or options in one area is a good practice. This allows users to make more efficient decisions and keeps them within the same screen real estate. Avoid giving users the impression that they are jumping from screen to screen unless it is by design. This has a tendency to give users the feeling that they are navigating a large menu system where they might easily get lost.

**Establish a grid.** Underlying almost all good design is a grid. The grid is a visual structure that provides the framework for the design and gives it balance. Take any magazine, newspaper, or advertisement as an example. If you study the piece, you will soon see a grid that all of the images and text fall within. At an almost unconscious level, this grid provides the consistency I talked about previously. From a design standpoint, the grid gives the artist a logical structure for the layout. A well-designed grid will give the GUI screens a look and feel that is consistent and tight. It also provides a good basis for narrowing down design decisions and establishing a set of rules or style guides that can

be applied to new screens (see Figures 3a and 3b).

**Construct a tiered menu system.** The most powerful menu system is one that can expand with the user. For the novice, it contains only the most basic of commands. Break their decision making down into simple, controlled segments. For the advanced user, the interface can be made to reveal a more complex level, allowing for greater control. As an example, in the GUI for a flight simulator I worked on several years back, we had a main menu with only five choices. One of those choices was a "fly now" mission, where the designer had predetermined all of the elements that the novice user would most likely choose. They were then presented with a preflight screen where the mission was described, the settings shown, and the "fly" button was available. If users preferred, they had the ability to alter any of the settings, but they could also simply press the "fly" button and be in the air within three menu clicks of the main menu. For advanced users, we offered menus within menus that gave them the ability to customize almost any element of the game. This proved to be very well received by the users, and the concept went on to become a standard for many of the flight simulators that the company produced in later years.

**Remember localization considerations.** If your product is small and has a limited target audience, you might not need to think about the localization impacts. However, overseas sales make up a substantial part of the target market for many of the games under development. By keeping in mind some of the more simple localization rules, you can avoid a lot of rework down the road:

- Don't embed any text into your art if at all possible. Text should be handled via code as either TrueType fonts or bitmapped fonts.

- If you do have text embedded in art (like in a road sign or logo), then it's a good idea to get into the habit of separating the text onto a unique layer in your working base art file. That way, the localization of it can be done easily.

- Allow 30 percent extra space in all areas where type is present. German language conversions are notorious for needing extra space.

- Be aware of cultural implications of symbols, colors, and sounds.

- Always avoid going below 12 pixels in font height. That is about the minimum number of pixels required to form the symbols in a Japanese font. If, for example, you have created a special button that requires your special eight-pixel font on it, the localization using a 12-pixel font will run into some serious space constraint problems.

## Typography Fundamentals

**W**hat font should be used? Should more than one be used? If so, should it just be a bold version of the same typeface, or should you go with a different one altogether? Is it O.K. to mix serifed and sans serif fonts? How does the font look at the game's resolution? These are some of the questions you will probably ask yourself when it comes to choosing the right font or fonts for your interface. A good choice will help solidify your design, while a bad choice will look out of place and detract. Here are some fundamentals to think about as you begin to narrow in on your selections:

**Shape recognition.** We recognize letters and words as shapes, which we have memorized as a meaning or concept. Think about how you read a page of type. You aren't sounding out each letter in the sentence you are reading. Instead, you have memorized the combination of letters into words. When text is written in all uppercase letters, it is much more difficult to read, since the pattern recognition is all just rectangles instead of more distinguishable groupings found in a combination of upper and lowercase letters (see Figure 4). A good test of this is to take any paragraph of text in a Word document, switch it to all uppercase, and read it. You will generally find that it is harder to read, and the speed at which you progress across the page is slower.

**Serifed vs. sans serif fonts.** In large bodies of text, serifed fonts are easier to read since the serifs provide horizontal structure for the eye to follow. However, consideration must be given to the resolution limits of a game screen. Even though hardware continues to improve, we often still design for the lowest common denominator in terms of screen resolution. On a standard 640x480

# Shape recognition

## SHAPE RECOGNITION



**FIGURE 4 (above).** The differences in shape recognition between upper- and lowercase letters.

**FIGURE 5 (left).** Avoid using text less than 12 pixels in height: smaller sizes pose problems with both readability and localization.

screen, small fonts become very cryptic. Once you get down into the ten-pixel range for letter height, most serifs and other subtleties are lost in an effort to make the letter's form even readable. You should generally avoid mixing serifed and sans serif fonts unless it is done carefully.

**International considerations.** If you are creating a bitmapped font for your game, you will most likely be opening a large can of worms you didn't know existed. This is especially true when the font is to be used for localization in different languages. All of the specialized ASCII characters will need to be present for localization.

**Kerning and the use of bitmapped fonts.** The type we are used to seeing in everyday print uses kerning, which is the adjustment of space between characters so that part of one letter extends over the body of the next. An example would be two circular letters such as a *c* and an *o*. They would have a much narrower kerning than two parallel lowercase *ls*. Kerning takes a beating when bitmapped fonts are generated. The code is usually set so that each letter is defined as a cell in the bitmapped font. The cells can be uniformly spaced, but kerning is a much more difficult proposition. In most cases it ends up being a low priority on the programming list and usually ends up being dropped. If the game engine supports TrueType fonts, then the kerning is maintained, but control over subtle alterations is reduced.

### Rapid Prototyping

There are several programs out there that allow GUI designers to create a mock-up of their idea quickly. These pro-

grams are easy to use, don't require a ton of ramp-up time, and, more specifically, don't require a programmer to become involved. Programs such as Macromedia's Flash or Director can easily create a mock-up of the navigation elements, with sound and functionality. These prototypes serve to solidify artistic elements as well as provide programmers and other team members with a very clear, concise vision of what you think the GUI should look like. If time and budget allow, this is a good way to work the kinks out of your design ideas without involving a lot of people. It is often easy for artists to visualize what the end product will look like, but they have difficulty describing it to others accurately. These prototypes provide artists with a means of communicating their design ideas clearly and with little room for misinterpretation.


### Creating Game-Ready Art

It is always a good idea to bear in mind the repetitive nature of the elements that make up the GUI for a game. Frequently, these elements can be made from a common set of base artwork without a visible loss in quality. It is often possible to create a basic set of building-block components (buttons, surround elements, text boxes, and so on) that 80 percent of the GUI elements can be generated from.

When creating art, especially hardware-only GUI elements, optimization of redundant elements is essential. You may find that in-game GUIs require special attention, since texture memory is usually at a premium. You must also consider multiple screen resolutions. Most games have the capability




to change resolution based upon the user's desires. Will you provide a single set of art that is scaled up programmatically to the larger sizes? How does this affect the look of the art? Or, will you provide two or three sets of artwork that will be used at the various screen resolutions? Does this decision essentially double or triple your art production time on the GUI?

### At the End of the Day





Now that you've learned a lot of rules of what to do and what not to do, you are free to break them. These rules are not hard and fast, but rather are guidelines that are to be followed when applicable, and broken when the design calls for it. But do so cautiously. These rules are a form of structure that give you a framework within which to base your designs. Within that structure, there is an almost infinite amount of freedom for the GUI designer. 

#### FOR MORE INFORMATION

##### WEB SITES

-  Interface Hall of Shame  
[www.iarchitect.com/mshame.htm](http://www.iarchitect.com/mshame.htm)
-  IBM — Ease of Use  
[www-3.ibm.com/ibm/easy/eou\\_ext.nsf/publish/561](http://www-3.ibm.com/ibm/easy/eou_ext.nsf/publish/561)
-  Yale Style Manual  
<http://info.med.yale.edu/caim/manual/contents.html>

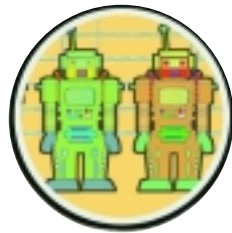
##### BOOKS

-  Beaumont, Michael. *Type: Design, Color, Character and Use*. London: Quarto Publishing, 1987.
-  Hamlin, J. Scott. *Interface Design with Photoshop*. Indianapolis: New Riders Publishing, 1996.
-  Marcus, Aaron. *Graphic Design for Electronic Documents and User Interfaces*. New York: ACM Press, 1992.
-  Swann, Alan. *How to Understand and Use Grids*. London: Quarto Publishing, 1989.



# The Sound of Money (Down the Potty)

## Common Audio Mistakes in Kids' Games



Do you make games

for young kids?

Would you like to know

how you can stop

flushing a whole load

of cash right down

the crapper?

Please, read on.

There is a great and tragic battle that has raged for decades and has taken a drastic toll on our industry. We have been fighting for dollars, but we have been losing business and alienating customers. And, oddly enough, the key soldiers in this battle are the musicians and the “sound guys.” While they themselves may have respect for the unique nature of the terrain upon which they shed their blood, often the commanders of their forces do not.

The most important point that gets missed is this: the person who buys the game (the parent) *only* experiences the game through the audio. This is an important point. History repeats itself, but since I am not yet history, I will paraphrase myself instead: Assuming that the game installs easily and that the kid can play the game mostly by him- or herself, and that the kid pretty much likes the game, all of the customer satisfaction, everything the buyer experiences, all of the motivation to buy the next product — comes from the audio. The parents do not see or play the game. They *hear* it.

Yet due to the inability of Command to recognize this fact, never so much as even three percent of resources has ever been directed to the soldiers at the very important musical front. Historians are still trying to figure that one out.

---

**THE FAT MAN** | *is a big-hearted alien who wears the skins of cowboy heroes whose bodies he has found in the desert. He finds it to his liking to hover over Austin in a huge radar-cloaked zeppelin, composing music for games with his legendary team of Cowboy Composers, Team Fat. His work on hit games has more than once changed the face of game music. People magazine called him “a top candidate for the most prolific — and obscure — living American composer,” yet as he consumes only ammonia and uncooked brown rice, his rates remain reasonable, and he is very well-behaved. He can be reached at fatman@fatman.com.*





## Atomic Weapon: Use with Discretion

**A**udio, especially game audio, is a powerful weapon. When used properly, it has the power to involve, immerse, elevate, and reward. It has the power to excite. It can make an artificial world appear to be deeper, older, and much more complex and complete than it actually is. But when misused, audio reveals its most awesome and deadly power — the power to annoy.

The annoyance situation for any game is already potentially dangerous. The game developers budget for an hour of music. That hour is stretched over a 40-hour entertainment experience. This can be likened to driving cross-country with one audio cassette that you didn't choose. Furthermore, the scarcity of disk space requires that the music be played at a low sample rate, or via MIDI, or, God forbid, through some crazy auto-composing routine like DirectMusic Composer. So what you're getting isn't exactly a direct view into the heart of Aaron Copland. Add to that tiny speakers and an audio environment that was never put through QA with anybody who knew what to listen for. Of course, I will be more than happy to send a formal letter of apology to anybody who can show me — in writing — that a feedback cycle exists in their development timeline in which the musician, the only one who knows how many times that D-minor section is supposed to repeat, is supposed to listen to the finished game and correct mistakes before it ships.

Now add to this dire situation the multipliers that are unique to kids' games. For some reason, somebody has decided that any game created for somebody under the age of nine will have the following audio characteristics:

- The compositions will be more repetitive than those in adults' games.
- The tones will be pedestrian.
- The tunes will be shorter and simpler than even normal game music.
- The tunes will all be in the same key, C-major.
- Half the tunes will be public domain "favorites" such as "Twinkle, Twinkle, Little Star."
- Characters will yell in high, squeaky voices the following phrases: "GOOD JOB!" "VERY GOOD!" "TRY AGAIN!" "NOT QUITE!" "HEY! YOU'RE GOOD AT THIS!" "GREAT JOB!" "HEY! YOU'RE GOOD AT THIS!" "GREAT JOB!"

Why? Because it's easy. Because people think kids don't notice

these things. Because people think kids actually like these things. But that's insane. None of them is necessary or desirable, ever. Kids like good music, just like you and me. They get bored, just like you and me. And even if they didn't, it doesn't matter because you're never going to drive the kid crazy with good audio. But you're sure to drive the parent crazy with that crap you're giving them, and that's the last sale you'll make in that household.

