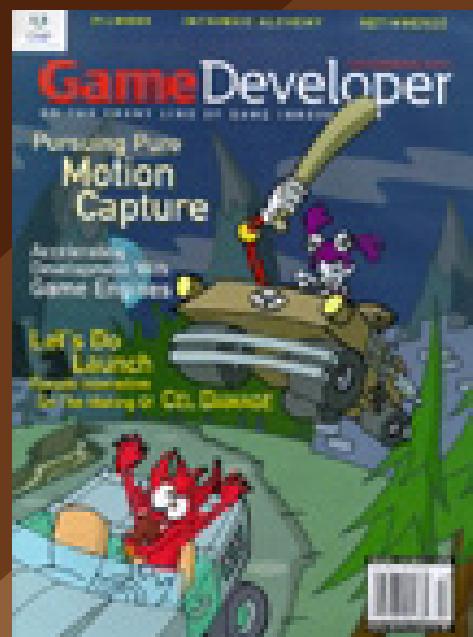


# gd

GAME DEVELOPER MAGAZINE

DECEMBER 2001





# GAME PLAN

LETTER FROM THE EDITOR

## Out to Launch

The long and sometimes excruciatingly anticipated North American launch of Microsoft's Xbox and Nintendo's Gamecube brings to a close one of the longest "transition years" this industry has had to bear; actually, it was more like two years. Dwindling amounts of new software reaching an existing installed base, development projects mired in midstream migrations to different hardware, and hardware shortages of new systems provided the game industry plenty of fodder for excuses for anything less than spectacular performance.

Finally, we'll be able to put all that behind us. Now we sit back and wait to cash in on all our hard work, patience, and sacrifice. Unfortunately, the road we now find laid out before us is anything but smooth.

Cycles of this sort of course aren't unique to the game industry. There are also cycles of peace and prosperity that interleave with cycles of war and economic recession. We entered this transition period at the extreme of one and have come out decidedly at the other.

You've no doubt heard the cheerful reminders that in times of national and economic uncertainty, people flock to entertainment for a low-cost diversion from the weightier realities of life. While this could be good news for the game industry, we still have to compete with movies, television, and other forms of commercial entertainment for a shrinking amount of consumer spending.

Hollywood faced a crisis in the 1960s when the studios, which had been happily churning out relatively uninspired contract-based films in the 1950s, were thrown into a tumult of consolidations and acquisitions by multi-national conglomerates. Studios found it difficult to recoup their skyrocketing production costs (sounding familiar yet?) as audiences increasingly abandoned their lukewarm fare for television. Most offerings did nothing to address the evolution in American tastes in the 1960s, characterized by a more somber national mood forged largely by a slowing economy and a depressingly divisive and drawn-out overseas conflict.

What could have spelled disaster for Hollywood instead spawned a renaissance that sustained the industry throughout the 1970s. Old conventions were thrown out and a bold generation of young filmmakers emerged to challenge audiences and the medium alike with heretofore unheard of content wrapped in films that were meticulously nurtured by their creators as works of art. It turned out to be exactly what would draw audiences back to theaters in droves.

Now that game industry is poised to unleash a torrent of product to potentially languid consumers, are we mindful that not only spending habits but also tastes could be changing? That some established themes may not appeal to audiences forever, or even feel appropriate anymore? It's a pertinent time to contemplate what and when the next renaissance in the game industry should be.

In 1969 there were more than 500,000 American troops in Vietnam, the economy plunged into recession, and *Easy Rider* came out. The film was a both a breakthrough commercial success and a harbinger of an era of unprecedented creativity in the medium. Hopefully our present circumstances will ultimately prove less dire than those of the late 1960s, but given three choices for our industry as the global economy heads south and the U.S. digs in its heels for prolonged military action — do nothing, make a slew of bargain-bin OSAMA GAME HUNTERS, or seize an opportunity to evolve our craft — which would you rather see, and what do you want to be part of?

**Programming Notes.** This month we are pleased to present a new programming column written by longtime contributor Jonathan Blow, "The Inner Product." It replaces "Graphic Content," which has been under the venerable stewardship of Jeff Lander since March 1998. Graphics will still feature highly in "The Inner Product," but look for expanded coverage of topics such as mathematics, networking, and more in the near future. We hope this broader focus serves you better and invite your feedback.

# Game Developer

600 Harrison Street, San Francisco, CA 94107 t: 415.947.6000 f: 415.947.6070

## Publisher

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

## EDITORIAL

### Editor-In-Chief

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

### Managing Editor

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

### Production Editor

Olga Zundel [ozundel@cmp.com](mailto:ozundel@cmp.com)

### Product Review Editor

Tor Berg [tberg@cmp.com](mailto:tberg@cmp.com)

### Art Director

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

Jonathan Blow [jon@bolt-action.com](mailto:jon@bolt-action.com)

Tito Pagán [tpagan@w-link.net](mailto:tpagan@w-link.net)

### Advisory Board

Hal Barwood LucasArts

Ellen Guon Beeman Beemania

Andy Gavin Naughty Dog

Joby Otero Luxoflux

Dave Pottinger Ensemble Studios

George Sanger Big Fat Inc.

Harvey Smith Ion Storm

Paul Steed WildTangent

## ADVERTISING SALES

### Director of Sales & Marketing

Greg Kerwin e: [gkerwin@cmp.com](mailto:gkerwin@cmp.com) t: 415.947.6218

### National Sales Manager

Jennifer Orvik e: [jorvik@cmp.com](mailto:jorvik@cmp.com) t: 415.947.6217

### Senior Account Manager, Eastern Region & Europe

Afton Thatcher e: [atthatcher@cmp.com](mailto:atthatcher@cmp.com) t: 415.947.6224

### Account Manager, Northern California & Southeast

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

### Account Manager, Recruitment

Raelene Maiben e: [rmaiben@cmp.com](mailto:rmaiben@cmp.com) t: 415.947.6225

### Account Manager, Western Region & Asia

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

### Sales Associate

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

## ADVERTISING PRODUCTION

Vice President, Manufacturing Bill Amstutz

Advertising Production Coordinator Kevin Chanel

Reprints Stella Valdez t: 916.983.6971

## GAMA NETWORK MARKETING

Senior MarCom Manager Jennifer McLean

Marketing Coordinator Scott Lyon

Audience Development Coordinator Jessica Shultz

## CIRCULATION



Group Circulation Director Catherine Flynn

Circulation Manager Ron Escobar

Circulation Assistant Ian Hay

Newsstand Analyst Pam Santoro  
is BPA approved

## 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

Executive Vice President & CFO John Day

President, Business Technology Group Adam K. Marder

President, Specialized Technologies Group Regina Starr Ridley

President, Technology Solutions Group Robert Falter

President, Electronics Group Steve Weitzner

President, Healthcare Group Vicki Masseria

Senior Vice President, Human Resources & Communications Leah Landro

Senior Vice President, Global Sales & Marketing Bill Howard

Senior Vice President, Business Development Vittoria Borazio

Vice President & General Counsel Sandra Grayson

Vice President, Creative Technologies Philip Chapnick



GamaNetwork

# INDUSTRY WATCH

daniel huebner | THE BUZZ ABOUT THE GAME BIZ



**Square deals with Sony, cuts movie funding.** Sony Computer Entertainment took a 19 percent stake in Tokyo-based Square. Sony purchased 11.2 million new shares in Square at an 18 percent discount from the trading price. The total value of the investment is approximately \$124 million. Sony said that investing in Square will allow both companies to work together to develop online games, though a Square spokesperson insisted that the deal will not limit the Square's ability to develop games for non-Sony platforms.

Bringing in Sony wasn't Square's only financial move. The company cut funding to its Square Pictures animation subsidiary after the studio's first project, *Final Fantasy: The Spirits Within*, failed to reach revenue targets. Square Pictures president Jun Aida said Square does not plan to sell nor close his studio, which currently employs 125 people. The studio will continue to operate independently on for-hire projects.

**Codemasters appointment in North America.** Codemasters has appointed Sébastien Soulier to the position of president of Codemasters North America. Soulier, who has over 15 years of experience in the entertainment and software industry, was previously the general manager of Codemasters, France. Prior to joining Codemasters, Soulier worked for Infogrames and Ecudis, a French software distributor.

**Infogrames financial picture improves, but still posts loss.** Infogrames Inc., the New York-based subsidiary of the French publisher, showed improvement in its fourth quarter and fiscal 2001 results. Improved sales coupled with cost cutting helped the company trim quarterly losses to \$23 million on revenues of \$72.7 million, compared to a loss of \$55.4 million on revenues of \$48.5 million in the same period one year ago.

Fiscal 2001 revenues of \$310.5 million, a slight dip from last year's \$313.2 million, translated to a loss of \$60.7 million. This year's red ink is a vast improvement from

last year, when the purchase of Hasbro Interactive pushed the company to a loss of \$397.6 million.

**Activision acquires Treyarch.** Activision announced the acquisition of Santa Monica-based developer Treyarch. Treyarch's five teams and 140 employees are engaged in the development of several key Activision titles, including sports games TONY HAWK'S PRO SKATER 2x and KELLY SLATER'S PRO SURFER. Activision purchased Treyarch, which will become a wholly-owned subsidiary, for \$20 million in stock. Treyarch's management and key employees will stay with the company under a long-term contract signed as part of the sale agreement.

**United Developers launches.** Ron Dimant and Mark Dochtermann officially launched United Developers, their Dallas-based development and publishing company. The company, which was originally conceived early last year, plans to offer administration, business and management infrastructure for game development and game publishing. With Dimant serving as managing director and Dochtermann as director of development, the management team is rounded out by Mark Cottam as president and Drew Fisher as director of technology. The first companies joining

under the United Developers banner include ALICE developer Rogue Entertainment, Inertia, and Mac Play.

**OH, BEHAVE, featuring Austin Powers.** published by Take-Two Interactive.

and Electronic Arts are both combating fluctuating share prices with ambitious buyback plans after the Security and Exchange Commission loosened rules on corporate share buying. Electronic Arts announced that it plans to buy back almost

2 million shares of its common stock because of the recent low market conditions, while Nintendo issued plans to repurchase as many as 14 million share of its stock. Take-Two announced undisclosed open market share purchases from a number of key board members.

**Sony cuts PS2 price in Europe, Australia.** Citing better production efficiencies in its Japanese factories, Sony reduced the retail price of the Playstation 2 in Europe and Australia by more than 25 percent, to \$293. The company did not indi-



SPIDER-MAN, on his way to fight evil, published by Activision.



NBA SHOOTOUT 2002, available on Sony's Playstation 2.

cate whether prices elsewhere in the world would be reduced in the near future. Some analysts had predicted that Sony would reduce the PS2 prices in the U.S. prior to the November launch of the Gamecube and Xbox, but Sony flatly denied any cut in U.S. pricing.