## And Again I Say, History Repeats

**T**hat is the battle. Repetition is the enemy, so you've got to fight it with everything you've got. The following are some tips:

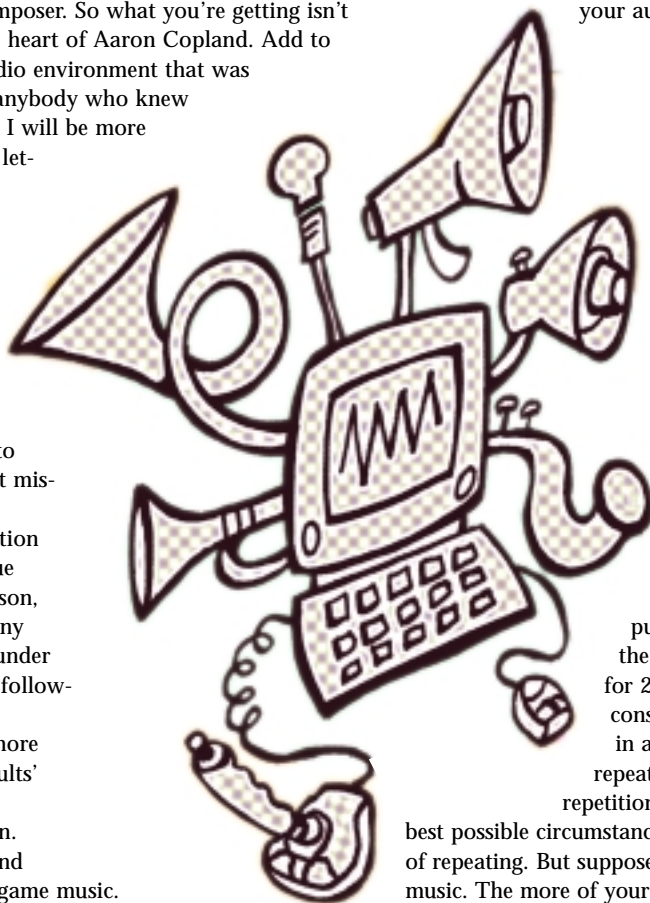
**Don't rely on new technology or clever gimmicks to make things sound better.** That is like trying to build a baby-sitting robot instead of being with your kids. Always direct all your audio energy toward making lots and lots of warm, exciting, varying, heartfelt audio. You can do this better with a kazoo and a cassette recorder than with physically modeled 3D interactive vaporware.

**Don't use one repeating tune for an entire level of a game.** That's old school, there's no excuse, and it will kill the parents. Don't do it.

O.K.? Just don't. If any one tune in your game repeats for more than five minutes, you should do one of the following: (1) change to another tune after five minutes, or (2) stick a hot fork into your own eye, you evil moron.

**Reuse your resources in different circumstances.** I know you want special "cinematic" pieces, and "payoffs," and a unique piece for the puzzle with the cute duckies and such. But the math is simple. If the game's budget is for 20 minutes of music, and the game is constructed so that music plays for an hour in a given session, the music is going to repeat somewhat. And remember that three repetitions of the music would happen only in the best possible circumstances, meaning all music has the same odds of repeating. But suppose you get greedy about special-case music. The more of your music that goes to special one-time cases, the more the other tunes have to repeat to cover for it. Reuse that "Binky meets the cougar" tune as a "tense puzzle-building" or "will we win the pony race?" background piece. The kids won't mind — the situation will be different enough that they'll experience it as two different pieces of entertainment. The parents will be grateful for one less repetition of that incessant "riding the pony" music.

**Do not use musical structures that utilize repetition to build familiarity.** This is hard to get away from. Sure, conventional musical theory suggests that we play familiarity against variation to



achieve tension. That's why conventional music uses forms such as AABA. But in a game, you're going to get 30 repetitions of the tune at least. Think about that. How many times have you listened to the CDs in your house? Even your favorite CD? In a game, you can concentrate on the variation and relax on the repetition. An hour into the game, the familiarity will be there, I guarantee it.

**Don't insult kids with poor tones and yelling, squeaky voices.** Elmo and Barney are beloved, but so are the softer, lower-voiced characters such as Mr. Rogers, Captain Kangaroo, and Marvin the Martian. Kids' ears are brand new, and they can probably hear better than you. If you want to delight kids, play a pretty little bell for them. Yes, they respond well to high tones. Yes, they even like those little square waves, by God. But even though some little girls might be inclined towards pink, Crayola has not yet rationalized filling an entire box of crayons with that one color.

## Somebody Stop Me!

● .K., the knife is in. Now let's get down to the twisting. Picture this typical scenario: Mom works very hard at the

office, then barely has the energy to cook. Somehow she manages. "Dinner! NOW!" shouts Dad, feeling guilty that it wasn't he who cooked it.

"But I'm right in the middle of my game!" comes the kid's answer. Good. The game is interesting. The makers of the game can be proud. But the parents — the customers — are getting angry.

"DINNER! Get in here right now or I'll throw that damn thing through the window."

"O.K.! O.K.! O.K.!" answers the kid, if the parents are lucky.

The kid comes to dinner. What do we hear from the other room all through the meal? Music! It's the ice cream truck, parked in our living room, clanking out "Twinkle, Twinkle, Little Star" over and over and over and over and over again. And what's worse, every 45 seconds, a shrill voice yells out, "HEY! ARE YOU THERE? HEY! ARE YOU GONNA PLAY OR WHAT? SNORE!!!"

Oh, yeah, the parents are going to love that. Why isn't there a "fade to silence after two minutes of inactivity" feature? Were the designers never in a human family? Are they designing for kids





who don't eat, go to school, or play soccer? Is the target kid one who buys his own software and sets his own bedtime?

And do you know why these games sell as well as any other games for kids? It's because even the greatest games in the world have these design problems, and the parents' choice is either to buy no games for their kids or to buy annoying ones. Can you imagine what would happen to sales of kids' games if some of them stopped being deathly annoying?

## And Another Thing

I should end the article here, but it is my duty as a Texan to go into areas I know nothing about. Here is my non-audio gripe:

Who in the world decided to let this happen: "Mom, I can't come to dinner now! There's no place to save my game until I get out of this battle!" One game even makes you earn a certain object that allows you to save your game more often.

(Long pause, Texas voice, one eyebrow raised.) Now I'm no game designer, but I know financial-suicide-by-greed when I see it. The kid has simply got to be able to save instantly at any time. Whatever the justifications are for having designated places in the game from which the player can save, trash them. If you have to hit your lead designer with a cattle prod until he admits that he screwed up, do it. I'll buy you a new cattle prod. If it's a hardware

problem, and you'd have to solder another chip into every last cartridge yourself to rectify the problem, do it. I'll hold the soldering iron. Because that one element of game design has done more damage to our industry than any other.

Parents might say that the problem is the violence, but it's not. It's the fact that games have committed the unthinkable crime. They have made parents' lives even more difficult than they already are. And they have done this by making it impossible to get a kid who is playing a game into a car, into his clothes, to school, to the dinner table, or even out of a burning building if that kid is in the middle of a game with no save screen. And what are the parents' choices? They can say, "O.K., I'll wait for you," which leads to untold misery and a quick undermining of the family dynamic, because now the sister, who was all ready to get into the car, asks if she can start a game too. The parents can say, "Quit without saving," which even parents know is a mortal sin — besides, it can easily lead to an hour of tears. Or the parents can say, "No more games for you anytime within an hour of when another activity is planned." Which is, when you think about it, exactly what happens, because it's the only option available.

Given the mistakes I've seen and heard, I think it's a damn miracle that games are even allowed in homes with kids. So pay attention to the lessons of your industry's history, and maybe you can make a bundle and save the world and a family or two. 🍴



# Achieving Real-Time Realistic Reflectance

## Part 2

---

**JAN KAUTZ** | Jan is a Ph.D. student at the Max-Planck-Institute for Computer Science in Saarbrücken, Germany. His main research area is interactive realistic lighting and shading using graphics hardware. He can be reached at [kautz@mpi-sb.mpg.de](mailto:kautz@mpi-sb.mpg.de). **CHRIS WYNN** | Chris is an OpenGL software engineer working at Nvidia Corp.'s technical developer relations group. You can ask him anything (BRDF and otherwise) at [cwynn@nvidia.com](mailto:cwynn@nvidia.com). **JONATHAN BLOW** | Jonathan prefers the AK-47 and the Colt M4A1. He will use the MP5, though, if that's what it comes down to. He reads e-mail sent to [jon@bolt-action.com](mailto:jon@bolt-action.com). **CHRIS BLASBAND** | Mr. Blasband has more than 17 years of experience in applying BRDF phenomenology to military and commercial applications. He can be contacted at [cblasban@flash.net](mailto:cblasban@flash.net). **ANIS AHMAD** | Anis is an undergraduate student at the University of Waterloo, majoring in computer science. You can contact him at [a3ahmad@student.math.uwaterloo.ca](mailto:a3ahmad@student.math.uwaterloo.ca). **MICHAEL MCCOOL** | Michael, who can be reached at [mmccool@cgl.uwaterloo.ca](mailto:mmccool@cgl.uwaterloo.ca), is an associate professor with the Computer Graphics Lab at the University of Waterloo, Canada. His research areas include real-time hardware-accelerated shading and illumination.



In last month's article ("Achieving Real-Time Realistic Reflectance," January 2001), we presented the necessary background on BRDFs (bidirectional reflectance distribution functions) and reflectance. This month, we will detail the separable decomposition technique and describe how it can be used to implement sophisticated real-time per-pixel lighting models on current graphics cards.

## Real-Time Hardware-Accelerated Techniques

**N**ow, how can we render materials with sophisticated BRDFs in real time? As we have seen, a BRDF is a four-dimensional function representing the reflective properties of a material.

A BRDF could be sampled using grids of incoming and outgoing directions and the results of all possible parameter combinations placed in a large 4D lookup table. This is the most general format, and the way measured data is often presented. However, this approach has several drawbacks for hardware rendering. To get a decent degree of accuracy and quality, numerous incoming and outgoing directions are needed and this results in an extremely large table. While the space requirements may, in some cases, be suitable for one or two BRDFs, the number of materials used in a typical game makes this an infeasible approach.

Fortunately, better real-time techniques for implementing reflectance models have recently been developed. One such approach, developed by Wolfgang Heidrich and Hans-Peter Seidel (see For More Information), uses view-independent pre-filtered environment maps to produce isotropic reflectance effects. This approach is simple and effective, and can obviously handle environment maps, but it is really only suitable for isotropic reflection models.

A second technique, separable decomposition, was introduced by Jan Kautz and Michael McCool. This is the technique we'll

describe in this article. It factors BRDFs into simpler terms that are then multiplied together using multi-pass rendering or multi-texturing. It can be considered a compression technique for BRDFs that uses graphics hardware for decompression. This technique is suitable for use with anisotropic reflectance models and can be used to generate approximations for measured data.

However, it can only be used with point or directional light sources.

---

**Separable decomposition  
can be considered a  
compression technique  
for BRDFs that uses  
graphics hardware  
for decompression.**

---

## The Separable Decomposition Technique

**S**eparable decomposition as a BRDF approximation technique is a two-step process. First, as a pre-process, the four-dimensional BRDF of choice is decomposed into one or more pairs of two-dimensional functions that are stored as textures. Either measured data or analytical models can be used, as the approximation techniques are purely numerical — in the case of an

analytical model, we just sample the reflectance function. Then, during actual rendering, these textures are parameterized with texture coordinates that depend on the orientation of the surface relative to a given viewer and to a given light source. The results of the texture lookups are multiplied together, and then multiplied with the results of the usual Lambertian lighting model. This technique requires only simple multi-pass or multi-texture operations, which are supported on almost all current consumer-end accelerators. The results, as seen in Figure 3c, are images generated in real time that are virtually indistinguishable from those generated by per-pixel software evaluation using the original BRDF.



## Generating a Separable Decomposition

Our goal in generating a separable decomposition is to take the BRDF and create a set of  $2N$  “subfunctions” which, when combined appropriately, will approximate the original BRDF:

$$f(\bar{x}, \bar{y}) \approx \sum_{j=1}^N p_j(\bar{x})q_j(\bar{y})$$

We can generate a full decomposition that is in fact exactly equal to the original BRDF, within the limits of numerical precision. Unfortunately, such an exact representation would require too many terms to be practical for game applications. However, we can arrange for most of the important features of the BRDF to be contained in the first few terms of the preceding series, and the sum can be truncated to give a good representation. In fact, in many cases, only one term is enough.

In this equation,  $f$  is the original BRDF and  $p_j$  and  $q_j$  are the result of the decomposition. The parameter vectors  $\bar{x}$  and  $\bar{y}$  are reparameterized versions of the incoming and outgoing directions, respectively. Reparameterization will be discussed later; assume for now that both  $\bar{x}$  and  $\bar{y}$  are 2D vectors. The first step in decomposing a BRDF is to tabulate it into a large matrix:

$$M = \begin{bmatrix} f(\bar{x}_1, \bar{y}_1) & \cdots & f(\bar{x}_1, \bar{y}_n) \\ \vdots & \ddots & \vdots \\ f(\bar{x}_n, \bar{y}_1) & \cdots & f(\bar{x}_n, \bar{y}_n) \end{bmatrix}$$

Here, the sequences  $(\bar{x}_1, \bar{x}_2, \bar{x}_3, \dots, \bar{x}_n)$  and  $(\bar{y}_1, \bar{y}_2, \bar{y}_3, \dots, \bar{y}_n)$  each represent a selection of appropriately spaced parameter values.