**Share buy-back.** Nintendo, Take-Two,

## UPCOMING EVENTS CALENDAR

### DV EXPO

Los Angeles Convention Center  
Los Angeles, Calif.  
December 3-7, 2001  
Cost: variable  
[www.dvexpo.com](http://www.dvexpo.com)

### MACWORLD CONFERENCE AND EXPO

Moscone Center  
San Francisco, Calif.  
January 7-11, 2002  
Cost: variable  
[www.macworldexpo.com](http://www.macworldexpo.com)

# PRODUCT REVIEWS



THE SKINNY ON NEW TOOLS

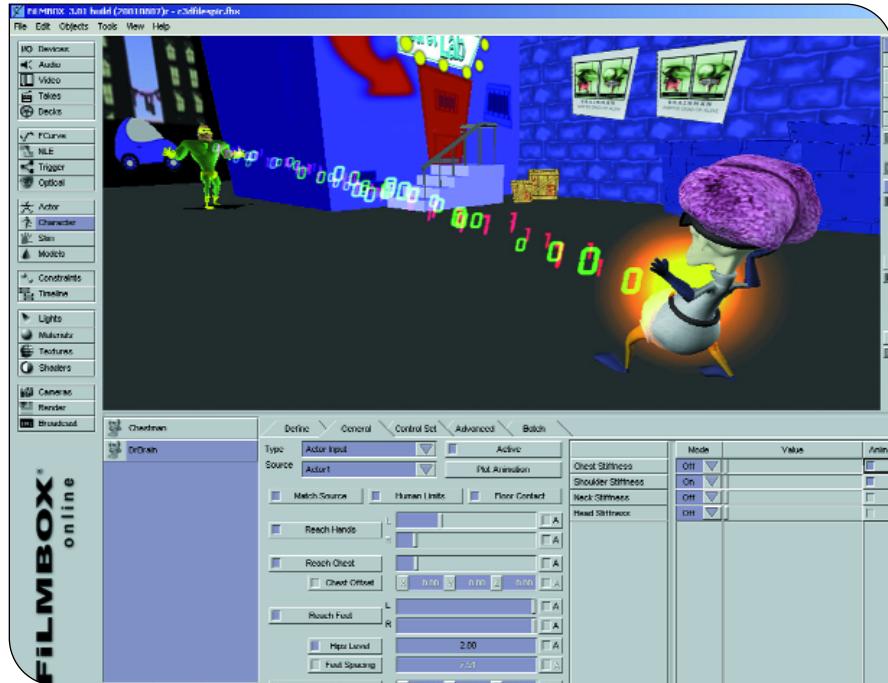
## Kaydara's Filmbox 3.0

by *david stripinis*

**W**hen I mention to colleagues that I work with motion capture, usually their response is "Oh, so you work with Filmbox?" Kaydara's Filmbox, recently upgraded to version 3, is the hidden weapon of many a motion capture artist. Filmbox provides a robust and focused toolset for editing and creating motion, especially motion capture. Kaydara has chosen, rather wisely, not to dilute their product with a halfhearted attempt to match the full toolsets offered by such products as 3DS Max or Lightwave. However, the tools for handling motion are without peer in any of its off-the-shelf competitors.

Filmbox is, quite simply, a rather complex program, especially for one so focused. Its interface and terminology can be rather intimidating, and it could take an artist quite a while to become accustomed to its working environment and proficient in its tools. To be fair, Kaydara repeatedly cautions the user all throughout the tutorials that only with time and experience will the skills emerge to truly take advantage of all that Filmbox offers.

The interface itself is a variation on what has become the standard structure for 3D applications, with tool palettes down either side of the workspace, which lives in the center of the screen. Clicking on any tool refreshes the workspace with the controls for that tool. One feature I was very happy to find was transformation manipulators. For those of you not familiar with the concept, transform manipulators give a visual tool for moving, rotating, and scaling objects. Somehow, in an amazing confluence of agreement, it seems that as manufacturers add this feature, they keep the interface of red, green, and blue arrows, circles, or squares for the XYZ axes of the move, rotate, and scale tools. While this may seem to be a minor thing to get excited about, the fact that the basis of Filmbox's functionality



The Filmbox Character tool allows for the retargeting, editing, and real-time filtering of motion data for the animation of 3D characters.

requires it to work smoothly with other software means that adopting this interface eases the strain on the artists of having to work across multiple programs.

Bringing in motion capture from a magnetic or optical system is rather similar, regardless of source. Filmbox supports a breadth of motion capture formats, from the ubiquitous Acclaim .ASF format to Biovision .BVH to Motion Analysis to Vicon .C3D, among others. Once imported into Filmbox, motion needs to be fitted to an Actor. The concept of Actors and Characters is quite important to the functionality of Filmbox. It allows you to use multiple sources of motion for a single character or one motion for multiple characters. When you create an Actor you get a generic biped figure, which you scale and translate to fit the sensors from your capture session, and create comparable markers in the scene to link the Actor figure to

your data. This is where a large amount of documentation on the user's part is necessary at the time of the shoot. Once the placement is complete and associations between your placed markers and your captured markers are made, you can spend time tweaking the position of the body parts making up your Actor. Time spent here is a valuable investment, resulting in cleaner data. The process of marking up an Actor is somewhat nonintuitive and definitely will take some practice.

Now you have to apply the motion from the Actor to the Character. Characters usually come in from other packages, such as 3DS Max, Maya, or Lightwave, through the use of plug-ins. Filmbox's functionality works best with Characters oriented in the so-called da Vinci pose, which is somewhat unusual. While the da Vinci pose is convenient in a mathematical sense, most modern modeling and skeletal setup techniques use a less rigid, more natural pose. Nonetheless, the pose is not a stringent requirement, and you can reposition and reorient

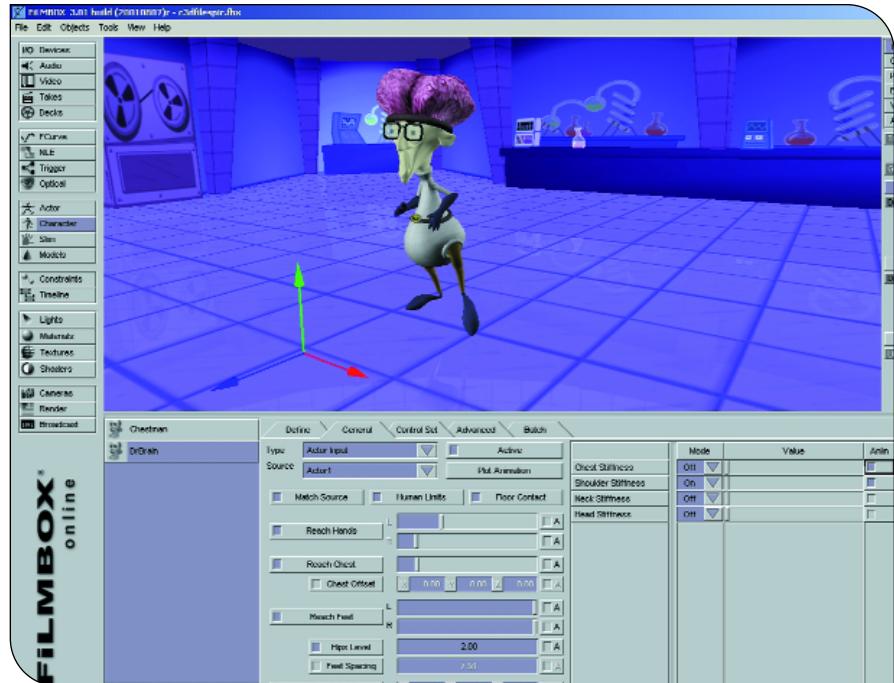
**DAVID STRIPINIS** | David is director of animation at Factor 5. Contact him at [david@factor5.com](mailto:david@factor5.com).



the limbs of your imported character from within Filmbox.

Once your data is in, editing it is pretty much a dream come true. Whether you are creating raw motion through keyframing or editing motion capture, Kaydara provides amazing tools right out of the box. One of the absolutely mind-blowing aspects of working with data in Filmbox is the concept of a floor — a simple idea brilliantly executed. Once you define a floor level, feet refuse to pass below that point. While it is merely an implementation of inverse kinematics, its simplicity of setup and use is a testimony to Filmbox's focused functionality. Creating what is most analogous to character setups in other programs, Filmbox has what are called Control Sets. Control Sets allow the user to create animation from scratch, using a variety of IK and FK tools. The same tools can be used to modify, tweak, and generally mess with captured data. Interaction was fast, and for the most part the quality of altered motion is dependent on the animation skills of the artist doing the alterations, as is to be expected.

One of the great ironies of motion capture in games is that the main requirement of game animation, motion that cleanly cycles and blends together, is contrary to the very nature of motion capture. No live performer has the precision to hit the exact marks to create a perfect, seamless cycle. Luckily, Filmbox offers a very viable solution in their nonlinear editing (NLE) system. NLE allows an artist to treat motion, or pieces of motion, as clips, which can be blended together in many ways. An artist can simply chain multiple shots together to create one long animation. Or, they can combine different parts of different animations to create new assets, such as the legs of a walking animation and the torso of a shooting animation to create a walking and shooting character. Filmbox's NLE also offers a ghosting feature that allows you to see multiple clips at the same time. The concept of Stabilizing Objects allows you to pick a piece of your character that is aligned among multiple clips to create a seamless blend. Using pose-based blending, an artist can create clips, which not only seamlessly loop within themselves, but also cleanly flow into each other. This functionality is, of course, what every game animator is after.



With a static floor and clear XYZ orientations, applying animation data to a Character is straightforward, but only for experienced animators.

Out of the NLE, you can export different assets of a character out to separate files to be loaded into the trigger tool. The trigger tool is a feature unlike any I've seen outside of dedicated real-time puppeteering software or games themselves. It allows users to link separate assets to an input device, such as the keys of a keyboard, and then interact with the character. It's a basic control system just like those you would find on any PC-based title. It is an amazing advantage for an artist to be able to see and adjust all assets on the desktop, directly in the application, rather than having to constantly re-export to the game's format and check it inside the game. Obviously that step is ultimately necessary, but this feature makes it required much less frequently.

Overall, Filmbox is a pretty fantastic product for what it does. It tries to be the best motion editor available today and to ease the interaction between artist and motion capture data, and for the most part it succeeds. Its seamless interaction with most other software provides motion capture artists no excuses for not having this gem in their toolset. ☺

## FILMBOX 3.0 ★★★★

### STATS

KAYDARA INC.

4428 Blvd. St-Laurent #300  
Montreal, Quebec, Canada  
H2W 1C5  
(514) 842-8446  
[www.kaydara.com](http://www.kaydara.com)

### PRICE

\$5,000

### SYSTEM REQUIREMENTS

Pentium processor running Windows NT 4.0 (SP 6), Windows 2000, or Red Hat Linux 6.1 or higher or MIPS R5000 processor running SGI IRIX 6.5.9 M; 128MB RAM (192MB recommended); 300MB disk space; OpenGL graphics card (8MB RAM min.)

### PROS

1. Powerful editing tools.
2. Game-specific features, such as the trigger tool.
3. Good interaction with other software.

### CONS

1. Interface is sometimes confusing.
2. Tutorials, while present, are not extremely in-depth.
3. Technical as well as artistic proficiency is required to access the full power of Filmbox.

# Rich Vogel: Producing Worlds

**R**ich Vogel is an elder statesman in the relatively young world of massively multiplayer online gaming. He produced MERIDIAN 59, the first MMOPRG to feature a flat-monthly-fee subscription model, which launched way back in 1996. He then moved to Origin where he was producer for ULTIMA ONLINE, then to Verant Interactive (now Sony Online Entertainment), where he is currently the executive producer of the upcoming STAR WARS GALAXIES. We recently caught up with Rich to talk about the state of online gaming and what it takes to produce a world for hundreds of thousands of people to play in.

**Game Developer.** What are some of the challenges unique to producing for massively multiplayer online games?

**Rich Vogel.** These games are very complicated to make. They are twice as hard to make as any single-player game because they require more content than any single-player game could imagine. You don't keep people for 30 to 60 hours of gameplay as in single-player games, you have to keep people for six months of gameplay.

Also, you're managing large teams. You have 40-plus people developing these games, because they have both a client and a server, and that takes different disciplines too. Normally in the gaming business you have people who just work on displaying game code. Now we have people who have to work on the Linux server side of things, those people are not normally in the game business. They are professionals that come out of the business sector, so it takes a melding of a large team with very different diverse backgrounds to make these games. It's a huge task.

**GD.** What should producers do to earn and maintain the cooperation and support of all the disparate elements of the development team, especially during those times when things get difficult?

**RV.** Good project management is a necessity. The biggest difference between a single-player game and an online game is that we have to survive and live on for five-plus years — that's what these games last. So we have to do things differently. We have to plan ahead, design systems well, spec things out. It takes a lot more discipline, to make sure that all the pieces are working well together and communicating.

We have an organizational structure on our team that's a lot different from most games. We have more management. We have a server lead, and a client lead reporting to a lead engineer. We have a game system lead, a content generation lead reporting to a lead designer. We have an art director that has a world lead and a creature/player-character lead.

The producer works with an associate producer and we schedule everything out.

**GD.** If something starts slipping unexpectedly, that's addressed quickly and decisively?

**RV.** Yes. We have priorities. Everything is prioritized in our games. We have a vision statement which lists what our game wants to be. Then we list all the priority one tasks, without



Sony Online executive producer Rich Vogel with an old friend.

which we cannot ship. Priority two tasks are things that would be nice to have. Priority three's are cuttables — we have to do that all the time. You can't ship a game of this type that is incomplete, and we've seen the disasters of that.

**GD.** Apparently you can ship a game like that.

**RV.** It's a huge mistake, and I hope people learn from that. Anybody that makes a bad product is going to face the consequences in an online game. Feedback is immediate. You only get one launch, and if it's poorly done, that's it. You've lost your customers' confidence. Companies have to understand this is a service, not a product. What you're going after is subscriptions. If you turn crap out then you're not going to get the subscription revenue that you're expecting or the amount of people playing your game. That is the problem that's happened lately, they pushed the product out too early.

**GD.** Do online games that have bad launches create a negative feedback loop among consumers toward online gaming that could harm the market as a whole?

**RV.** No, this market still has a lot of growth. UO has been growing ever since it launched and so has EVERQUEST. They haven't stopped. There is a big potential out there, you just have to be careful about when you launch a product, and I don't think it has anything to do with negative publicity at all when that happens. They look at it and say, "Oh, that company doesn't know what they're doing." Now, if that company comes out with another product, they're probably less likely to have people join up on it. It's all about the service, and people's perceptions. Perception is reality in our business.

**GD.** How would you sum up your role on GALAXIES?

**RV.** I help the team towards our goal by providing what each individual might need to be successful, solving problems and conflicts and removing bureaucratic hurdles from our path. I'm succeeding at my job when our team is working towards bringing all these moving parts together into a single vision. ☺

# Mipmapping, Part 1

Welcome to the first installment of “The Inner Product,” the successor to the column “Graphic Content.” “Graphic Content” was first written by Brian Hook in 1997, and the torch was carried by Jeff Lander from 1998 through to the present day.

The new name indicates a change in theme; graphics are important, but we don’t want to neglect other areas. How often do you see games ruined by bad AI or faulty network code?

As with “Graphic Content,” each “Inner Product” will be highly technical in nature and come with full source code. My goal is to make this column as useful as possible to experienced and expert game developers. My content guideline is this: if it wouldn’t have been new and useful information to me two months ago, then I won’t write it.

The inner product, also known as the “dot product,” is a mathematical operation on vectors. It’s one of the simplest and most useful pieces of 3D math; I chose the name to underscore the importance of mathematics in building game engines. Additionally, the phrase “the inner product” refers to the game engine itself. The people who play your game see a lot of obvious things created by texture artists, 3D modelers, and level designers. But the inner part of the product, the engine, makes it possible for all that art stuff to come together and represent a coherent game world. As such, the engine is all-important.

Though eventually we will cover subjects besides graphics, we’re going to start by looking at the process of mipmapping. Our goal is to achieve a sharper display of the entire scene, and perhaps to improve color fidelity, too.

## What Is Mipmapping All About?

When mipmapping, we build scaled-down versions of our texture maps; when rendering portions of the scene where low texture detail is needed, we use

the smaller textures. Mipmapping can save memory and rendering time, but the motivating idea behind the technique’s initial formulation is to increase the quality of a scene by reducing aliasing. Aliasing happens because we’re coloring each pixel based on the surface that, when projected to the view plane, contains the pixel center. Small visual details can fit between pixel centers, so that as the viewpoint moves around, they appear and disappear. This causes the graininess that makes some Playstation 2 games look icky.

Fortunately for us, a lot of smart people have been thinking about aliasing for a long time; all we have to do is stand on their shoulders.

Way back in 1805, Karl Friedrich Gauss invented the Fast Fourier Transform, a way of decomposing any sampled function into a group of sine waves. The Fourier Transform is more than a mystic voodoo recipe for number crunching; it often presents us with a nice framework for thinking about problems. When manipulating an image, sometimes it’s easier to visualize an operator’s effect on simple sine waves than on arbitrary shapes.

So how does the Fourier Transform help us here? To eliminate aliasing, we need to throw away all the sine waves with narrow wavelengths — which represent the only things that can fit between pixels — and keep the waves with broad peaks. Such a task is performed by a digital filter. For a detailed introduction to filtering and other signal processing tasks, see the books by Steiglitz and Hamming listed in the For More Information section.

Once we understand that generating mipmaps is just the task of building smaller textures while eliminating aliasing, we can draw on the vast signal processing knowledge built up by those who came

before us. So much work has been done already that once we have the basic concepts, our code almost writes itself.

## The Usual Approach to Mipmapping

Historically, game programmers haven’t thought very hard about mipmap generation. We want to make a series of textures that decrease in size by powers of two. So we tend to take our input texture and average the colors of each four-pixel block to yield the color of one pixel in the output texture. Listing 1 depicts this technique, which we’ll call “pixel averaging” or “box filtering.”

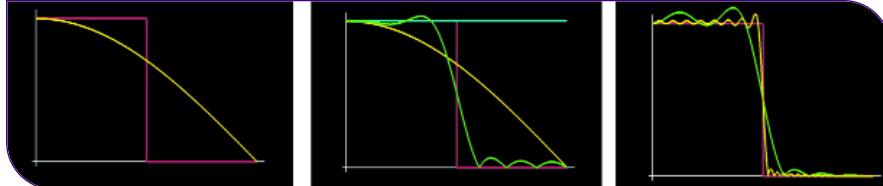
This function looks like it works — when we try it, we get textures that are smaller versions of what we started with. We often figure that that’s the end of the story and move on to think about other things. The fact is that this technique generates images that are blurry and a little bit confused. We can do better.

We can view Listing 1 as a digital filter operating on our image data; then we can investigate what the filter does to all the sine waves that compose our texture map. Figure 1 contains a graph of the frequency response of the pixel-averaging filter; the X-axis represents the frequencies of waves in our input image, and the Y-axis shows the magnitude by which they’re multiplied to get the output.

To perfectly eliminate aliasing, we want a filter that matches the brown line in Figure 1 — zero amounts of high frequencies, full amounts of low frequencies. In signal-processing terms, we want an ideal low-pass filter. The graph shows that the pixel-averaging filter doesn’t come very close to this. It eliminates a big chunk of the stuff we want to keep, which makes



**JONATHAN BLOW** | Jon is a game technology consultant living in San Francisco. His e-mail address is [jon@bolt-action.com](mailto:jon@bolt-action.com). Music that influenced this article includes *Sleepytime Gorilla Museum*, “Grand Opening and Closing”; *Nick Cave and the Bad Seeds*, “No More Shall We Part”; and *Jarboe*, “Sacrificial Cake.”



**FIGURE 1 (left).** Frequency response of the box filter versus the ideal. The brown line represents the ideal frequency response; yellow represents the box filter. **FIGURE 2 (center).** The filters discussed in this article. The brown line represents the ideal; cyan represents the point filter; yellow represents the box filter; green represents Lanczos- and Kaiser-windowed sinc pulses (the graphs nearly coincide). **FIGURE 3 (right).** Kaiser filters at different filter widths. The brown line represents the ideal; green represents 16 samples; yellow represents 64 samples.

the output blurry; it keeps a lot of stuff we'd rather get rid of, which causes aliasing. This is important: switching to a smaller mipmap for rendering will not eliminate as much aliasing as it could, because we accidentally baked some aliasing into the smaller texture.

## How to Build a Better Filter

It's well established what we must do to achieve an ideal low pass on our input texture. We need to use a filter consisting of the sinc function, where  $\text{sinc}(x) = \sin(\pi x)/(\pi x)$ ,  $x$  indicating the offset of each pixel from the center of the filter. (At the center,  $x = 0$ , recall the Fun Calculus Fact that  $\lim_{x \rightarrow 0} \sin(x)/x = 1$ .) The problem with sinc is that it's infinitely wide; you can go as far as you'd like along positive or negative  $x$ , and sinc will just keep bobbing up and down, with an amplitude that decreases slowly as you zoom toward infinity. Filtering a texture map with sinc would require an infinite amount of CPU time, which is a bummer.

We can compromise. First we decide how much CPU we want to spend building mipmaps; this roughly determines the width of the filter we can use. Then we construct an approximation to sinc, up to that width, and use the approximation to filter our images. We could create our approximation by just chopping off the sinc function once it reaches our maximum width, but this introduces a discontinuity that does bad things to the output. Instead, we multiply sinc by a windowing function; the job of the windowing function is to ease the sinc pulse down to 0, so that when we chop off the ends, badness is minimized.

For a very accessible description of why sinc is the appropriate filter, and how windowing functions work, see the book *Jim Blinn's Corner: Dirty Pixels* in For More Information.

So, to design our mipmap creation filter, we just need to choose a filter width (in texture map pixels) and a windowing function. Windowing functions tend to look similar

when you graph them, but the differences matter. Filters are funny that way: they're just arrays of real numbers, but just small tweaks to those numbers can significantly change the output. (If you start with a good filter and tweak the coefficients arbitrarily, know that the result is most likely bad.)

Each windowing function represents a different compromise between blurring and aliasing. We can predict what the window will do by graphing the frequency response of the resulting filter, as we did in Figure 1, or we can write some experimentation code that builds mipmaps with arbitrary filters and look at the results. This month's sample code does just that.

## Filters We've Tried

We tried out several filters. The first is the point filter, which is what you get when you just pick one value from every four pixels and use that in the lower-

**LISTING 1.** A common method of building mipmaps, known as "pixel averaging" or "box filtering."

```
void build_lame_mipmap(Mipmap *source, Mipmap *dest) {
    assert(dest->width == source->width / 2);
    assert(dest->height == source->height / 2);

    int i, j;
    for (j = 0; j < dest->height; j++) {
        for (i = 0; i < dest->width; i++) {
            int dest_red, dest_green, dest_blue;

            // Average the colors of 4 adjacent pixels in the source texture.

            dest_red = (source->red[i*2][j*2] + source->red[i*2][j*2+1]
                       + source->red[i*2+1][j*2] + source->red[i*2+1][j*2+1]) / 4;
            dest_green = (source->green[i*2][j*2] + source->green[i*2][j*2+1]
                          + source->green[i*2+1][j*2] + source->green[i*2+1][j*2+1]) / 4;
            dest_blue = (source->blue[i*2][j*2] + source->blue[i*2][j*2+1]
                          + source->blue[i*2+1][j*2] + source->blue[i*2+1][j*2+1]) / 4;

            // Store those colors in the destination texture.

            dest->red[i][j] = dest_red;
            dest->green[i][j] = dest_green;
            dest->blue[i][j] = dest_blue;
        }
    }
}
```



**FIGURE 4.** Ringing induced by different filter widths. 14 samples for the image in the left; 64 samples for the image in the right.

level mipmap. (For textures displayed at a scale of one texture pixel per screen pixel, it's also equivalent to rendering without mipmapping.)

The second filter is pixel averaging, the algorithm given in Listing 1.

Third, we gave the Lanczos-windowed sinc function a try, since Jim Blinn says good things about it in his book, and it is popular among graphics programmers.

Fourth, we look at the Kaiser window. Don Mitchell recently did some experiments revealing that the Kaiser window, with alpha parameter 4, is noticeably better than Lanczos for graphics purposes. Don's been around enough to have a filter named after him, so it seemed prudent to pursue this.