There are two approaches to decomposing this matrix: the singular value decomposition (SVD) algorithm and the normalized decomposition (ND) algorithm. The SVD approach is more general and produces better decompositions, and produces a series which can approximate a BRDF to arbitrary accuracy, but it is more complex and consumes a fair amount of resources. It also produces signed factors, and so requires signed arithmetic for reconstruction. This is a problem. Since we'll be reconstructing the BRDF by putting the terms into texture maps that are modulated and added together, we would

require graphics hardware that supports arithmetic on negative values, which currently is a rare thing. Signed arithmetic can be simulated on current graphics hardware, but it's painful.

The ND approach produces single-term decompositions that aren't typically as good as those generated by an SVD, but they are often good enough. The results from the ND tend to be oversmoothed, but this is not necessarily a bad thing if the BRDF data is noisy. However, the ND approach is far simpler than the SVD, it consumes much less memory, and it automatically produces positive factors. In fact, rather than filling out the entire preceding matrix, the ND can work on the matrix one row at a time. We will describe only the ND approach, as it is of the most immediate practical benefit. See For More Information to find resources on the SVD algorithm.

To compute a normalized decomposition, first compute the average  $\|m_k\|$  of every row  $m_k$  of the initial matrix,  $M$ . This will give you a vector of  $N$  values. This vector can then be put into the 2D function  $p_1(\bar{x})$ . The next step is to divide each row of the matrix by its average value to get a normalized row. Finally, average all normalized values in each column to construct one row vector, and put the result into the function  $q_1(\bar{y})$ . The product  $p_1(\bar{x})q_1(\bar{y})$  is a single-term separable approximation to the BRDF.

## Choosing an Appropriate Parameterization

In order to achieve good results, the parameterization for the BRDF must be carefully chosen. By parameterization we mean the parameter space in which the BRDF is evaluated. The first constraint is that it must be possible to interpolate the parameters using linear interpolation, since this is what texture coordinate interpolation does. This eliminates a number of otherwise useful parameterizations. Furthermore, if the important features of the BRDF do not align with the axes of the parameterization, the separable decomposition will cause significant blurring. This blurring is particularly visible when the ND approach is used. Unfortunately, no single parameterization of BRDFs is ideal

for all cases, because of the variety of physical phenomena that can influence reflectance. We will discuss two parameterizations that do, however, support many surface types of interest.

The most intuitive parameterization (and the one used in most definitions of the BRDF) simply uses the incoming and outgoing directions  $\hat{\omega}_i$  and  $\hat{\omega}_o$  (expressed relative to a local surface coordinate frame). An example of a BRDF that separates well under this parameterization is velvet. The reflectance of velvet is characterized by self-shadowing and self-masking, and for this type of material the  $(\hat{\omega}_i, \hat{\omega}_o)$  parameterization generally works well. For glossy types of surfaces, however, this parameterization is not accurate and something different is needed.

The following parameterization (which we will call the half-vector parameterization) not only leads to good separability of glossy BRDFs, it also results in texture coordinates that can be interpolated correctly across the surface. The first step in this approach is to find the normalized half-vector,  $\hat{h}$ . The half-vector is the vector halfway between the incoming direction and outgoing direction. In this case, the incoming and outgoing directions must be expressed relative to the same coordinate system that the surface tangents and normal are expressed in, though not relative to them. We will use the notation  $\hat{v}$  and  $\hat{\ell}$  for these “global” view and light vectors. With that caveat, we compute the “global” half-vector as follows:

$$\hat{h}_g = \frac{\hat{v} + \hat{\ell}}{|\hat{v} + \hat{\ell}|}$$

This vector, re-expressed relative to the local surface frame  $(\hat{t}, \hat{s}, \hat{n})$  will be used as our first parameter:

$$\hat{h} = \begin{pmatrix} \hat{h}_g \cdot \hat{t} \\ \hat{h}_g \cdot \hat{s} \\ \hat{h}_g \cdot \hat{n} \end{pmatrix}$$

To obtain the second parameter, a new reference frame is first created such that the half-vector is the new “vertical” polar axis. Two new tangent vectors  $\hat{t}_h$  and  $\hat{s}_h$  must be created and oriented so they are perpendicular to this new polar axis but



as close to the old tangents as possible. This can be accomplished by orthonormalizing the tangents of the original local surface frame:

$$\tilde{t}_h = \hat{t} - (\hat{t} \cdot \hat{h}_g) \hat{h}_g$$

$$\hat{t}_h = \frac{\tilde{t}_h}{|\tilde{t}_h|}$$

$$\hat{s}_h = \hat{h}_g \times \hat{t}_h$$

Now we express  $\hat{\ell}$  with respect to this new coordinate frame and get our second parameter:

$$\hat{d} = \begin{pmatrix} \hat{\ell} \cdot \hat{t}_h \\ \hat{\ell} \cdot \hat{s}_h \\ \hat{\ell} \cdot \hat{h}_g \end{pmatrix}$$

Our reparameterized BRDF now depends on  $\hat{h}$  and  $\hat{d}$  instead of  $\hat{\omega}_i$  and  $\hat{\omega}_o$ . This yields much better separability for many "glossy" BRDFs that peak when the half-vector is near the normal,  $\hat{n}$ . It is possible that other parameterizations can be found that result in even better separability for certain reflectance models and/or are faster to compute. Be sure to let us know.

## Separable Decomposition and Half-Vector Parameterization

In order to clarify the relation between the decomposition algorithm and BRDF parameterization, we will walk through how the half-vector parameterization is used together with decomposition.

As before, we sample the BRDF into a matrix, but this time using the half-vector parameterization:

$$M = \begin{pmatrix} f(\hat{h}_1, \hat{d}_1) & \cdots & f(\hat{h}_1, \hat{d}_n) \\ \vdots & \ddots & \vdots \\ f(\hat{h}_n, \hat{d}_1) & \cdots & f(\hat{h}_n, \hat{d}_n) \end{pmatrix}$$

For discrete values of  $\hat{h}$  and  $\hat{d}$  we sample our BRDF and store it in the matrix  $M$ . Since BRDFs are usually given in the  $(\hat{\omega}_i, \hat{\omega}_o)$  parameterization, we have to calculate  $\hat{\omega}_i$  and  $\hat{\omega}_o$  from  $\hat{h}$  and  $\hat{d}$  in order to

be able to look up the BRDF value in our BRDF:

$$\hat{t}_r = [1, 0, 0]^T$$

$$\hat{t}_h = \text{norm}(\hat{t}_r - (\hat{t}_r \cdot \hat{h}) \hat{h})$$

$$\hat{s}_h = \hat{h} \times \hat{t}_h$$

$$\hat{\omega}_i = d_x \hat{t}_h + d_y \hat{s}_h + d_z \hat{h}$$

$$\hat{\omega}_o = 2(\hat{h} \cdot \hat{\omega}_i) \hat{h} - \hat{\omega}_i$$

Here  $d_x$ ,  $d_y$ , and  $d_z$  are the coordinates of  $\hat{d}$ . Now we can use  $\hat{\omega}_i$  and  $\hat{\omega}_o$  to sample our BRDF with  $f(\hat{\omega}_i, \hat{\omega}_o)$  and store the value in the matrix. Then we can apply the ND algorithm discussed previously.

## Putting the Factors into Texture Maps and Texture Coordinate Generation

The separable decomposition results in two two-dimensional functions,  $p_1(\bar{x})$  and  $q_1(\bar{y})$ . As we want to put these functions into textures, we have to map them into a 2D texture space in some way that gives reasonable interpolation. Both of the parameterizations presented depend on unit vectors varying over the hemisphere. In all cases we will have represented these unit vectors with coordinates relative to some local coordinate frame, either the frame given by  $\hat{n}$ ,  $\hat{t}$ , and  $\hat{s}$ , or the frame given by  $\hat{h}_g$ ,  $\hat{t}_h$ , and  $\hat{s}_h$  (for  $\hat{d}$ ). The coordinates of  $\hat{h}$ ,  $\hat{\omega}_i$ , and  $\hat{\omega}_o$  are taken relative to  $\hat{n}$ ,  $\hat{t}$ , and  $\hat{s}$ , and are computed with dot products against these vectors. For example, the coordinates of  $\hat{h}$  are computed with  $\hat{h} = [(\hat{h}_g \cdot \hat{t}), (\hat{h}_g \cdot \hat{s}), (\hat{h}_g \cdot \hat{n})]^T$ , where  $\hat{h}_g$  is the half-vector in world coordinates. Likewise, we've already shown how  $\hat{d}$  is in fact just the light direction parameterized with respect to  $\hat{t}_h$ ,  $\hat{s}_h$ , and  $\hat{h}_g$ .

Now we must map these local coordinates into texture space. There are several ways to do this: hemisphere maps, parabolic maps, and (on graphics accelerators that support them) cube maps. Both hemisphere and parabolic maps are stored in 2D texture maps, are easy to set up, and will work on any hardware that supports 2D texture maps (in other words, everything), but cube maps will give better

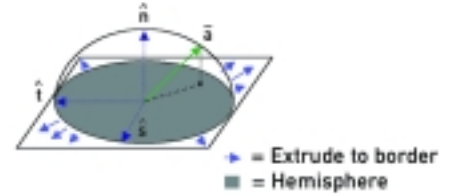


FIGURE 1. Texture map.

interpolation in some situations. To store the factors in hemisphere maps, simply use the following as texture coordinates (see Figure 1), where  $\bar{a}$  is either  $\hat{\omega}_i$ ,  $\hat{\omega}_o$ ,  $\hat{h}$ , or  $\hat{d}$ :

$$u = \frac{1 + \bar{a}_x}{2}$$

$$v = \frac{1 + \bar{a}_y}{2}$$

Unfortunately, the hemisphere map has problems when  $\bar{a}_z < 0$ . In theory this can't happen for true surfaces for the parameterizations we will give, but it can happen in practice when polygonal approximations to surfaces are used with vertex normals, and these normals align badly with the surface. Also, the hemisphere map has poor resolution near its edge. A slightly better mapping of the unit hemisphere onto the unit square of texture coordinate space is the parabolic map, given by

$$u = \frac{\bar{a}_x}{2(1 + \bar{a}_z)} + \frac{1}{2}$$

$$v = \frac{\bar{a}_y}{2(1 + \bar{a}_z)} + \frac{1}{2}$$

This looks a lot more complicated, but in fact you can just use the three dot products as homogeneous texture coordinates, with  $\bar{a}_z$  as the homogeneous coordinate, and then set up an appropriate texture transformation matrix to compute the parabolic map:

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 \end{pmatrix} \begin{pmatrix} \bar{a}_x \\ \bar{a}_y \\ 1 \\ \bar{a}_z \end{pmatrix}$$

If cube maps are supported, these are really the best representations for functions defined over a hemisphere, since

interpolation will be over great arcs whereas linear interpolation in hemisphere and parabolic maps only approximates this. The benefits of using cube maps are particularly noticeable for BRDFs that contain a significant amount of anisotropy or when the light direction or view direction is nearly parallel to the illuminated surface.

Unfortunately, with all these maps, some space is inevitably wasted. In all cases, the “edge” of the function defined over the hemisphere should be extended (that is, by extrapolating the value at the edge of the hemisphere) into the “unused” part of the map to avoid interpolation artifacts. These artifacts can occur for two reasons. First, the bilinear interpolation of texture maps may pick up values from outside the hemisphere. Second, for parabolic and cube maps, the generated texture coordinates may lie outside the normal range if  $\vec{a}_z < 0$ , although again this should be an unusual occurrence.