Figure 2 shows the frequency responses of all these filters. The coefficients for the Kaiser filter are only subtly different from the Lanczos filter, so their frequency responses are almost the same. Eyeing images on the screen that were mipmapped by each, I can't see the difference. Mipmapping is not a very strenuous application of image filtering, though; Don's tests were more hardcore. I figure that, so long as the computational expense is the same, we should be in the habit of using the slightly better filter, so it's all warmed up for more difficult future problems.

## Avoiding Ripples

You can see from Figure 2 that there are some ripples in the frequency responses of the higher-quality filters. Signal-processing math says that we can increase the quality of these filters just by making them wider. When we do this, we get a curve that more closely approximates the ideal low-pass filter; but it has more ripples, and each peak and trough is concentrated on a tighter group of frequencies. Figure 3 contains a graph of the Kaiser filters for sizes of 16 and 64 samples.

In the world of audio processing, it's common to ignore ripples of this magnitude, but our eyes are pickier than our ears. As those ripples become concentrated around tighter (more coherent) frequency

groups, they create more coherent visual artifacts in the image. Pick a wide enough filter, apply it to a texture map that has large areas of constant color, and you've got artifacts ("ringing") all over the place.

I was only able to crank the width of the Kaiser filter up to 14 samples before I started seeing prominent ringing in some test images. To show what happens beyond that, I ran a wide filter on the texture of a road sign. Figure 4 compares the ringing induced for filters at widths of 14 and 64 samples.

One might think that if you were willing to spend a lot of CPU time, you could use a filter so huge — that is, so close to approximating the ideal low-pass response — that you wouldn't see the ripples. I tried this for filters up to 1,000 samples wide, and the amount of ringing never seems to decrease.

## Preventing the Propagation of Distortion

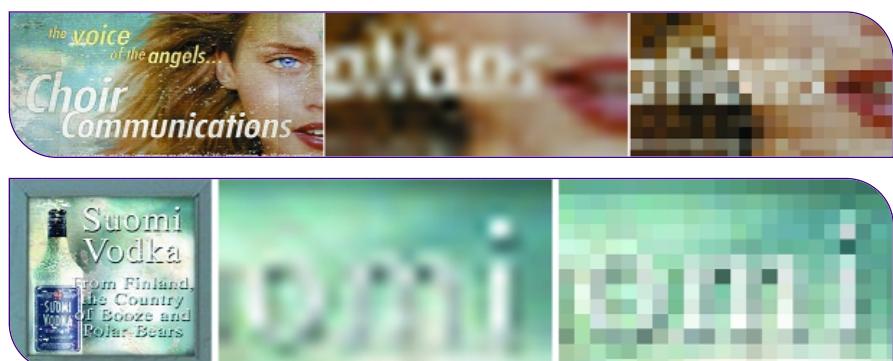
So we're restricted to relatively narrow filters; their frequency responses will

be better than the box filter, but there will be some distortion that we can't avoid. Fortunately, we can reduce that distortion's influence on our images.

When we sit down to create a mipmap generator, we often think up the following procedure: Take an input image of some width, perhaps 256 pixels, and pump it through a mipmapping function that returns a texture 128 pixels wide. Then take that result, push it through the same function, and get back a texture 64 pixels wide. Repeat until it's gotten as small as you need.

This technique causes problems because the output of each filtering step, which contains distortion due to the imperfect filter, is used as input to the next stage. Thus the second stage gets doubly distorted, the third stage triply distorted, and so on.

We can fix this by starting fresh from the highest-detail image when generating each mipmap level. We still want an ideal low-pass filter, but we want to vary the filter's cutoff frequency. (For the first mipmap level, we want to throw out 1/2 of the frequencies; for the second, we want to discard 3/4; for the third level, 7/8; and so on.) To accomplish this, we progressively double the width of our filter and tweak the parameters so the sinc pulse and the window become wider too. (It is O.K. to make the filter wider and introduce more ripples, because the size ratio between the source and destination images gets larger.) This procedure provided the highest-quality results.



**FIGURE 5A (top left).** A texture from MAX PAYNE. **FIGURE 5B (top center).** Results of box-filtered mipmapping, zoomed in on the lower right corner. **FIGURE 5C (top right).** Results of Kaiser-filtered mipmapping. **FIGURE 6A (bottom left).** Another texture from MAX PAYNE. **FIGURE 6B (bottom center).** Results of box-filtered mipmapping, zoomed in on the upper right corner. **FIGURE 6C (bottom right).** Results of Kaiser-filtered mipmapping.

## The Sample Code

This month's sample code demonstrates these filters acting on a bunch of images. The code is built for versatility and readability, not speed; it's meant to serve as a code base for you to experiment with. It uses only simple OpenGL calls, so it should work for most people; it's been tested on a few GeForce and Radeon cards.

To emphasize the difference between box filtering and Kaiser filtering, there's also a "fill screen mode" that you can toggle by pressing the F key. This tiles the entire screen with the current mipmap; pressing the spacebar switches filters. You can see the effect of the whole screen getting slightly sharper.

## The Results

In general, the output images from the Kaiser filter look better. Some examples: Figure 5 shows a billboard from Remedy Entertainment's MAX PAYNE. In the pixel-averaged version, three mipmap levels down, the copyright notice along the bottom of the billboard has disappeared, and the woman's mouth is something of a blur. In the Kaiser-filtered version, the copyright notice is still visible and the woman's teeth are still easily distinguishable.

Figure 6 shows another billboard. In the box-filtered version, the text is more smudged and indistinct, and the bottle label is less clean-cut.

From the sample application you can also see that once the images get small enough, the choice of filter doesn't matter much; you just get a mess of pixels either way.

## Next Month

Here are some mipmapping quality issues that don't actually have to do with filter choice: I'll confront those next month, and see if there's anything we can do to eliminate this problem of wide filters causing our output images to ring. I'll also question the appropriateness of the Fourier Transform. ☺

## ACKNOWLEDGEMENTS

Thanks to Remedy Entertainment for allowing us to use images from MAX PAYNE in this article and the sample code.

The "Phoenix 1 Mile" texture in Figure 4 is by Dethtex, see <http://users.nac.net/schwenz>.

Thanks to Don Mitchell, Sean Barrett, Chris Hecker, and Jay Stelly.

## FOR MORE INFORMATION

- Watt, Alan and Mark Watt. *Advanced Animation and Rendering Techniques: Theory and Practice*. Addison-Wesley, 1992.
- Blinn, Jim. *Jim Blinn's Corner: Dirty Pixels*. Morgan Kaufmann, 1998.
- Hamming, R.W. *Digital Filters*. Dover Publications, 1998.
- Steiglitz, Ken. *A DSP Primer: With Applications to Digital Audio and Computer Music*. Addison-Wesley, 1998.



**B**eing the art lead or art director in the computer gaming industry can be a glorious and high-profile position with many opportunities for creative expression. Today's computer and console games have become an overwhelmingly prolific and advanced visual media that captures the attention of huge player audiences of every age in nearly every country. These prolonged game experiences, in a feast of visual pleasure, make up-and-coming computer artists yearn for total control in defining and creating the rich images and 3D content in a game. Consequently, highly motivated and ambitious artists may accept a rare opportunity to become responsible for leading art production efforts by a team of game artists through a game's development cycle. These individuals may or may not have had any art direction, art asset organization, or art resource management experience.

If you have the right mix of creative and technical art-related talent, play computer and console games, and are more than just casually acquainted with the tools commonly used by most game developers, then maybe you too have recently become the designated point person for art content creation on a project. Maybe you can credit your new lead status to having been at the right place at the right time, or knowing the right people. Success is sometimes as much a matter of luck as a matter of ability. Well, for better or worse, congratulations. In an industry that bases so much value on what title you worked on last, my advice to you is to take great care to ensure as much success for your current project and

# The Role of the Art Lead

those who work with you as you would with your own professional career.

Now that you've been granted the privilege to take charge, you should ask yourself some important questions. Do you have what it takes to lead a team of skilled artists through months (or years) of creative and technical production work and complete all art assets on time? Do you have the ability to carry out daily the broad range of responsibilities you have to your development team, to the product, and to the creative individuals that report to you? Are you committed, disciplined, and yet flexible enough to accommodate the needs of your evolving product, the publisher, and the many others involved that contribute their unique ideas to your game? This privilege comes with a price. Are you ready to pay it?

This month's column is much more about managing art talent resources — employees — than it is about managing the visual content these creative and skilled individuals produce for your game. For anyone who hasn't already figured out what this role entails, I want to help bleach out some of the uncertain gray areas of what this job has grown into within our industry. Having had the good fortune of working my way up the ladder from an entry-level artist position 10 years ago, I have been directly exposed to both the good and the bad aspects of being an art lead and an art director. Hopefully the "dos" and "don'ts" that I learned the hard way can save you heartaches and headaches.

By nature, the art lead or art director depends on the support and help of many colleagues. I've gained much insight on the job from other artists, art directors, producers, programmers, game testers, designers, and sales folks who are in the business

of making and selling successful games beyond a single published title.

## Job Title vs. Job Description

**N**ot all game companies share the same art team structure. Art team job titles vary considerably with many job responsibilities overlapping horizontally and vertically in a team hierarchy. Our industry's job titles are borrowed primarily from the print advertising and film industries and so are quite loosely used and defined. Let's assume for now that all game development teams have a leading artist called an art lead, an art director, a technical or creative art director, a senior artist, or even an art manager. In larger companies, art leads may report to art directors who then report to art department managers. Where I currently work, the development teams are so small and streamlined that art leads report directly to studio producers in most cases. For this reason it is important to keep in mind that an art lead at company X isn't the same job title and set of responsibilities as it is at company Y.

For now, I'd like to narrow my focus to the person specifically responsible for managing the creation of art assets on a project, defining the art paths for producing and implementing such game content, and delegating art tasks to several art resources. I'll refer to this function from here forward as the art lead (AL).

The art lead for a game development team has many responsibilities and obligations common to games of any genre, platform, or company. A successful and seasoned AL is a fairly well rounded creative person that is usually proficient with drawing and composition, color and lighting theories, 3D modeling, design, anima-



**TITO PAGÁN |** *Tito is a seasoned 3D artist/ animator working at WildTangent and teaching at DigiPen in Seattle. His e-mail address is [tpagan@w-link.net](mailto:tpagan@w-link.net), and his web site is [www.titopagan.com](http://www.titopagan.com).*

tion timing, and texture map creation and application. This person is also technically inclined and is often called upon to resolve issues like art content optimization, production automation, and defining and streamlining art paths. The outstanding AL is a manager of people resources that execute orders and deliver the best game art possible from the team. The AL's skills and knowledge go far beyond the realm of art and technology to include managerial and organizational skills, communication skills (both written and verbal), and business/marketing and media knowledge. Today's successful AL is something of a renaissance person.

## Lead If You Must, But Learn How

**A**s the job title implies, it's the art lead's job to lead or direct others. This may be a staff of several CG artists or just one concept artist, modeler, or contract artist at the beginning of a project. The AL guides artists to achieve department objectives. Some people are born with all the traits of a leader. Others have to develop leadership skills.

Leadership is not about making a quick decision, but making the right one at the right time. The ability to do so consistently comes with experience. Becoming a good leader requires having individuals who want to follow your lead. This, in turn, comes from having gained the trust of other professionals by demonstrating the ability to make good decisions and treating others with respect. This level of trust is almost always earned on the job and never assumed because of credits or claims the lead has made of past work at some other company. Once this trust is gained, however, it can be lost again quickly should it be taken for granted, not reaffirmed periodically, or simply abused.

## Staying on Top and Reaching Higher

**F**ollowing are 11 suggestions for first-time art leads who want to improve ongoing relations with members of a staff and continue to move up the food chain. Keep in mind of that there is always

someone more ambitious and determined than you who will take these to heart if you don't.