## Half-Vector Parameterization, Hemisphere Maps, and Texture Coordinate Generation

Now it's time to show how the texture map representation and the BRDF parameterization go together. Assuming we use a hemisphere map and the half-vector parameterization, texture coordinates are computed the following way for a given light and viewing direction:

$$\hat{h}_g = \text{norm}(\hat{v} + \hat{\ell})$$

$$\hat{t}_h = \text{norm}\left(\hat{t} - (\hat{t} \cdot \hat{h}_g)\hat{h}_g\right)$$

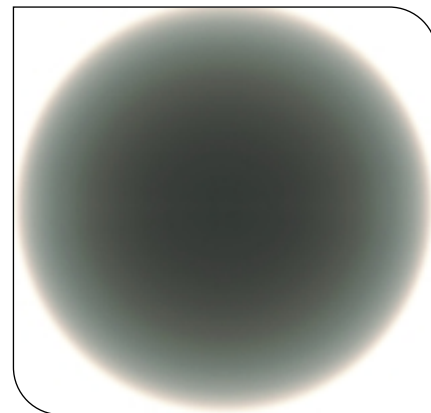
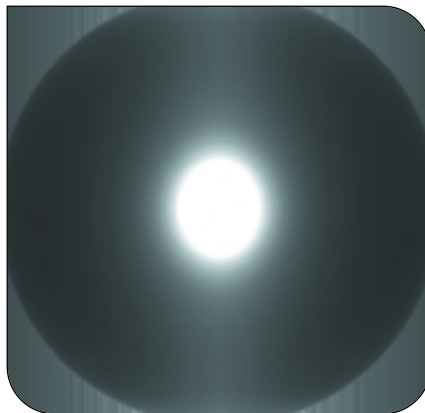
$$\hat{s}_h = \hat{h}_g \times \hat{t}_h$$

$$h_u = \frac{\hat{h}_g \cdot \hat{t} + 1}{2}$$

$$h_v = \frac{\hat{h}_g \cdot \hat{s} + 1}{2}$$

$$d_u = \frac{\hat{\ell} \cdot \hat{t}_h + 1}{2}$$

$$d_v = \frac{\hat{\ell} \cdot \hat{s}_h + 1}{2}$$



FIGURES 2A (left) & 2B (right).  $\hat{h}$  and  $\hat{d}$  texture maps using the hemisphere map parameterization.

Please note again that  $\hat{\ell}$  and  $\hat{v}$  are the light and viewing direction in “global” coordinates, whereas  $\hat{\omega}_i$  and  $\hat{\omega}_o$  are by convention relative to the local surface coordinate frame. The vector  $\hat{h}_g$  is the half-vector between  $\hat{\ell}$  and  $\hat{v}$  in “global” coordinates. The value pairs  $(h_u, h_v)$  and  $(d_u, d_v)$  are the texture coordinates for the factors  $p_1(\hat{h})$  and  $q_1(\hat{d})$  that should have been put into hemisphere maps. This above computation has to be done at every vertex of an object using a separable approximation with the half-vector parameterization and the hemisphere map representation. Part of the computation of  $(h_u, h_v)$  and  $(d_u, d_v)$  can be done with appropriate texture transformation matrices, although this probably won't be faster unless you have hardware T&L.

## Rendering with the Separable Approximation

To use the separable approximation for direct lighting, we'll revisit the point-source reflectance equation discussed in last month's article. This time, however, we will replace the BRDF with an  $N$ -term separable approximation, use an expansion for  $K$  light sources, and use our current vector notation:

$$L_o(\hat{v}) = \sum_{k=1}^K \left[ \sum_{j=1}^N p_j(\hat{h}_k) q_j(\hat{d}_k) \right] L_{i_j}(\hat{\ell}_k) (\hat{n} \cdot \hat{\ell}_k)_+$$

In order to compute the radiance of the light source multiplied by the positive cosine of the angle between the incoming direction and the surface normal (the term  $L_{i_j}(\hat{\ell}_k) (\hat{n} \cdot \hat{\ell}_k)_+$ ), we will simply rely on the diffuse component of the Lambertian

model, which is already supported by existing APIs, and for which Gouraud shading works well. Note that the Lambertian reflectance model also multiplies by zero the parts of the reflectance model that face away from the light source.

Since multi-texturing is widely supported on gaming platforms and leads to significantly higher performance for this technique, we will describe the steps required to render a single-term separable approximation using one point light source in one pass using multi-texturing (enhancements and gotchas will be discussed following):

0. Place the two factors ( $p_1(\hat{\omega}_i)$  and  $q_1(\hat{\omega}_o)$  or  $p_1(\hat{h})$  and  $q_1(\hat{d})$ ) into separate textures  $t_0$  and  $t_1$ . (See Figures 2a and 2b.)

Then, for each frame:

1. Compute the two vector parameters and generate corresponding texture coordinates for each vertex of the model (that is,  $\hat{\omega}_i$  and  $\hat{\omega}_o$  or  $\hat{h}$  and  $\hat{d}$  and then apply the UV texture coordinate mapping).
2. Enable the diffuse component of the point source to handle Lambertian lighting.
3. Enable the texture units  $t_0$  and  $t_1$ .
4. Set up the multi-texture combiner unit to compute  $t_0 * t_1 * \text{fragment color}$ .
5. Draw the object specifying  $(u_o, v_o)$  and  $(u_1, v_1)$  as texture coordinates.

This efficiently evaluates the reflectance equation for all points on the object — assuming single term approximation and one point light source. (See Figures 3a–c.) In order to render the separable approximation on hardware that doesn't support multi-texturing, steps 3 through 5 would be



FIGURES 3A–C (left to right). Teapot rendered with  $\hat{h}$  texture (A), with  $\hat{d}$  texture (B), and with  $\hat{h}$  texture \*  $\hat{d}$  texture \*  $\cos(\theta)$  (C).

replaced with two separate passes and the multiplications would be performed using compositing operations in the frame buffer.

## Implementation for Games

To use separable decomposition in a game, we need to have source data of the material types we wish to render, so that we can create our texture maps. As mentioned in last month's article, some collections of measured data, such as the CURET archive, are already available (see For More Information). By choosing the appropriate analytic function and sampling it, this technique can simulate many other materials.

For added visual effect, we can combine a texture-mapped BRDF with gloss maps or bump maps on the same surface. However, what we really would like is for the bumps on the material to respond to light in accordance with our BRDF. Unfortunately, to accomplish this we would need to repeat our per-vertex computation at every pixel and use dependent texturing (a feature expected in the next generation of graphics hardware which allows you to use texture coordinates stored in one texture to reference another) to get bump maps with arbitrary BRDFs.

As we mentioned earlier, we can use ordinary texture maps to provide varying color over the material's surface to draw things such as the grain pattern of wood. Whether the BRDF on a surface should modulate or add to the surface color depends on the type of reflectance that is being encoded; diffuse reflectance should modulate the surface color, and specular reflectance should add to it. For materials that have important

diffuse and specular components, we will want to divide their reflectances into two BRDFs, so that we can use one to add and the other to modulate.

## Enhancements

As noted, a diffuse texture map may be added to the result to give the surface more detail, especially if the BRDF has only a near-specular component. For this pass, normal Lambertian lighting can be used, which also fills in the ambiently illuminated part of the model. An alpha map can also be used to modulate the regions where multiple BRDFs are applied, so a single surface can have many BRDFs in a generalization of texture mapping probably best termed "material mapping."

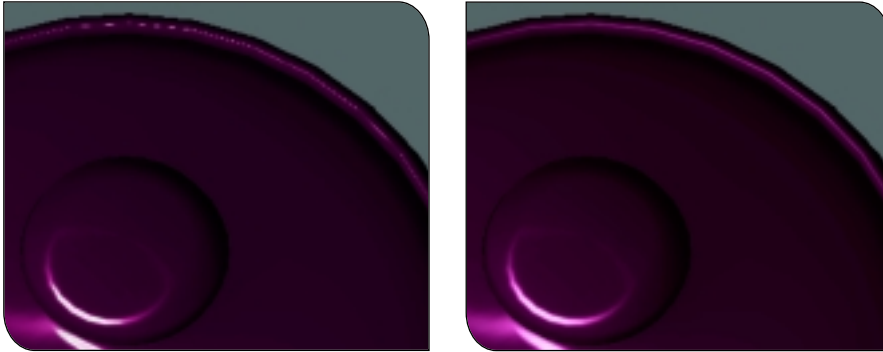
Some BRDFs are a sum of several effects (for example, the reflectances of human skin and certain paints are a combination of surface specularity and subsurface scattering). For these kinds of BRDFs it may not be possible to get a single separable decomposition to work well, but a sum of two decompositions parameterized in different ways should work. In other words, you may have to layer reflectances to get more subtle overall effects. You can fit such BRDFs numerically by finding an approximation with one parameterization, subtracting that approximation from the original data, then finding another for the residual, iterating if necessary.

To support lighting from multiple light sources, the separable approximation needs to be rendered multiple times, once from each light source. Note however that the "ambient/diffuse" passes and some of the computations (such as the normalization of the view vector) can be shared.

Finally, one advantage of the separable approximation technique is that it can be easily antialiased by using MIP-maps for each factor of the decomposition. This prevents highlight aliasing when the curvature of the surface is too high relative to the pixel spacing by automatically choosing a broader representation of the BRDF exactly when needed. (See Figure 4.) You should generate the levels of the MIP-map by smoothing and downsampling the BRDF itself before generating a separate approximation for each resolution level of the MIP-map. As an approximation, you can also just blur and downsample the factors themselves in the usual way. This approximation reconstructs a slightly incorrect reflectance for the downsampled factors because the integral of a product is not the same as the product of the integrals. However, preventing aliasing is far more important visually than getting the reflectance exactly right, so do per-factor filtering anyway if for some reason you can't smooth and decompose the original BRDF data itself at different scales. It should be noted that good reflectance model antialiasing is also possible with other texture-based reflectance techniques, such as the prefiltered environment map technique mentioned briefly earlier.

## Gotchas

Currently, separable approximations should only be used when necessary. While good frame rates can be achieved, separable approximation is still relatively costly, since the texture coordinates depend on the light source direction and view direction, and computation of the parameters has to be done on the host at



FIGURES 4A (left) & 4B (right). Close-up of teapot with and without MIP-mapping. Figure A is without MIP-maps and shows aliasing. Figure B uses MIP-maps and shows no aliasing.

the moment. In the near future, extensions to hardware APIs will support some additional per-vertex “shader” programmability which could be used to support rendering with separable reflectance models (see, for instance, Nvidia’s vertex shader extensions to OpenGL).

Most models don’t come with per-vertex tangents, most modeling programs don’t export them, and existing APIs don’t know how to transform them. Ideally, there would be a “Tangent” call added to APIs as well to support those new texture-generation modes. In the meantime, even if you transform tangents yourself (or back-transform the light and view direction instead), there is the problem of adding tangents to object models. If a spline model is used (such as the teapot used in our examples), tangents can be found by evaluating partial derivatives. In fact, the normals for spline models are usually evaluated by taking cross products of these tangents. For polygonal models, take a “global” tangent and orthonormalize it against the normal at each vertex to get per-vertex orthonormal tangents. This works especially well for surfaces of revolution, where the global tangent can be taken as the direction of the axis of the model. If a model has already been texture-mapped by a 3D artist, then the tangents can also be extracted from the texture maps. Just take the texture coordinate vector  $(u, v) = (1, 0)$  at each vertex and transform it from texture space into model space. Another technique to generate tangents has been developed by Nvidia in the context of bump mapping but it is also applicable here (see Nvidia’s web site for more details).

In general, though, there is the problem that it is impossible to give a smooth tangent-space parameterization of arbitrary closed surfaces in 3D. This is similar to


the texture-mapping parameterization problem, and should have similar “solutions” in practice.

A final issue is dynamic range and precision. BRDFs can vary over 0 to infinity, whereas current graphics hardware computes with values only in the range 0 to 1. To get the BRDF computation to “fit” in the available dynamic range, it is necessary to scale the factors of the decomposition down and scale the result back up after multiplication. Since modern multi-texturing units support scale factors of two or four in a single pass, making the product of the scale-down factors  $1/2$  or  $1/4$  is convenient. This scaling unfortunately loses precision and makes it hard to do certain high-dynamic-range BRDFs well. This restriction, as well as the lack of signed arithmetic (which inhibits use of the singular value decomposition), has the potential to be removed with future generations of consumer-level graphics hardware.

## Conclusions

**W**e hope to have shown the benefits of using separable approximations to improve lighting in real-time applications. The technique can be used to render many interesting reflectance models, including anisotropic models, with anti-aliasing. It scales over a range of cost-performance trade-offs. It fits well into a multi-texturing, multi-pass game rendering engine, and can be layered with other effects like specular maps. Finally, it can be implemented on practically every existing installed graphics accelerator.

Jonathan Blow of Bolt-Action Software has written an OpenGL demo of this technique, which was used to generate the images in this article. You can download the source code for this demo from *Game*

*Developer’s* web site at [www.gdmag.com](http://www.gdmag.com). The demo works on ATI Rage and Nvidia TNT2 and GeForce cards. Nvidia also has an introductory tutorial on BRDFs as well as sample source code that demonstrates the technique available at [www.nvidia.com/developer](http://www.nvidia.com/developer). Surface Optics Corp. is building a commercial database of measured BRDFs, which will be available soon; see [www.surfaceoptics.com](http://www.surfaceoptics.com) for information. 

## FOR MORE INFORMATION

### PAPERS

Heidrich, W., and H. P. Seidel. “Realistic, Hardware-Accelerated Shading and Lighting.” *Proceedings of SIGGRAPH 1999*. pp. 171–178.

Kautz, J., and M. McCool, “Interactive Rendering with Arbitrary BRDFs Using Separable Approximations,” *Rendering Techniques ‘99* (Proceedings of the 10th Eurographics Rendering Workshop). New York: Springer-Verlag. pp. 281–292.

### WEB SITES

CURET BRDF Database  
[www.cs.columbia.edu/CAVE/curet](http://www.cs.columbia.edu/CAVE/curet)

Cornell BRDF Measurements  
[www.graphics.cornell.edu/online/measurements](http://www.graphics.cornell.edu/online/measurements)

University of Waterloo Computer Graphics Lab  
[www.cgl.uwaterloo.ca/Projects/rendering](http://www.cgl.uwaterloo.ca/Projects/rendering)

Jan Kautz’s BRDF Page  
[www.mpi-sb.mpg.de/~jnkautz/projects/hw\\_bidir](http://www.mpi-sb.mpg.de/~jnkautz/projects/hw_bidir)

Nvidia  
[www.nvidia.com](http://www.nvidia.com)





# The Past, Present, and Future of PC Mod Development

**S**ince the dawn of PC games, players have always hacked on them. The tinkerer nature of the PC gamer encouraged them to try to figure out how to add magic items to games, tweak levels, and hack the high-score list. *NETHACK* was probably the first game to promote modification and to have widespread distribution of a modified version of itself. It has been in postpartum development now for 15 years, a process which still continues today. The gameplay continues to evolve and grow, reaching unprecedented depth. A similar thing happened with id Software's *DOOM*. Players figured out how to create their own levels, then distributed them to extend the game's multiplayer lifespan. When id saw how much players modified *DOOM*, they intentionally built their next game, *QUAKE*, to be user-modifiable. After *QUAKE* was finished, id released some of the tools they had used to create maps, including a mini-language that let players script new behavior into the game. That's when the PC "mod movement" really started, and id Software is largely credited for promoting it.

Since *QUAKE*, id Software, Epic Games, Valve Software, and others have all been promoting "mod development," or game modifications, by designing their games to be easy to change. In exchange for creating, releasing, and supporting content creation tools, providing occasional informal technical support, and letting registered owners of their games use their engine for any noncommercial purposes they desire, these companies have dedicated fans creating unforeseen variants of their games. Modifiable games remain popular for longer, appeal to more people, and blur the line between game player and game creator. Making games "open" and modifiable also increases sales: almost all mods require the original game in order to be played. Therefore, if you find a good free add-on for a particular game, you might go out and buy the base game just so you can play it. Recently, both Valve and Epic have profited from releasing newly packaged versions of their games which include content developed for free by their player communities.

The term "mod" now refers to any code modification of a game and is usually a combination of new levels, game rules, and artwork. The "mod community" refers to programmers, game designers, artists, musicians, and level designers ("mappers") involved with using a published game engine in a new way. There are fan-created levels, weapons, models, "skins," sounds, and gameplay variants available for free. Sometimes new, complete games are written, games with no recognizable attributes from the original host game. These "total conversions" (TCs) are all written with an existing game's engine and tools. A "mod platform" is a host engine and game that a mod can be written for, such as *UNREAL TOURNAMENT* or *HALF-LIFE*. This kind of movement to modify and expand was bound to happen on the PC, where players are technically savvy and have keyboards, mice, Internet access, and a tolerance for delayed gratification.

The mod movement represents the reasons why many are attracted to game development in the first place. For me, it's self-expression, music, breathtaking visuals, and drama. It's also the chance to spend all my waking hours and thoughts working on an experience that could give a sense of awe and wonder to players. Creating a mod is all about good ideas and content creation, not technology development and compatibility testing. Most

---

**CHARLIE CLEVELAND** | *When he isn't leading an underground game development coup d'état, Charlie can be found working on PC strategy games at Stainless Steel Studios in Cambridge, Mass. The "dictator of freedom" can be reached at flayra@overmind.org.*



LEFT. *OPPOSING FORCE* for *HALF-LIFE* features smart companions and a good story in the *HALF-LIFE* tradition.

of mod development is spent on visible results: game design, non-engine code, art, levels, music, and sound.

## Significant Mods

**T**EAM FORTRESS for the original *QUAKE* engine was one of the first mods to gain widespread popularity. Players are divided into two teams, each with its own fortress to defend. Inside each fortress is a flag which the enemy tries to capture. Most importantly, *TEAM FORTRESS* introduced the “class” system, where players can choose the role they want to play, such as soldier, engineer, or medic, each with its own special abilities and attributes. *TEAM FORTRESS* is one of the earliest and most influential mods.

An innovative but underplayed mod is the atmospheric and elegant *GLOOM*, for the *QUAKE 2* engine. All players choose to be either an alien or a marine, whose purpose is to take out the enemy’s base. Players score points which can be spent to respawn as a more powerful alien, or a marine with better weapons. The production values are top-notch and every alien has a unique feel and tactics.

The most popular mod of all time is *COUNTER-STRIKE*, a realistic terrorism and counter-terrorism game for the *HALF-LIFE* engine. Players choose to play as a terrorist or counter-terrorist and participate in a number of objectives. *COUNTER-STRIKE* is important because it shows how a talented mod team can make a game that is fully accepted into the mainstream. Just as Valve brought the first-person shooter into the mainstream by making the superlatively paced *HALF-LIFE*, *COUNTER-STRIKE* brings online play to the masses, using the *HALF-LIFE* technology as a springboard. Today, *COUNTER-STRIKE* has more people playing it online at any given time than all of the other first-person shooters combined.

While most mods are twists or extensions to deathmatch or team-based first-person shooters, there are also mods created for other genres. There is at least one driving simulation (*QUAKE RALLY*), *QUAKE* chess (*QUESS*), and a real-time strategy game (RTS

*QUAKE*). There are also entire single-player campaigns with new stories in new settings (*THEY HUNGER*, *OPPOSING FORCE*). Most mods are first-person shooters because most mod platforms’ tools, source code, and fan-run servers are all heavily biased toward this genre. It takes talent and patience to try to make these architectures work for other genres.

## Making the Right Choices

**C**hoosing an engine as a mod platform can be a difficult task. The three all-around best choices right now are the *UNREAL TOURNAMENT* engine, the *QUAKE 3: ARENA* engine, and the *HALF-LIFE* engine.

**UNREAL TOURNAMENT.** With its beautiful engine, powerful IDE (UnrealEd), and easy-to-learn UnrealScript, the *UNREAL TOURNAMENT* engine is a solid mod platform that is the most accessible for beginning mod developers. If you don’t want to stray far from standard FPS gameplay, there is no easier choice. If you need something more powerful than UnrealScript, it also has native bindings to let you use C or C++ to build game code. Any mod that is created using UnrealScript ships with the source code, meaning that most mods are public and modifiable, a boon for new programmers.

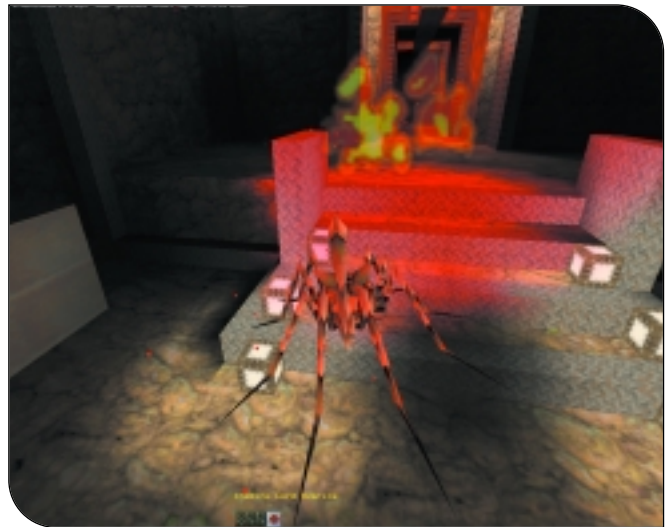
**QUAKE 3: ARENA.** It still has the best graphics out there. *id* started the whole mod movement and still boasts the most numerous and highest-quality mods for any platform. The *QUAKE 3* SDK is a well-designed interpreted C which means free cross-platform support on Mac OS, BeOS, IRIX, and possibly even next-generation consoles. Because it uses C, it isn’t as well suited to amateur programmers as UnrealScript is, but it is very powerful. There is also a large and loyal *QUAKE* community, making it easier to find talent and resources.

**HALF-LIFE.** The *HALF-LIFE* engine is showing its age, but the upcoming *TEAM FORTRESS 2* engine will bring it up to date and should be backwards-compatible with existing levels and tools.



LEFT. COUNTER-STRIKE for HALF-LIFE is the most popular mod ever made.

RIGHT. GLOOM is a dark and innovative strategic shooter for the QUAKE 2 engine.



While id got the mod movement going, Valve is currently the company most committed to the mod community. They regularly update their SDK and have even been known to fund promising mod teams. It's written in C/C++, so it's best for more experienced programmers. The HALF-LIFE mod platform currently boasts many thousands of players, many more than the other platforms. Finally, it is the only mod platform to allow complete mod ownership and resale without negotiations or paying for a license.

## Alternative Platforms

There are other platforms to consider if you have more specialized needs. If your main goal is to learn how game engines are created, the QUAKE I source code is available for free under the GPL (General Public License). Huge outdoor mods are possible with the TRIBES or the upcoming TRIBES 2 engine, both of which are heavily modifiable. If role-playing and interactive storytelling are your main goals, Nihilistic's VAMPIRE: THE MASQUERADE — REDEMPTION or Bioware's upcoming NEVERWINTER NIGHTS could serve your needs best. For real-time strategy mods, DARK REIGN 2 is the most extensible, but TOTAL ANNIHILATION is heavily modded as well.

## What Does It All Mean?

Along with mods and mod tools came a new way to create games — the distributed development team. Since mod creators generally work in their spare time on specialized projects, many mod teams have members that have never seen each other in person. To stay organized, some teams use CVS or SourceForge to organize their development, both of which are free. Some teams have meetings over ICQ or IRC to discuss development challenges and deadlines. The distributed development team means more (and more specialized) game development efforts are feasible, but communication is challenging.

A significant benefit of the mod community is that it can serve as a gateway to the professional game development industry. It has always been hard to land your first game industry job. Companies require experience, but there usually isn't any way to get experience without being hired. Mods can help job-seekers get around this catch-22, because a hard-working and talented person or mod team can more easily create something that is technically competitive and gets noticed by game companies. Mods bring back the grassroots style of game development, enabling amateurs to "just do it." If you want to show off your game creation skills, creating a mod is the perfect way to get noticed and get into the industry.

The process for creating a mod is essentially the same as creating a game at a game company. Of course, you won't have to deal with a publisher's schedule and creative demands, but not every game company has to deal with that, either. The important things are those hard skills that take years of experience and working on many titles to learn: writing design documents; controlling feature creep; creating, adapting, and communicating a clear game vision with the rest of the team; creating a cohesive art vision; play-balancing and player feedback; design iteration; tweaking game responsiveness and "feel"; cutting features that don't (or no longer) fit; managing a team; and, to an extent, public relations and marketing. Creating your own game mod is the fastest way to learn game development, period.

It is also likely to be as close to a "pure" game development experience as you will ever have, with virtually all effort going directly into game creation instead of finances, publisher and human relations, and running a company. Leveraging mod technology instead of writing a game from scratch means a small team can compete with other titles on the market, and it also greatly increases the team's chance of finishing the project. Because engines have recently become so complex, mods mean that, for the first time in many years, a small team can compete with gaming veterans. The core COUNTER-STRIKE team is just two people, and most mod teams are well under ten people.