**1. Educate yourself as a manager and an artist.** To the best of my knowledge, there is no formal program available today that teaches a computer artist to become the art lead for a game — no college, art institution, or even company-sponsored training program. On occasion you may find a seminar or special training class regarding a related topic at an industry conference. If you have to, invest in yourself and attend. Otherwise, all art leads today have to learn through a process of trial and error, on the job, and at the expense of a project and all others involved on that project. If you take your career seriously, I suggest you take evening college courses or read books on management principles. And if you find that your skills in other areas are lacking instead, such as drawing, knowledge of color, or basic animation techniques, find classes or tutorials that will round out your abilities.

A good art lead will make an ongoing effort to stay on top of the latest production techniques and available tools that help speed up art production time. Sharing information with other artists on the team is a great way for the AL to gain the trust that makes people want to work and share knowledge with them and others in return. It helps people feel like the lead has their staff's best interest in mind as well. Of course, this can have a reverse affect if what the AL is sharing involves new high-end tools that can't be purchased for other artists because of a tight budget.

**2. Know your game.** Know your competition and customers, too. Different games require different kinds of art and different ways of creating and implementing them. The expectation of quality, quantity, and complexity of art assets will vary between genres and target audiences. Avid gamers have an advantage because they understand many of these issues more intuitively. So do the required research, even it means you have to take five and play another darn game.

**3. Treat people with respect and show consideration.** This is one of the most basic principles in human relations. Far too often I've witnessed people abuse their power in a leadership role. Public reprimands, for

example, do nothing constructive in any situation. In fact, it is the fastest way to lose others' respect and trust. It is usually intended to belittle and stir fear in others by demonstrating power over them. Avoid doing so at all costs. If you have to discipline an artist, do so in private.

**4. Interact with other members on the team from artists to programmers to game testers.** Get to know what they're like, what their strengths are, and what ideas they may have to offer. Don't work in a vacuum. Let them share their vision of what the final product will be and incorporate what you can to realize some of these ideas. Let the game have parts and pieces owned by others; a game should be the end result of all involved in making it. This practice usually goes a long way in fostering a more cohesive and immersive game for your players to enjoy. It would be fairly pretentious of any art lead, designer, or level designer to think that he or she is the only one with great ideas all of the time on a project (unless of course they are the only person developing a game, in which case they better have great ideas). More people are willing to put in time above and beyond the call of duty if they are given opportunities to contribute, learn, and grow on the job. To put it another way, sharing creative opportunities is a small price to pay for days and days of free man-hours on your project.

**5. Communicate.** The word alone sounds like a cliché. Perhaps this is because so many people fail miserably at this crucial aspect of leadership. A successful AL communicates ideas effectively and thoroughly while allowing opportunities for some creative input by other members on the team. Game development is a team effort involving many other creative individuals with specific skills in storyboarding, 3D modeling, texturing, animation, level design, programming, game design, and business management. Learn to appreciate and rely on those other people sharing their opinions, knowledge, and cooperation. The result is more likely to be a successful launch of a game that is engaging and cohesive.

Because an AL has to manage others, avoiding them is never an option, no matter how annoying they may be. Be patient with other artists. Newly formed teams take time to gel as everyone becomes

acquainted with each other's abilities, strengths, and weaknesses, and members settle into their established positions.

If you are the new guy or gal on an existing team, things can be quite challenging at first. Let's face it, there are insecure people in our industry, just like in any other, that fear losing their jobs or status to some newcomer. You may be confronted with a challenge to prove your worth before you can gain their respect. Take the time to foster and nurture that relationship no matter how difficult the other person makes it for you. I have seen many people be won over with kindness and direct attention. Here is where patience, maturity, and experience really pay off in the long run.

**6. Preplan your work.** Use the tools at your disposal to get your thoughts out of your head and on a visual medium as soon as possible. Everyone can relate to an image. Despite popular beliefs that others can read minds, I have yet to meet someone who can. Examples of ways to facilitate better planning are storyboarding a cinematic sequence, or drawing a concept of a monument for a level design before actual building the model. Designing on paper first is much cheaper and faster than designing as you go while modeling objects on a computer. Once the preliminary designs are done, others can more easily develop a task list of required art assets from them.

I have found that half of the average artist's time is spent actually designing and creating the art, and the other half is devoted just to getting it in the game, looking good, and interacting correctly with other game assets and the player. This ratio varies, of course, depending on the level of proficiency an artist has with the tools he or she uses. Allow enough time for integration.

**7. Help others help you by being organized and structured.** The many individual pieces of art content you and your team have to create and manage throughout a development cycle can be overwhelming. Set up a well-thought-out directory structure and naming convention for all of the art assets you are planning to have your team create. Being organized early on will help you communicate and delegate smaller chunks of the work much more easily.

**8. Give credit where it is due.** Seriously. And do it often, not just at the end of a project in the game's credit list or only during an employee's review. A fair amount of acknowledgment along the way for a job well done will do wonders for that person's motivation, and consequently, the game. All artists want to know that their work makes a difference in the outcome of the game. It promotes responsibility and accountability to every other team member and to the product.

And remember this: Even if you weren't the one to create all or even most of the cool features that are the focal areas in a game or specific level, know that in the end, you will still get the credit you deserve for having established a standard of quality and having managed and directed a creative force to completion. Take comfort in that and feel proud of your many contributions, no matter how spread out and tucked away they may seem. Granting an opportunity for artists to earn that credit doing something they enjoy, instead of keeping it for yourself, is another way to keep artists focused on the job and devoted to you as a lead.

**9. Accept criticism.** This is a very important and often hard thing for many artists to do. As the AL, you bear the brunt of all incoming art-related criticism daily. Art is a very subjective thing and is at the mercy of everyone involved, regardless of whether they are part of the formal approval process. Have confidence in yourself. Your ability got you there in the first place. But learn to compromise your ideas when necessary in an effort to complete the work on time. One thing that helps get me through a public critiquing more quickly is to present more than one option whenever possible. Giving others a couple of ideas to pick from gets them to commit to one or the other, instead of sending you away repeatedly with a response like, "I'm not sure what I want but I'll know it when I see it." Also, accepting criticism well makes you better at critiquing others, much like how waiters are better tippers at a restaurant. They know what it's like to be on the receiving end.

**10. Show support in all areas of your team.** Help foster a sense of teamwide camaraderie and collaboration by extending out help to others that may need it. A good

example is a programmer in need of test art early in a project. I strongly suggest you support all efforts of programmers that are working to develop internal tools that will further your progress in creating and implementing custom game art. Do this by providing them with the artwork they need regularly. If you know it's going to be placeholder art, get a less experienced artist to create these pieces instead. The practice provides another with good experience while freeing up your time to do other things, such as progress reports.

If you can't delegate this support task to another artist, consider the fact that the interaction may improve your relationship with that programmer. A goods relationship can help you later on when you try to sell the programmer on adding a new feature that can help make up for any lost production time you have invested, as a returned favor. In the meantime, your direct involvement in a tool's creation reduces your learning time later when the tool is ready and online.

**11. Get the job done.** This is more a matter of knowing when to call a job "good enough." We all take pride in a job well done and want to create the best quality art possible. However, art that is not completed will never help finish a game. A game not finished will never make your company any money.

## Is It Worth It?

Having creative control as a leader in the game industry does have a down side. Among them is the high probability of getting burned out. The many mundane administrative responsibilities, pending deadlines, and ego-driven personality issues most leads have to deal with can be mind-boggling. Being the art lead through the course of one or more projects can leave you wondering if it is even worth all the hassle. I believe it is, but the job requires that you work hard at trying to stay positive.

For many art leads, art direction is an all-encompassing and demanding undertaking. This can make for a very dull existence over time. As you gain more responsibility in this position, the tendency is to become more administrative and less creative. Before long, you realize that you've

---

traded pushing a drawing pencil, pixels, vertices, and keyframes for pushing Word documents and spreadsheets through the company's internal network and e-mail system. If this situation applies to you, you'll have to take certain personal measures to help you maintain a positive and healthy outlook and continue moving forward with your project.

If you really want to lead others, consider it an obligation you owe to yourself and to them to stay positive and optimistic at all times. I have learned from many other professionals that the cure for this, simply put, is to get a life outside of work. Make the time to pursue other interests. Get a hobby. Get a dog. Make new friends. Read books. Find some balance in your life by not just doing or focusing on one thing all the time. Such new activities will get your mind off your work and may even lend new perspectives to what you are doing on the job with your game and how you relate to other coworkers in the process.

The consequence of not finding an alternate creative outlet to combat feelings of burnout or the onset of a creative funk can be demoralizing and even destructive. Through the years I have witnessed many industry friends fall victim to this far less fulfilling and rather stale state of being for months at a time. It has eventually led to more serious problems such as poor behavior, substance abuse, and a negative outlook on their work, their company, and even their personal or family life. If and when it should get this far, sometimes professional help is the only real alternative.

One of the creative outlets that helps me feel challenged and inspired is giving myself personal art projects of my own design that address specific goals. These enable me to learn as much as I can about the tools and techniques used to create things I like in 3D art and character animation. These assignments certainly don't feel like "work" to me. Since this was my passion at the time I embarked on this career, I still fill what spare time I have with more advanced projects. Here I can make up for what fun things I don't get to do at work, while honing my skills in preferred areas. My greatest challenge so far has been making the time to work on them

and buying the hardware and software required. The end result every time, however, has been very rewarding and fulfilling, personally as well as professionally.

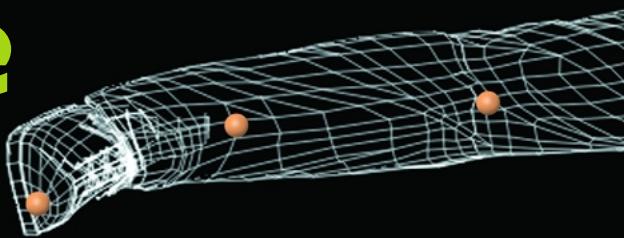
### One Step at a Time

If you're working on becoming an art lead on a game title, you'll find that you do have a tall order to fill. Of course, the

sooner you can get a handle on what is expected of you in this new position, the sooner you can take measures to ensure that you're on a good course to getting there. The process can be long, but take your time and do it right. In the end, it's the respect you'll have earned from your team members that will take your new role as a leader to new heights in achievements and personal satisfaction. ☺

---

# The Quest for Pure Motion Capture



**T**he first motion capture systems were designed for the medical industry to better analyze physical injuries and defects. By the mid-1990s, motion capture developers recognized a potential market in the entertainment industry and began to modify their systems to suit the needs of production houses and game developers.

The transition from medical tool to animation tool has been a difficult and painful struggle, and motion capture has left some ugly marks on game developers. Entire projects have been canceled due to problems related to motion capture.

Motion capture hardware and software developers as well as motion capture service providers have sometimes struggled to stay in business, and some have not survived. Many animators rejected motion capture even before attempting to make it useful. Some said that a true artist would never use motion capture or that using motion capture was cheating.

Despite the challenges, few would argue that motion capture is an essential aid in character animation. Game developers have come to rely on motion capture as a

vital tool in producing quality and efficient character animation. Today, motion capture systems are utilized in a number of industries. None, however, comes close to competing with the amount of motion capture used in game development.

The same game developers that once canceled projects due to motion capture problems now have their own motion capture studios and highly skilled technicians, or wranglers, to operate them. Animators have realized that motion capture can be a tool just like any other effect available in 3D animation packages and are learning how to work with motion capture rather than fight it.

Incredible results can be seen where motion capture has been implemented correctly. Namco's SOUL CALIBUR is a good example of how motion capture can really bring a game to life.

Still, many game developers face the challenge of acquiring quality motion capture data and implementing it correctly. We've all seen the ugliness: sliding feet, jittering spasms, snap-crackle-pop, mushiness, and weird mesh deformations. For some reason, things just don't look right. The biggest problem is that most often what you see in

the motion capture performance is not what you see in the game. Motion capture departments are constantly striving to reach the goal of WYSIWYG (what you see is what you get) motion capture.