Independent mod development teams can innovate regardless of financial pressures or market desires. Even mod teams that get publishers (another exciting trend) are more likely to remain autonomous because the publisher recognizes that as a strength. Diverse talent, fresh ideas, and freedom from time-consuming technology development puts mod authors in the unique position to push the art form forward. Climbing production costs for traditional game development ensures more publisher intervention to protect their interests. Additionally, many mod platforms have a larger receptive audience than the average traditional title on crowded store shelves. Unless you have a high-profile title, you're likely to have a smaller receptive audience with a published game than with a mod on an established and popular platform. This is especially true for multiplayer, where it isn't uncommon for a really good game to only have a mere ten or twenty people playing online at any given time, due to marketing or product-placement problems. Leveraging an existing engine means gaining access to thousands of fan-run servers and many thousands of players familiar with and loyal to the base game. To some extent, mod platforms even have the standardization advantage of consoles: the host game is already installed, configured, and running acceptably. The mod should, too.

## Mods of the Future

If the past is any indication, the future of mods is bright. A handful of mod teams have recently landed publishing deals, possibly achieving the holy grail of independent but funded development. As mods grow in popularity and their social and financial benefits become obvious, game companies outside the FPS genre might open up more to the mod community. Mods will become easier to create, submit, organize, and download. As mods fuel sales of their host games, those games will popularize the mods by packaging, selling, and supporting them, although this will probably remain unique to the PC. With the arrival of next-generation consoles that have writeable mass storage, keyboards, and Internet access, there could be mod activity on non-PC platforms as well, but it's hard to imagine console gamers staying up all night furiously coding a new game on a joypad in their living room.

Just as the web is moving toward more interactivity, more gamers will use the game tools to design and create, enriching their own game experiences and expanding their skills. Promoting the community around a game could become as high a priority as creating the game in the first place. Instant messaging, e-mail, and web site creation tools are being added to some games in development in order to promote mod development and tighter community bonds.

## Lots of Potential

Mods can be used for many purposes. They can be used for prototyping new game ideas, learning how games work, getting a job in the industry, creating a great game in record time, or just for fun and experimentation. While the capabilities of a mod platform seem limiting, they are extensible. The artificial intelligence, user interface, physics, and even networking and

graphics can be rewritten or extended. Most importantly however, unlike traditional game development, the initial lack of these systems doesn't prevent or slow development. All team members can experiment and contribute immediately, with new systems added as you go. Mod development can be faster, more experimental, more creative, and potentially more profitable than traditional game development. Games developed as mods have the potential to be created in only a few months but still compete with or exceed traditional games in every respect. The mod movement embodies the innovation and spirit of PC game creation and it's happening now. Viva la revolución! 🎮

### FOR MORE INFORMATION

#### MODS

COUNTER-STRIKE

[www.counter-strike.net](http://www.counter-strike.net)

GLOOM

<http://gloom.teamreaction.com>

OPPOSING FORCE

[www.sierrastudios.com/games/opposingforce](http://www.sierrastudios.com/games/opposingforce)

TEAM FORTRESS

[www.planetfortress.com/teamfortress](http://www.planetfortress.com/teamfortress)

NEANDERTHAL

[www.overmind.org/neanderthal](http://www.overmind.org/neanderthal)

QUESS

<ftp://ftp.zdnet.com/gs/action/quake/quess12.zip>

ALIENS VS. HUMANS

[www.overmind.org](http://www.overmind.org)

UNREAL Mods

[www.planetunreal.com/modcentral](http://www.planetunreal.com/modcentral)

QUAKE Mods

[www.planetquake.com/motw](http://www.planetquake.com/motw)

HALF-LIFE Mods

[www.half-life.net/triggerhappy/mdatabase.html](http://www.half-life.net/triggerhappy/mdatabase.html)

#### PROGRAMMING

HALF-LIFE coders list (VorteX)

[www.topica.com/lists/hlcoders](http://www.topica.com/lists/hlcoders)

HALF-LIFE SDK and tools

[www.planethalf-life.com/half-life/files](http://www.planethalf-life.com/half-life/files)

QUAKE 3 SDK and tools

[www.planetquake.com/quake3/files.shtml](http://www.planetquake.com/quake3/files.shtml)

UNREAL Technology Page

<http://unreal.epicgames.com>

#### OTHER MOD PLATFORMS

DARK REIGN 2

[www.pandemicstudios.com/dr2](http://www.pandemicstudios.com/dr2)

TRIBES

[www.sierrastudios.com/games/tribesplayers](http://www.sierrastudios.com/games/tribesplayers)

VAMPIRE: THE MASQUERADE — REDEMPTION

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

**POSTMORTEM** *eric gross with ryan touchon*



**Humongous  
Entertainment's**

**BACKYARD  
SOCCER  
MLS  
EDITION**

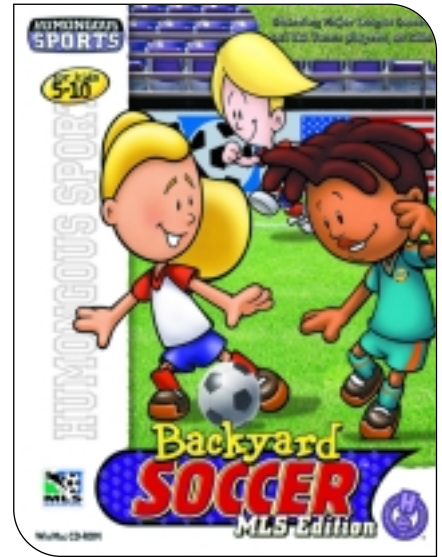


**K**nown for creating engaging interactive content for children, Humongous Entertainment broke new ground in 1998 when it introduced an entirely new genre to the world of interactive sports games for kids.

Sports simulations were nothing new on the PC, but even to experienced game players and avid sports fans, the overall experience out of the box was rarely an enjoyable one.

Controls could eventually be mastered and the players would probably stumble upon most features of interest, but only after a dedicated effort and a significant amount of time invested. We knew that kids loved sports, and it was obvious that they were being completely left out by the current offerings, so our goal was to design a series of great sports simulations designed for our younger audience. These products would need to be not only true to their sports, but also easy to navigate, simple to control, and most of all, fun to play.

By the time our first Humongous Sports title was completed, almost every adult in our studio was addicted to BACKYARD BASEBALL, and we knew we had a winner on our hands. The game featured an intuitive menu flow with a very kidlike feel, and a simple mouse-driven interface that allowed kids to jump right into a game with almost no instruction or ramp-up time. The game enabled the youngest of our players to laugh and click their way through entire games on the easiest setting, while featuring enough gameplay and statistic-tracking features to hold the interest of even the most sophisticated sports addict. BACKYARD SOCCER was the next entry in the series, which continued to push our development system to its limits, while helping to establish our line of fun, exciting sports games for children of all ages.



## GAME DATA

**NUMBER OF FULL-TIME DEVELOPERS:** 1 lead artist, 4 art subleads, 16 artists, 1 lead programmer, 3–4 programmers, 1 QA lead, 4 testers

**NUMBER OF CONTRACTORS:** 5–8 ink-and-painters, 1 writer, 1 musician, 9 voice actors, 1 additional tester

**LENGTH OF DEVELOPMENT:** 6 months

**PROJECT LENGTH:** 174 code files between 100 and 15,892 lines each; 3,075 art files; 6,678 sound files; and 13,765 voice files

**RELEASE DATE:** September 4, 2000

**INTENDED PLATFORMS:** Windows 95/98; Mac OS 7.5.3 and up.

**CRITICAL DEVELOPMENT HARDWARE:** 450MHz Pentium PCs.

**CRITICAL DEVELOPMENT SOFTWARE:** Lightwave, Photoshop, Debabelizer, Codewright, and a host of proprietary art and archival software

**NOTABLE TECHNOLOGIES:** Humongous Entertainment's proprietary scumm language and sputm engine

**ERIC GROSS** | Starting at striker for The Programmers is Eric “The Red Menace” Gross. Standing 5’10” and weighing in at 170 pounds, Eric brings to the field speed, agility, and a penchant for full-contact programming. His first foray into the Humongous world of Backyard Sports was as co-designer and co-lead on 1998’s original BACKYARD SOCCER. He also filled the role of lead programmer on FREDDI FISH 4: THE HOGFISH RUSTLERS OF BRINY GULCH, then co-designed and led the programming team of the hit children’s adventure PAJAMA SAM 3: YOU ARE WHAT YOU EAT FROM YOUR HEAD TO YOUR FEET.

**RYAN TOUCHON** | And on the other side of the field, starting in goal for The Artists is Ryan “Billybubba” Touchon. Stretching the tape at 6’2” and 168 pounds, Ryan brings everything he has to both work and play (frequently leaving parts of himself in his wake). Starting as a storyboarder, he worked his way up to 3D animation lead and then to lead artist. He has worked on every sports title that Humongous Entertainment has released: BACKYARD BASEBALL, BACKYARD SOCCER, BACKYARD FOOTBALL, BACKYARD BASEBALL 2001, and BACKYARD SOCCER MLS EDITION.

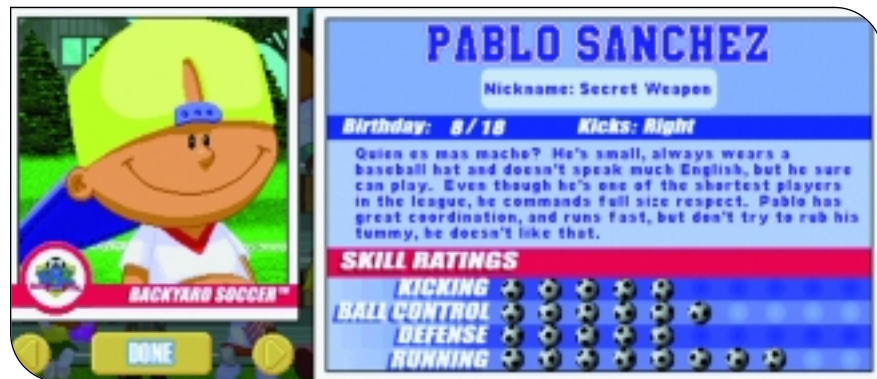
The next change to the Backyard Sports line was the introduction of major-league team logos and professional players, as kids, to our neighborhood rosters, which in our case came in the form of Major League Soccer players and members of the U.S. Women's National Team. Just like our 30 Backyard Kids, each pro came with his or her own unique attributes and characteristics. Also planned for BACKYARD SOCCER MLS EDITION were head-to-head play, keyboard and gamepad support, art enhancements taking advantage of 16-bit color, and an entirely new navigation and menuing system, which was patterned off of the intuitive and successful new system that the BACKYARD BASEBALL 2001 team had just developed.

Ryan Touchon and I agreed to lead what I would later jokingly refer to as the flaming train-wreck from hell. Our schedule was finalized, our teams were assembled, and what follows is a synopsis of what we learned.

## What Went Right

**1 Experienced people are invaluable.** If there was any single factor responsible for the successful release of this product, our team was it. The BACKYARD SOCCER MLS EDITION development team comprised three main groups: art, programming, and quality assurance.

The art team had five 3D animators responsible for bringing our 45 on-field kids into the wonderful world of 3D. Not only were the animators experienced, but they were also well acquainted with the look and feel of the Backyard World, were



ABOVE. Pablo Sanchez's official player card.

familiar with their tools, and were even all seasoned sports animators. They consistently met or beat their deadlines, which allowed us to identify and resolve various small problems with our approval process early on, and kept a constant flow of art to the programmers. They rallied for one brief crunch to hit an overly aggressive milestone, which meant one less inevitable negative ripple effect on the team.

We also had six 2D animators who were charged with creating the concept art, player cards, thumbnails, team photo poses, and bleacher animations for the pro kids. Again, almost all of these artists were extremely experienced. They avoided a group crunch and instead took it upon themselves individually to work any extra hours that were required to keep the team on track, and hit every one of their collective milestones.

A small but dedicated subset of these groups was responsible for the game's amazingly elaborate intro backgrounds and animations. They were allowed a great deal of creative freedom, which contributed to the motivation of everyone

involved. This resulted in the team constantly taking it upon themselves to push the artistic level without prompting from their leads.

Also included in the art team was a four-person group that specialized in all aspects of the game's menus, interface, and logos. The scheduling for this group faced significant hurdles (more on this later), but they ultimately turned out some amazing work.

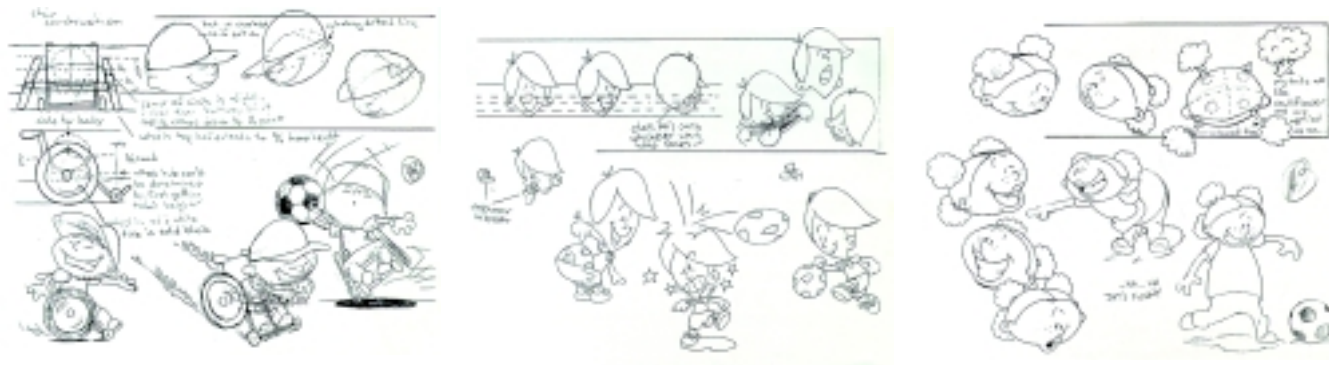
Rounding out the art side was our vaunted staff of ink-and-painters, who did a beautiful job of cleaning up and bringing color to our mountain of 2D animations. Each of these groups had a dedicated sublead, and each of these leads reported to Ryan, our art lead.

For most of our cycle, my programming team consisted of three senior programmers. We were able to grab one last engineer just in time for the final month of development. The programming team was some of Humongous Entertainment's finest programming talent. Their skills and experience (each had worked on at least one other sports title) were huge fac-



RIGHT. Detail from a character sketch for Marky Dubois.





LEFT TO RIGHT. Character studies of Kenny Kawaguchi, Reese Worthington, and Keisha Phillips.

tors in actually hitting our scheduled release date. One of the benefits of a team with this level of experience and dedication is their ability to multi-task. When problems arose, be they art fixes or issues with bits of legacy code, the programmers wasted no time in contacting the responsible team member or lead, then delving into whatever task was next on their list of priorities. The leads helped complete the circle by ensuring that issues were resolved quickly, and that details of the resolution got back to the programmers immediately thereafter. This team was handed a non-revision-friendly code base, full of redundant arrays, vague and undocumented variable names, and hard-coded assumptions. They had to log ungodly hours to make up for the aggressive schedule we finally established (discussed later) and still managed to keep their heads and put out a fantastic soccer game.

I also worked closely with our QA lead, Eric Snyder, whose team was made up of a core of four testers, with others rotating through when possible. Eric had just come off of BACKYARD BASEBALL 2001 and was well prepared for the design we were

implementing. His team did excellent work despite not having received a fully playable version of the game until very late in the process.

**2 • Our team structure worked well.** We stuck with our established project-leadership structure, which continually proves to be effective. The roles of each lead and sublead are well defined and understood. On the art side, Ryan allowed a high level of creative flexibility, which took full advantage of the artistic talents of his subleads and their teams. As a result, the majority of the smaller issues were resolved without having to go through multiple levels of approval. This was possible only because of the level of self-motivation exhibited by the individual team members.

The lead structure is even simpler on the programming side. We had a small and talented team of senior programmers working closely together and reporting to an experienced lead programmer. Our small team size and high level of experience were both keys to our success. With so few engineers involved, there was little to no confusion over who was responsible for what por-

tions of the code. As you might also guess, there was a fair amount of overlap with interlocking aspects of the game, so the programmers were encouraged to work closely together. They took this sense of teamwork to another level, and used breaks from their own work to lend a hand when an extra head was needed to puzzle through another programmer's perplexities.

They also had a great familiarity with our development process and the roles our various artists played. When minor art fixes or additions were needed, my programmers were able to approach the appropriate artists directly. By being able to describe their requirements directly to the artist, and without having to explain the problem to and wait for the associate producer and all of the leads for every little tweak (copying them on an e-mail message was sufficient), art was ultimately able to get into the game quicker. Another bonus to this scenario was that the programmers didn't spend as much time with their hands tied, waiting for others to evaluate and respond to their requests, empowering them to be responsible for and accomplish more.

**3 • Outside resources were helpful, professional, competent, and a pleasure to work with.** Every contact we had with our Major League Soccer representative was sheer joy. Not only were they businesslike and efficient where MLS interests were concerned, but they showed a great deal of interest in the overall development of our game. They, more than any other outside resource, had the potential to devastate our already har-



LEFT. Keisha's bio and skill ratings, compiled from dozens of unique characteristics.



**ABOVE.** Players see 20 different fields and five different surfaces — grass, dirt, cement, sand, and indoor turf — with the appropriate material-specific physics and player characteristics. For example sand affects stamina, while cement discourages side tackles.

ried timeline and compromise our release date. They not only completed their reviews and approvals of the various MLS-related aspects of our game in a timely fashion, but were thorough enough in each of their play-throughs so that no one was faced with the all too common “I know we signed off on that beta, but we didn’t notice how big that guy’s eyebrows were. Please re-do his face everywhere it appears in the game” scenario. The individual pro players’ agents also demonstrated a degree of professionalism and alacrity that was not only appreciated, but vital to our constrained timeline.

For the game’s music, our musician read us perfectly. The music for the original game had been written and performed by my co-lead, Rhett Mathis. As one of the lead designers, he had obviously been intimately aware of our audio needs. Unfortunately,



**LEFT.** Marcelo Balboa

Rhett’s time and musical skills were unavailable to us, and we were forced to look outside of our team and studio.

Luckily, Tom McGurk, a talented musician whom I had recently worked with on *FREDDI FISH 4: THE CASE OF THE HOGFISH RUSTLERS OF BRINY GULCH*, was available. Ryan and I were looking for exciting new intro music and a number of pads for the pro kids that needed to be in the *BACKYARD* style, but also called for a slightly harder, hipper edge. We met with Tom for about an hour, gave him all of Rhett’s original music, picked out a slew of examples from our favorite artists, passed along some timing specs, and he was off. The pads he brought in at his first milestone were almost all right on the money. This was the first good sign. Even more promising was his ability to interpret our relatively vague and muddled suggestions and provide at his next milestone exactly what we were looking for. In keeping with his first two milestones, and to our great pleasure, the balance of his work was delivered on time (a good thing, since the rest of our development was done in a

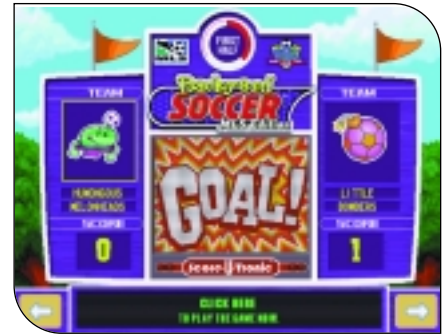


frenzy with no room for additional scheduling issues), and was the perfect complement to our new group of characters.

In addition to the outstanding work from our musician, the recording studio that we’ve grown to love over the years, Seattle-based *Bad Animals*, came through for us again. First, they set us up with a great pool of talent to choose from for the casting of our pro kids. We saw children as young as five and adults of all ages, with nothing in common other than a strong sense of professionalism. The voice talent we selected, a few of whom hadn’t had the opportunity to amass much experience (due primarily to conflicts with their heavy grade-school schedules), were great sports and lots of fun to work with. They came through for us like seasoned veterans, checking yet another potential time (and money) sink off of our list. *Bad Animals’* engineers also continued their tradition of working seamlessly with our leads, and everything from auditions to final pick-ups was exactly what we needed them to be — flawless.

**4 • Good internal team communication.** Whether it’s a tribute to our entire team’s experience, professional-





ABOVE LEFT. Pick-team screen. TOP RIGHT. Scoreboard screen. BOTTOM RIGHT. Penalty kick attempt against Mr. Clanky in single-player practice mode.

ism, or personalities, all of the individual aspects of development worked extremely well together. Our programmers felt comfortable approaching the artists with minor questions, issues, suggestions, and vice versa. Having worked together before in many cases, or at least in the same development environment for an extended period of time, they also had a high level of knowledge of and confidence in each other's abilities. Problems could be discussed openly and succinctly, leading to swift and accurate resolutions.

QA could stop by and chat about areas of the game that were troublesome, while the programmers never hesitated to offer suggestions on where to look for potential problems. This led to quicker bug isolation, and kept the amount of programming work lost from compounding faulty code to a minimum. The art, interface, and programming leads were all soccer junkies and intimately familiar with the sport, but not all of the team members were necessarily soccer fans. The important thing was that everyone had Backyard Sports experience, giving us a common language and understanding, which served as a shared frame of reference allowing us to communicate more effectively and keep the need for explanations of background and context to a bare

minimum. This level of communication and ease of teamwork made the day-to-day work that much more enjoyable, greatly aided the workflow, and saved countless hours of production. Even our marketing, sales, and creative services departments were familiar enough with our characters and sports line that communication between departments about our game was virtually trouble-free.

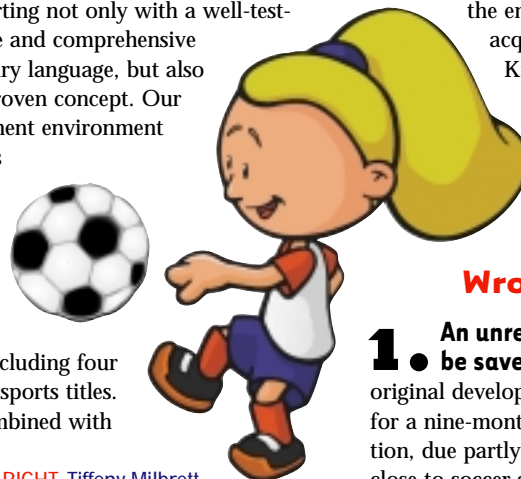
**5** ● **Our proven technology and concept meant two fewer things to worry about.** We had the benefit of starting not only with a well-tested engine and comprehensive proprietary language, but also with a proven concept. Our development environment and tools have undergone dozens of production cycles, including four previous sports titles. This, combined with

our confidence in the engineers that maintain our system, allowed for our relatively short QA cycle to focus primarily on the higher-level art, design, and gameplay issues. Even the last-minute additions of high-color support and relatively late implementations of gamepad support and head-to-head play had seen some production test time.

We were pleased with the way the original BACKYARD SOCCER had turned out. With much of the general gameplay already established, our tasks were easier to identify and attack. In addition, the entire art team was very well acquainted with our Backyard Kids' physical characteristics and personalities, which made designing circumstantial actions and reactions effortless.

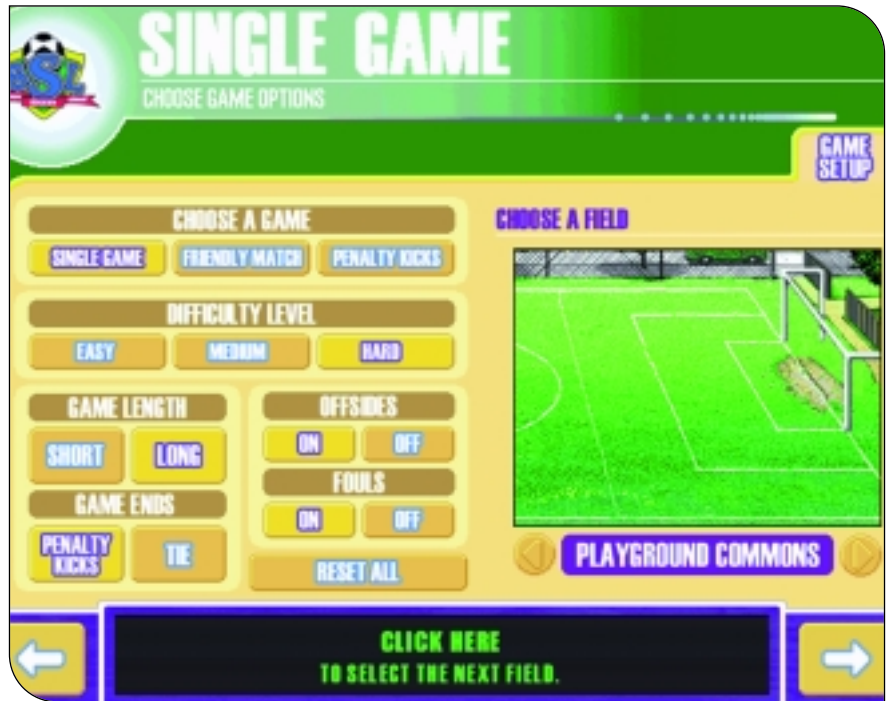
## What Went Wrong

**1** ● **An unrealistic schedule can't be saved without pain.** Our original development schedule was slated for a nine-month cycle. Shortly after inception, due partly to our desire to release as close to soccer season as possible, the cycle



RIGHT: Tiffeny Milbrett.