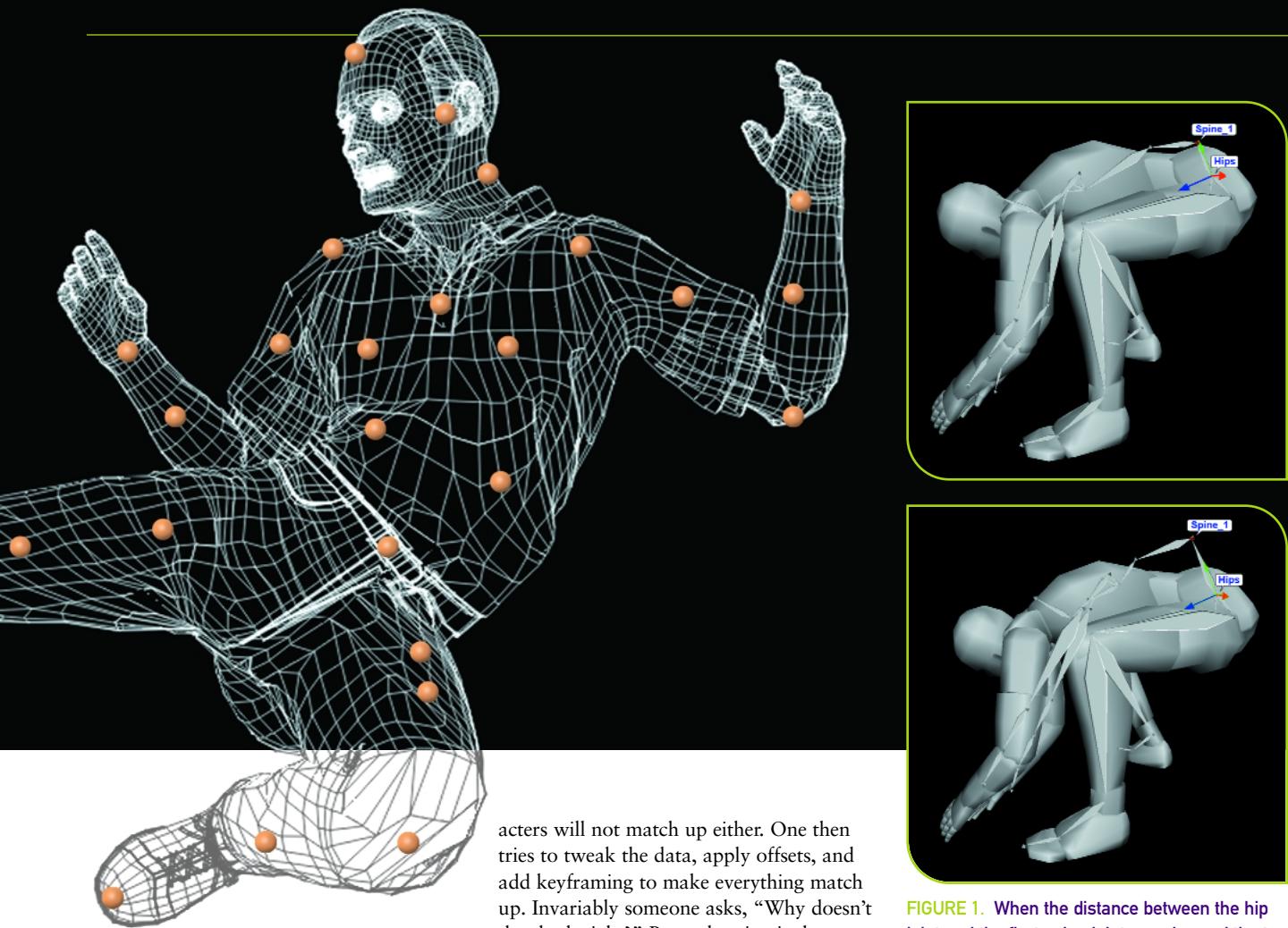
This article examines some of the obstacles in reaching this goal and how to minimize or solve them. One must consider many issues: various aspects of preproduction, performance, data quality, and software tools. Some of the information presented here is specific to retro-reflective, optical motion capture systems, but much of it can be applied to any system you choose.

## Planning

Certain aspects of preproduction are critical to WYSIWYG. This article can't cover every point in preparing for a motion capture shoot, but some key issues must be considered before acquiring and implementing motion capture data.

**Character definition.** One of the first issues to address is character definition. A complete character design includes artwork that illustrates the character's dimensions and physical traits, a description of its personality, its special abilities, and anything else that defines the character. A biography of the character can be very useful. Super-human features are often key elements of a character. A character with long gorilla arms or with no neck and long bull horns coming out of its head must be

**DAVID WASHBURN** | David began his motion capture career as a technician for Biovision Motion Capture Studios in 1994. He has done motion capture for more than 100 games, various commercials, and numerous sports analysis sessions. He has also worked as a motion capture consultant providing training, system installations, and production pipeline optimization for new studios. He is currently employed by Westwood Studios.



**FIGURE 1.** When the distance between the hip joint and the first spine joint goes beyond the top of the pelvis, the back will break as the character bends over.

precisely articulated before attempting motion capture.

Poorly defined characters are often changed significantly after the capture, and of course the original motion data won't look right. Character design is directly related to performance and casting issues that are discussed later. Without a well-defined, solidly built character, directors and actors cannot possibly hope to create believable, interesting, and appropriate motion.

**Character scaling.** Differing sizes and proportions between the live actor and the virtual character can also cause problems. This scaling issue is even more critical when props or set pieces are used.

For example, you might capture a five-foot, two-inch woman to portray a ten-year-old girl. You can always scale the motion to the mesh, right? What if the ten-year-old girl sits down at a table and puts her hands under her chin and then stands up and leans on the table with her hands? When the motion is scaled she will not interact with the floor, the chair, the table, or herself correctly, because the distance between the table and her chin will not match the distance between her elbow and her hands. Interaction with the other char-

acters will not match up either. One then tries to tweak the data, apply offsets, and add keyframing to make everything match up. Invariably someone asks, "Why doesn't that look right?" Poor planning is the answer.

When possible, casting the right person for each character will make a big difference. Another solution is recognizing the problem before capture and choreographing motions that will scale better. For instance, if the girl's motion were choreographed so she didn't put her hands under her chin and didn't lean on the table when she stood up, then simply scaling the motion would work.

**Skeleton design and solving.** Skeleton design is another piece of the puzzle. To achieve the best results from capture data, skeletons must be designed and tested before the shoot. The first rule of building a skeleton is never to rotate or scale the bones. Always use translations to position the joints in the mesh. When joints are rotated or scaled, the motion data is applied on top of those offsets, which usually causes problems such as flipping, snapping, and popping. Motion capture data always tests the limits of mesh deformation, so the number of joints and their locations are critical (see Figure 1). A skeleton with a short femur (thighbone) and an extra long tibia (shinbone) will not be able to follow the motion capture data and thus won't look like the performance

because the knee joint can never reach its correct position.

Most game skeletons don't allow bones to stretch or allow joints to move in and out of sockets like they do in real life. The hip and the shoulder ball joints move around a great deal in their sockets and can be very noticeable in certain animations. Sitting and kneeling cause the hip bone to move in its socket so much that the knee joint can no longer reach its position. Ideally, prerendered animations should be set up with dynamic bones, allowing child joints to vary in distance from the parent.

The character mesh should be designed to deform correctly with the skeleton. Vertices in the wrong places and vertices that are incorrectly weighted or bound to the wrong bones will never produce animations that look like the motion capture performance (see Figure 2).

Most of the mistakes made when configuring translational marker data to drive skeleton joint rotations are related to the difference in scale between the actor and the virtual character. The details of how to



correctly map/solve/characterize (all the same thing) are application dependent and too complex to articulate here. The goal should be a mapping solution where the joints mirror the markers. Inverse and forward kinematics and all other constraints shouldn't modify the motion but rather help solve scale issues.

**Marker set.** A marker set designed for each character obviously must be well thought out. Choosing the right marker set has a major impact on whether animations end up looking like the reference video. Decisions regarding where to put markers and how many to use should be based on your software tools and the skeleton design.

Software tools can take advantage of redundant markers to create rigid bodies for gap filling and smoothing caused by occlusion or dropout. Additional markers can be used to create asymmetry for labeling. The design must be balanced, though; having too many markers causes them to merge into each other. Inner knee markers will interfere with each other, but markers just above the knees on the thigh can be just as useful. Hands, arms, and props often occlude chest markers, but back markers may be adequate to articulate all torso movement.

You must also consider the skeleton design when designing a marker set. A skeleton with two spine joints doesn't need as many markers as a four-segmented spine. Configuring translational marker data to drive skeleton joint rotations should be optimized with the right number of markers placed on the right body locations (see Figure 3).

Using translational marker data to drive facial bones is a simple way to produce facial animation. Because the facial bones have no children, marker data can be scaled to match the bones without any complications.

Attaching markers to clothing is another reason why motion capture data doesn't look like the performance. Usually you want the motion of the body, not the motion of the clothes. For example, the act of sitting causes clothing to slide up the leg. When arms are raised above the head, clothing on the elbows slides up the arms. It's best to attach markers directly to the skin where possible. The best motion capture suit a birthday suit.

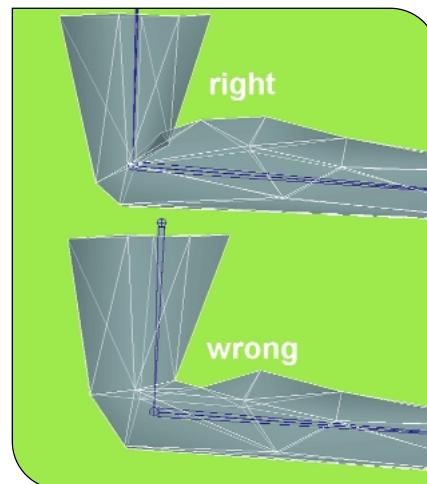
**Props and set design.** Props and set pieces need to be designed with motion capture in mind. Where possible, structures and props should be frames only; solid structures occlude markers, making the motion capture data noisy and fragmented. In addition, props need to be the right dimensions and weight, or else performance believability will suffer. The props Namco used in SOUL CALIBUR added a lot of believability; you can actually see the weight of the weapons as the characters swing them around.

**Audio.** When characters are speaking or responding to speech or other sounds, it's important to have the audio playing during the capture. Actors should actually vocalize along with the prerecorded dialogue during the capture session; this extra measure of authenticity will come through in the performance. You cannot synchronize motion capture data to specific audio unless you use that recording during the performance.

For example, if you have three people talking to each other and you want both body and facial animation, what are the logistics of getting everything to synchronize?

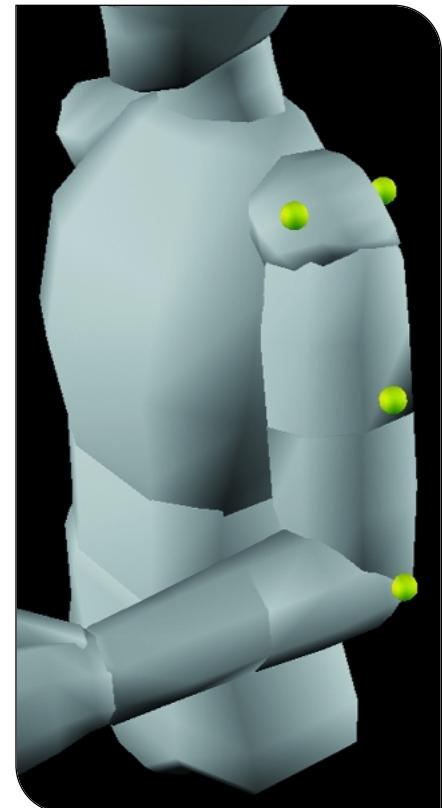
nize in post? Assume that the body and facial animations will be captured separately. First, there should be a source audio file with all the characters interacting for the entire shot. From that source file, cut out the pieces that represent the body takes and use them during the body capture. Then cut out all the little pieces of dialogue from the body takes to be used for the facial takes. Create a hierarchy of audio files: the facial audio files are children of the body audio files, and the body audio files are children of the source audio files. It's critical to keep track of the frame range numbers extracted from each parent file so you know the exact sequence and ranges to blend back into one take.