TOP LEFT. Clubhouse screen. BOTTOM LEFT. Season-mode team page with navigational tabs to the Statistics, Trophy Case, Schedule, Standings, League Leaders, and Game/Control Options screens. ABOVE RIGHT. Single game setup.

was shortened to six months, with little change in design and no additional staffing. At this point, our schedule was obviously too tight. It didn't allow for any unanticipated setbacks that could cause milestones to be missed. When we did miss them, it created a huge crunch time and resulted in inadequate testing during the final phases of production. There was no way to buy extra time at the end of production, when we needed it most.

Unfortunately, no additional staff was available, so the only way this situation could have been alleviated was by removing features from the design. Another option might have been postponing our title in favor of another, but other circumstances made this impossible for us to consider.

Given the choices we made, one decision could have eased our pain immensely had we chosen to take a different route. When confronted with our revised release date, we chose to dive right into production in an effort to maximize our available staffing resources. This came at the expense of a full complement of preproduction aids. Failure to complete our design documents cost us more production time in the end than we would have spent on finishing this vital stage of development. The immediate hit of another week or two of idle programmers and artists would have been a

far wiser choice than the countless hours spent fixing and refixing flawed and incomplete design on the fly.

**2 ● Late arrivals and double duty aren't tolerable.** Several factors combined to force a late arrival of the all-important interface team, but the main contributor was a previous project's missed deadline. Several weeks of unrecoverable production time were lost right at the start. This combined with the lack of preproduction time created a very large hole to climb out of.

The interface lead arrived after three months of severe crunch on his previous project. To make matters worse, his time was divided between two products for nearly the entire run of the project. Unfortunately, no one else was available to take over his role on the second project, so he was forced by default into being the interface lead on two projects simultaneously. Both deserved his full attention, so he was constantly forced to compromise.

Had we spent the time to complete our preproduction, we would have realized what a huge undertaking the interface was going to be. With that knowledge, Ryan would have recognized the need to devote more of his energy toward the menu-art layouts and design, taking some of the bur-

den off of the interface lead's shoulders.

In addition, our main in-house tools weren't quite ready for high-color applications. This caused some pretty big headaches when dealing with palettes and the various conversions and manipulations necessary. Trying to take a 16-bit image and put it into a program that only works with 8-bit images required adding several interim steps. While none of these steps was terribly difficult or time-consuming, they added an additional layer of work to an already overcrowded schedule.

**3 ● Lost team members need to be addressed immediately.** We had a number of staffing issues. The first, and one of the most severe, was the loss of our producer. She had been the driving force behind the MLS license, and was the one person keeping abreast of the overall development of the game in relation to the other titles in our Backyard Sports line. Shortly thereafter, our associate producer was out of the picture as well. One of his primary duties had been to track the flow of art files, so his departure created a very dangerous void.

Compounding this misfortune was the very short notice we were given with both of these departures. We did bring on a new producer, but she ended up wearing both

the producer's and associate producer's hats, and was further hampered by insufficient time to be briefed adequately by the departing team members.

Less severe but still enormously problematic was an agreement that we would shift our test team's focus to another product for a short period of time. This was in consideration of the other game's imminent release date, and with the understanding that we would get their time back, with interest. As you might guess, we never did get that time back, and QA ended up even farther behind than they already were, given our accelerated schedule. This resulted in a game that wasn't fully testable until very late in the cycle. The obvious result of all of these factors was frantic testing with minimal coverage toward the end of development.

#### 4 ● Conventions should have been better documented, communicated, and adhered to.

Although we had established workflow documentation to aid in our day-to-day file tracking, it wasn't complete. Most of the art files had their routes in place: animation went from the artist to production for scanning, back to the artist for cleanup, then off to the lead for approval, on to the programmer, and so on. The routing for the menu-art files hadn't been addressed, which resulted in files being sent to the wrong people or not being sent at all, and contributed to the overall confusion and breakdown of the entire system. Not helping matters was our lack of standardized terms for a number of our new menu-related features.

We contributed to the confusion by failing to establish the order that our new team names and logos should appear in their various art files until weeks before release. The effects of losing our associate producer, who was in charge of keeping tabs on all of these files, would have been minimized had we finalized and enforced our workflow conventions. Determining standard names for all of our new art elements and setting sequences for our logos and team names up front would have saved us a fair amount of frustration and lost time as well. These three items are no-brainers, and would have been covered during the course of normal preproduction, so this turns out to be yet another

tribute to the importance of solid and complete design.

#### 5 ● We underestimated the importance of focus and morale.

During the production of BACKYARD SOCCER MLS EDITION, Humongous Entertainment was undergoing substantial management changes. Ownership of the company had recently changed hands, resulting in a state of upheaval and a number of morale issues for the staff. This resulted in a high level of distraction and cost our team focus during the period of restructuring.

An example of one of the more minor elements affecting the team's focus was the implementation of new office assignments. This company-wide reorganization into separate studios and teams, while a positive change overall, came near the end of production, when most of the BACKYARD SOCCER team was in massive crunch.

We should have recognized focus and morale as top priorities. Instead, we underestimated the level to which focus and morale issues could affect project productivity. In our attempts to address both project- and restructuring-related priorities, we neglected to address the issue of team morale sufficiently. Employee burnout is a very real phenomenon but can largely be averted by paying careful attention to the ebb and flow of employee satisfaction. Managers and leads must work together to develop and adhere to realistic schedules and recognize all potential morale issues as their highest priorities.

#### The Big Picture

**B**ACKYARD SOCCER MLS EDITION had a great team of talented artists, testers, and programmers and overcame many obstacles to put out a clean, fun game on time and under budget. Our efforts have certainly been recognized, as the game continues to sell extremely well. As I write this, BACKYARD SOCCER MLS EDITION is currently PC Data's number-one best-selling chil-

RIGHT: Jorge Garcia gets a wake up call.

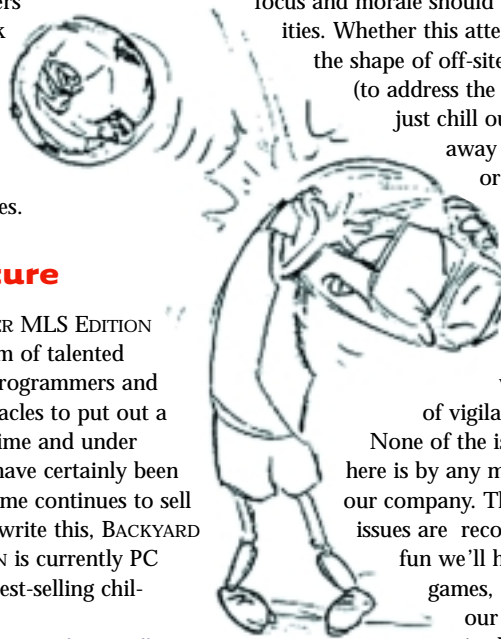
dren's software title alongside our other recent sports titles, BACKYARD BASEBALL 2001 and BACKYARD FOOTBALL. Even so, as I've demonstrated, there are some clear lessons that can be learned from our experience in developing this product.

First and happiest is the not-so-surprising news that talented and experienced developers, both full-time and contract, are invaluable. Also, a well-conceived and established organizational structure within the project plays a positive role on many levels. Cutting down on confusion is its most obvious benefit, but the positive influence it has on overall team communication is also undeniable.

No matter how short a production cycle is, complete design and preproduction is vital. I know this sounds like common sense, but the reality is that shaving time off of the front end of your schedule to pull your release date back is one of the easiest traps to fall into. The pitfalls you will head off and redundant work you will avoid, not to mention the stress and wasted time associated with filling design holes on the fly, will more than make up for the time required to complete the design and preproduction phases of development.

The project development process can be a fair source of anguish all by itself. If outside influences are adding to the overall stress level of your team members, their focus and morale should become top priorities. Whether this attention comes in the shape of off-site team meetings (to address the problems or to just chill out and decompress away from the office), or constant one-on-one contact to ensure that your team members aren't neglected in any way, some form of vigilance is vital.

None of the issues discussed here is by any means unique to our company. The more these issues are recognized, the more fun we'll have making games, and the better off our industry will be in the long run. ☺



# Surviving Children's Software

**C**hildren's software isn't what it used to be, and perhaps that is all to the good. Cagey old vets in the children's software industry can be defined as having at least five years' total experience and having worked at a minimum of two companies.

Perhaps the second sentence somewhat illuminates the first one. I remember being flabbergasted, after three years' experience, at being referred to as "seasoned." I came from academia, where it is not unknown for someone to spend several years mulling the first chapter of a publication, several weeks refining a sentence, and where research is not a trip to the bookstore for three seventh-grade-reading-level books on a given subject. Stepping from the abstract, cool passion of the brain-bending activities of scholarship to the ship-at-alpha, action-item, oh-they-cancelled-it, what's-the-minimum-config, can-you-pull-in-a-quarter world of software development is impossible to describe, so I won't try.

I do know, however, why I did this to myself (apart from the idea that I could make money, I mean). When you're in theater graduate school, it's impossible to tell what the quality of a thing is by the way it is described. Nearly everyone talks a good game, but when they stand up to perform, it either is or isn't good, is or isn't entertainment. A software team is a much more immediate kind of place. While teams are not immune to other types of agendas — and we all have our war stories — the industry is still a place where talent is respected among the rank and file, and the end result is made to be touched and used rather than analyzed. The success of the end product is vulnerable to all the generally discussed factors, and a few others that aren't mentioned except in whispers. Yet it's still true that, every once in a while, making things wins out.

The industry is pretty much an ice field that keeps cracking open. We jump from ice floe to ice floe, or reorg to reorg, or, in my case, out of large companies and into the death-defying world of running a development group and making product that actually ships. Attachments to company loyalties, brand identities, and favorite tools, co-creators, or delivery media have to fall away. Can't let go? Well, then you're done. The perfect game stays tantalizingly just out of reach, and there is a large human cost that you can observe in those trying to catch it.

The world of children's software has some additional obstacles. It has been more difficult to capture the budgets, the place on the SKU plan, and the talent for a children's title. Movies, books, and television don't have this problem. Within the captive world of larger companies, not a lot of quality product has been made. At times it seems that the quality of the packaging is in inverse relationship to the worth of its contents. From the some-

what staid but worthy days of educational value, to quasi-educational value (and we've had a lot of that), to entertainment products riding on the coattails of the big brands, we've arrived at a full shelf. A-B-C, 1-2-3, entertain yourself with a large blue dog, a bigger purple dinosaur, or the one and only pink goddess. So

*continued on page 63*






*continued from page 64*

why do many of these games cycle around the same old play patterns? The production values are unnamed to protect the innocent.

CD-ROM was never intended as the be-all or end-all. It was just a big data delivery space with incredibly pokey access. The latest crack in the ice field — and the boom of the ground opening underneath our feet is nearly deafening — is the Internet and the digital toy. It's possible to create content without dedicating a team of six people, ten months (O.K., 14 months), and a million dollars to it. This is both good news and bad news to developers.

The good news is that you can experiment; school's out. The bad news is that most of the old play patterns can be coded in Flash or Java in about a month — and with much the same fidelity of the old CD-ROM games. And so far, it's free. Sure, many families still have one phone line and don't want their kids surfing in the largest unsupervised space of all time, and kids still fall in love with characters and want to interact with them over and over again. The game Concentration is not news, but it's news to the four-year-old who plays it for the first time.

But overall, what does this new phase mean? Well, it means that we finally *have*

to do something different in order to be commercial. Time to put on your skates and start jumping again. Maybe we should not think about what is "fun," or "educational," or "worthy." Maybe we should just look at what is compelling. Compelling can be silly or dark, short or long, 256 colors or made out of sticks. It has a quality that can't be analyzed, but you can see it in the way children — or players of any age — react to it. So while we're surviving, it's good to remember why we're doing it. It's because people are at their best while they're playing, and that's the space in which we're talking to them. 

---

**GANO HAINE** | *Gano is an interactive designer and writer who has worked in the industry since 1991 at companies such as Sierra Online, Electronic Arts, and Mpath Interactive. She has been a co-owner of Stunt Puppy Entertainment since 1996. Stunt Puppy's credits include BARBIE GENERATION GIRL GOTTA GROOVE CD-ROM and BARBIE NAIL DESIGNER. Gano is currently at work on titles for Hasbro Interactive and LeapFrog Toys.*

---