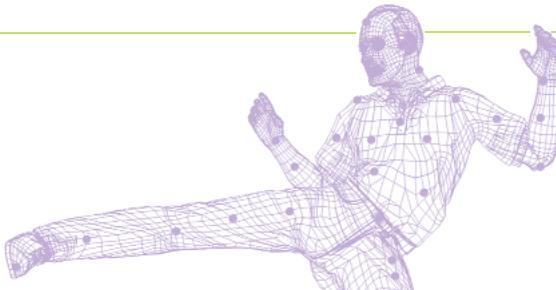
Continuing with the preceding example, let's say you have 30 seconds of interaction between the three characters. Jim and Ed are arguing, when Tim enters and solves the dispute. During this interaction, the camera cuts to three different perspectives. The body takes are decided by the actions, and the facial takes are decided by the length of the dialogue. The first body take will be Jim and Ed arguing for 16.8 sec-



**FIGURE 2 (above).** Knees, armpits, and elbows need enough vertices to allow each joint to bend in the right place. The correct setup is on the top; the skeleton on the bottom has been set up incorrectly.

**FIGURE 3 (right).** It has always been difficult to get good motion capture data for shoulder joints because of their complex anatomy and movement. Placing markers on the front of the shoulder, the back of the shoulder, and down the arm a few inches articulates a plane for the shoulder rotation.





onds. The second body take will be Tim entering and approaching for 3.2 seconds. The third body take will be Jim, Ed, and Tim for 18.8 seconds. (Capturing extra time provides some freedom when cutting the scene.) Then capture 12 facial takes ranging between 1.2 and 8 seconds. Using the table you created that lists the frame ranges of each audio file (Table 1 shows an example), you know the exact frame number where the sixth and seventh facial takes fit in the 3.2 seconds of the second body take.

Obviously, audio must be recorded and prepared before the shoot. Create each body audio file and each facial audio file, and prepend each file with a specified number of beeps or tones so the actors know exactly when to begin each motion.

**Shot lists.** Like the character definition step, the shot list should be thorough and complete in numerating and accounting for each shot needed. A poor shot list will limit the ability of the director and the actor to produce a believable performance. Time will be wasted, props and set pieces may not be prepared, and some motions may require the motion capture volume (the stage) to be reconfigured.

When defining a shot list, it is important to be clear in describing the exact motion desired. Motion should not be confused with emotion. For example, “sad” is not a motion. It is, however, a fine descriptive word for the quality of a particular

motion. “From a neutral position, actor slumps shoulders forward, tilts head to the left, shakes head back and forth” is a fairly complete description of a motion. “From a neutral position, actor slumps shoulders forward, tilts head to the left, shakes head back and forth sadly” will only help to reinforce the desired quality of the movement.

## Performance

**C**reating good animations out of a performance obviously requires starting with a good performance, but achieving a desirable performance is its own challenge. Performance is where observing even small considerations can have a big impact on results.

**Casting.** Casting the right talent to do motion capture is the first step and one that is overlooked too often. Designers and producers need to take the time to audition talent until they find the right person who will bring the character to life. They must have the skills related to the character: a dancer, a soccer player, a martial artist. They should match the character’s scale and proportions as close as possible. Strong muscle memory (the ability to remember the exact ready pose or idle pose and return to that pose) is required for in-game characters, and actors should be auditioned with this in mind. Mimes or actors with mime training

usually have good muscle memory. All characters that interact with each other need to be cast in relation to each other’s size so that those interactions can be performed realistically. Performance is one case where what you see is truly what you get. For example, preference to an old injury that was not noticed during the performance may be painfully obvious in the animation. You can’t fix an animation if it wasn’t performed correctly.

**Character interpretation.** Character design

and performance are closely related. Understanding the character design and performing the motions is what acting is all about. An actor cannot get into character if he or she doesn’t know the character. The director and the actor need to see artwork of the character and get to know its personality, its special abilities, and its role in the game. Even a simple walk cycle should be performed based on the particular character design. Character interpretation is subjective and will often differ from designer to artist to director to actor. Clear and complete character design will help to bring varying opinions closer to agreement.

**Directing.** Using an experienced director will significantly affect the quality of the performance. The director needs to understand the motion capture technology, the software pipeline, and how the data will be implemented. A great performance is of no use if the cameras can’t see the markers. A good director also knows the performance he or she is looking for and doesn’t miss it when it happens. Often poor performances are accepted and pushed through the production pipeline, only to be cut later on.

Sequencing the motion list so the actor can stay in character is also helpful. It may be difficult for an actor to perform a death by fire, followed by a sword fight, an idle, a run, and then another death by sword. In addition, separating motions according to their degree of difficulty saves time. The motion capture technician may spend a great deal of time replacing markers that fall off during aggressive takes, so it is usually best to group similar motions together: idles, walks, runs, fights, deaths.

**Attendees.** The lead designer and lead animator should be at the shoot. The designer should be available for questions and watching the performance to evaluate it for gameplay. The animator should be taking notes on the performance so he or she can understand how the markers are going to make the character move and how characters will interact with virtual sets and props.

**Etiquette.** Studio etiquette is required for the actor to give the best performance. Too many cooks in the kitchen will ruin the performance. Only the director should give

ID	Filename	Offset	Length
1	20_02_0090_Nep-Welcome	90	138
2	20_06_0000_Kat-You know me	0	210
3	20_06_0231_Nep-Of that	231	152
4	20_06_0403_Nep-I knew your	403	296
5	20_06_0721_Kat-Seems everyone	721	244
6	20_06_0978_Nep-It is the time	978	177
7	20_06_1172_Nep-Your mother	1172	245
8	20_06_1441_Kat-Well it certainly	1441	89
9	20_06_1530_Nep-Wait a moment	1530	199
10	20_06_1754_Nep-It was your mother's	1754	188

**TABLE 1.** Macro Express can create a series of macros that extract the separate audio files and create a spreadsheet with all the audio file names and frame number offsets that are needed to sync and blend all the motions back together. The offset numbers are embedded in the audio file names. The spreadsheet is used as a shot list during capture, and for tracking progress and as a reference during post.

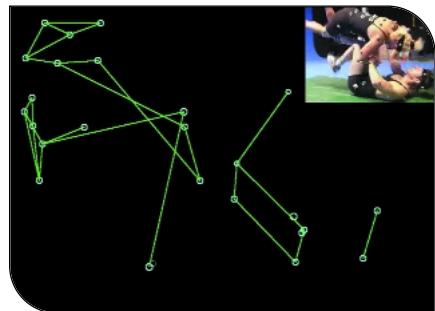
direction — an actor that has been over-directed can't even walk. A chain of command should be established so the director takes input from interested parties and then decides if and how to direct that input to the actor.

## Noise

**O**bviously you can't expect an animation to look like the performance if you have noisy marker data. A good example of noise is the jitter you see in a character's foot when he stands still. There are many variables to address when optimizing a system to get the cleanest possible marker data. We won't discuss this in detail, but we know that cameras must be configured with the right lenses, optimally positioned around the capture volume, and correctly calibrated to each other. Positioning cameras at different heights provides better vantage points and better triangulation. When all cameras are at the same height, they tend to produce noisier data. Dirty or misshapen markers, bad camera cables, outdated algorithms, fluctuations in the video capture signals, and Gooch's microwave can all cause noise.

Noise is also introduced when an actor performs at or near the limits of the capture volume or when markers are occluded. Motions need to be choreographed for optimal marker exposure while staying within the motion capture volume.

For example, when an actor lies on his back and his chest gets covered by a prop or another actor, chances are the anima-



**FIGURE 4.** It ain't mocap if you can't see the markers. Some of the markers on the bottom actor have been occluded during this motion sequence, jeopardizing the likelihood of getting clean animation out of his performance.

tion won't look like the performance because of occlusion (see Figure 4). The cameras can't see the markers on his back when he's lying on them, and the other actor is covering his chest markers.

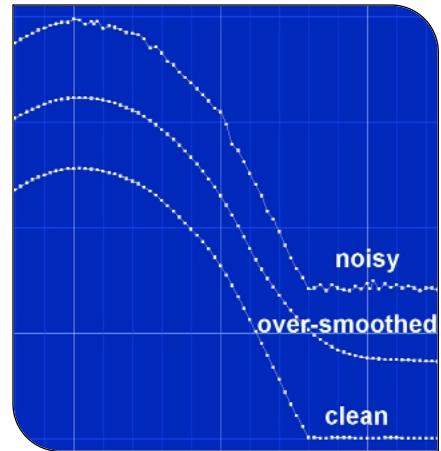
**Cleaning.** Filtering out noise in translational data, or smoothing, cannot be applied with a broad stroke. Each frame of each trajectory (X, Y, and Z) for each marker must be evaluated individually. Too much filtering will ruin motion data too. Motions that look soft or mushy have been filtered too much (see Figure 5).

Cleaning motion capture data is a skill developed by experience. The amount of noise or data quality is usually directly related to the amount of experience the data wrangler has and the amount of time available for cleaning the data. Experienced wranglers know how to use good data from one marker to fix bad data in a related marker. They recognize the difference between spikes, tails, noise, and good data. They know the order in which markers should be cleaned, just how much to smooth a marker, and when to fill gaps. Without the right tools, cleaning noise can be very tedious and time consuming.

**From bad to worse.** Motion capture data should be cleaned at the raw marker level. When you filter hierarchical rotational data, the children get really mad, so if you smooth a noisy elbow, the child hand is going to become even noisier. To understand this, it's important to understand how markers drive joint rotations.

Markers are translational data only, individual points in space over time. Three or more markers are used to create a plane, and the translation and rotation of that plane is what drives a joint. One plane drives the elbow and another plane drives the hand, and the two planes have their own data sources. When noisy elbow markers and clean hand markers are baked into the skeletal data, editing the parent elbow joint causes the child hand joint to leave its clean path and be augmented by its parent's new path.

It's the wrangler's job to produce good skeletal motion capture data. Markers that are noisy or have been swapped or mislabeled should be reprocessed and cleaned using motion capture tools such as House of Moves' Diva or Kaydara's



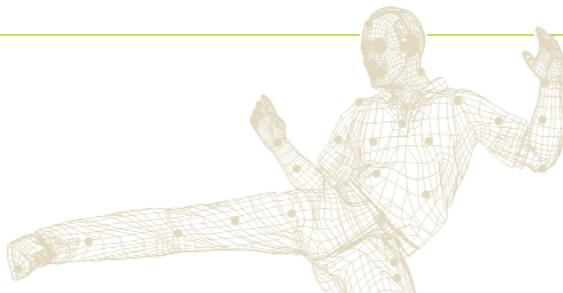
**FIGURE 5.** Graph of a jittery foot marker, a clean foot marker, and an over-smoothed foot marker.

Filmbox (see page 8 for a review of Filmbox). The motion capture department should have the source data, the right tools, and the experience to fix the problems.

Joint constraints and inverse and forward kinematics setups can cause problems, too. When animators keyframe skeletal data, too many keys often end up getting cut. Before you know it, the performance you saw captured . . . is gone! Filmbox has keyframing tools that preserve the motion data and allow motions to be modified without deleting keys.

**Real-time by-products.** One of the solutions hardware vendors have been working on is real-time motion capture. Broadcasters and filmmakers have been patiently waiting for real-time optical motion capture systems to portray their virtual characters. Game developers should be very excited about the by-products of real time, such as instant playback, updated algorithms for cleaner data, and no labeling or marker cleanup. When these real-time products mature a bit more, an optimized setup could mean that the time taken to reconstruct, label, clean markers, and fill gaps will happen in real time during the capture, and animators would have quality data the same day.

Even with the latest real-time products and the best camera setup, noise is still a problem that motion capture hardware developers need to work on. More development needs to be put into the algorithms that calculate the 3D coordinates.



## A Working Software Pipeline

All the proper planning, perfect performances, and masterful noise cleaning in the world won't produce WYSIWYG motion capture without the right software pipeline. The following is my current setup that I have devised in my ongoing attempt to achieve pure motion capture.

The **Vicon Workstation** is a solid tool for capturing data, calculating the 3D coordinates, and labeling the raw marker data. The Vicon system is a retro-reflective, optical motion capture system.

**Macro Express** is a very efficient way to set up hotkeys that copy file names and descriptions from a shot list into the Vicon interface. Rather than typing in each file name, notes, and descriptions, or renaming files after the capture session, you just press one key and you're ready to capture the next take. Once you start using macros, you won't be able to go back; you'll always find new ways to improve efficiency.

**Diva**, a brand-new product by House of Moves, is another very powerful tool. Diva replaces all other marker cleanup tools. It imports and exports almost every type of motion capture data and is a powerful component in any production pipeline. With a marker set designed to take advantage of Diva's tools and scripts optimized for the data it's processing, the bottleneck in motion capture production shifts to marker labeling.

What makes Diva so useful is that it was (and is being) designed in a production house. The wranglers on the front line who have to produce clean data every day have provided the input to build this panacea application. House of Moves listens to its users. The studio takes all requests and asks for clarification when

### FILMBOX TIP

When setting up the Actor init pose, notice that the shoulders need to be translated up and back, away from the torso, just like the markers do when the live actor raises his or her arms into the init pose. When this is not done, the character will have droopy shoulders.

the request is not clear. Valid requests are prioritized and implemented as soon as possible. House of Moves provides simple, customized scripts for special needs, on the same day when possible. The interface is completely customizable, and the best parts of the Maya and 3DS Max GUIs have been rewritten for Diva. Every imaginable tool that you would ever want to use with raw marker data is scriptable, batchable, and assignable to buttons, hotkeys, and marking menus. Diva is surprisingly mature for its age, but it is still maturing and growing beyond expectations.

Kaydara's **Filmbox** is the tool for mapping motion capture data onto a skeleton. It has taken giant steps to fill the need for a precise and flexible solution to constrain skeleton joints to marker data. The Actor and Character tools are innovative, powerful, and smart. Filmbox includes a nonlinear editing interface that allows you to create loops and blends, and the Control Set is the answer to keyframing data that needs to be modified without destroying the motion capture data. Every motion capture department should be using Filmbox. My pipeline depends on it. Nonetheless, many features are buried in its obtuse and cumbersome interface. It has a very steep learning curve, and many tools and functionality still need to be added.

Westwood animators use the **Maya**, **3DS Max**, and **Lightwave** animation packages. The skeletons built in these packages are imported into Filmbox, where the motion capture data is applied and then saved out in the Filmbox file format. These animation packages then import/merge back the animated .FBX data into the scene. The skeleton in the animation package and the skeleton in Filmbox must be identical or the merge back will not work correctly.

Discreet's **Character Studio** is very easy to use. Using the right motion capture data, it's a simple process to load motion capture data onto a character. Many animators prefer Character Studio because of its layering tool, which is used in keyframing, looping, and blending. Character Studio is well suited for many projects, but it can be limiting. For example, Character Studio only supports the .BVH hierarchical data format or the .CSM data format, which is limited to a specific marker set that often causes occlusion of the chest, knee, and

elbow markers. With a different marker set, Diva can be used to generate the required .CSM markers and then export clean .CSM marker data.

## Work in Progress

The efficiency and quality of character animation, for better or for worse, is now directly tied to the science of motion capture. Arguments against the artistic integrity of using motion capture are rapidly falling by the wayside as more game developers are anxious to capitalize on the realism that motion capture affords. Motion capture can save time and money, but software pipelines and techniques for implementing motion capture data need to be improved and refined continuously. Experienced wranglers are becoming valuable assets.

Slowly, the challenges that have plagued developers trying to turn motion capture into accurate in-game animations are being improved or eliminated. Real-time technology is advancing and will eventually resolve the problems of raw marker noise and develop robust interpolation for marker occlusion and dropout. Skeleton-solving tools will become more powerful and flexible, and new tools will be developed for animators to keyframe over motion capture data. Those of us in motion capture anxiously await such developments, hoping that one day WYS will truly be WYG.

## FOR MORE INFORMATION

House of Moves' Diva

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

Insight Software Solutions' Macro Express

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

OMG's Vicon

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

Motion Analysis's Eagle Camera

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

Kaydara's Filmbox

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

Discreet's 3DS Max and Character Studio

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

Alias|Wavefront's Maya

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

Newtek's Lightwave 3D

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

# Test Drive

**On the Open Road  
with Two of Today's  
Most Powerful  
Game Engines**

**G**reeted initially with fair helpings of both optimism and skepticism, middleware has been gaining momentum over the past couple of years. New products are appearing all the time, and older ones are maturing nicely with revisions and refinements based in part on feedback from developers using these tools in shipping products.

Of all the different kinds of middleware available today, licensable game engines are clearly the most complex to make workable in a production environment. The promises of shorter and less painful development cycles are many, while the burden on those charged with incorporating licensed technology into a project remains considerable. In many cases, developers find that licensed technology didn't save the months of project development time they thought it would, but instead that time and effort ended up redistributed to other areas such as art, game design, and story development. This kind of trade-off is gaining appeal: development cycles won't shorten themselves, nor can most developers stand to see them grow any longer.

Here we present the first in a recurring series of features that will take an in-depth look at engines available to developers. For this installment, we sat

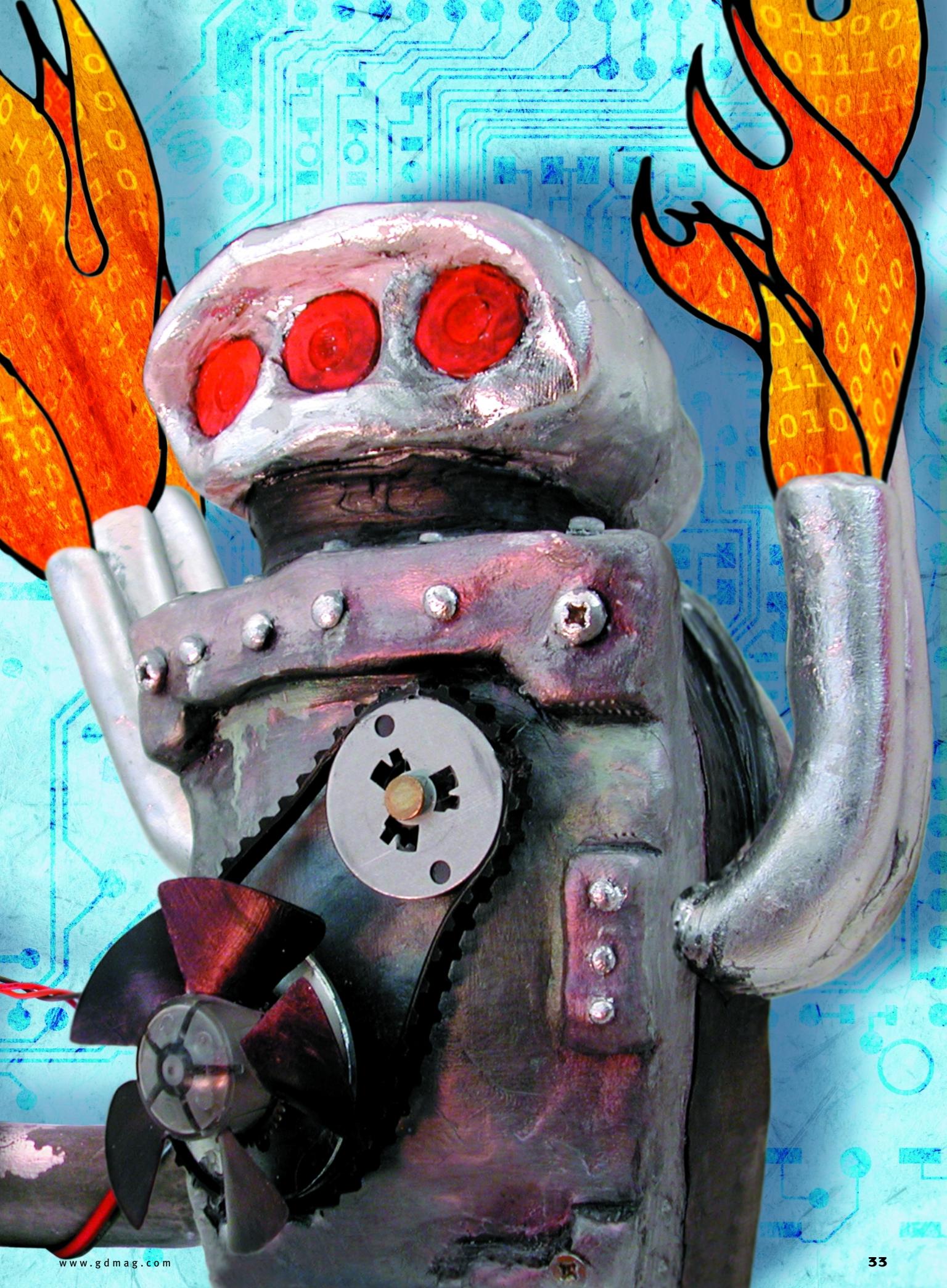
two experienced game programmers down with two different engines: NDL's NetImmerse — one of the more mature offerings on the market — which has been used in numerous PC and console titles including, recently, Oddworld Inhabitants' MUNCH'S ODDYSEE and Mythic Entertainment's DARK AGE OF CAMELOT; and Intrinsic Graphics' Alchemy, a more recent arrival on the scene, which debuted in October 2000.

— Jennifer Olsen

**ANDREW KIRMSE** | Andrew's previous projects include the games MERIDIAN 59 and STAR WARS: STARFIGHTER, and contributions to the Game Programming Gems books. He is currently a lead programmer at LucasArts, where he can be reached at [akirmse@lucasarts.com](mailto:akirmse@lucasarts.com).

**DANIEL SÁNCHEZ-CRESPO** | Dani has a degree in computer science from the Universitat Politècnica de Catalunya. He has a four-year relationship with the Computer Graphics Group at UPC, where he has been active in the fields of human-computer interfaces and real-time graphics. In March 2000 he founded a game development studio called Novarama. You can reach him at [dani@novarama.com](mailto:dani@novarama.com).





# Intrinsic Graphics' Alchemy 1.5

**A** relative newcomer to game middleware, Intrinsic Graphics Inc. was founded by former Silicon Graphics employees in 1999 and released version 1.0 of its flagship product, Intrinsic Alchemy, in October 2000. For this review, I looked at Alchemy 1.5, which supports development for the PC, Sony's Playstation 2, and Microsoft's Xbox. Version 1.6, which adds Gamecube support and additional features and includes some bug fixes, should be available by the time you read this. A major new 2.0 release is planned for February 2002. At the moment, the PC and Playstation 2 versions are the most mature. Supported compilers are Metrowerks Codewarrior for the Playstation 2 and Gamecube, Microsoft Visual C++ for the PC and Xbox, and SN Systems ProDG and GNU C++ for the Playstation 2 (though not yet for the Gamecube).

I tested Alchemy on a 1GHz Pentium III with 512MB of memory, an Nvidia Geforce 2 MX graphics card, a clean install of Windows 2000 with service pack 2, and Microsoft Visual C++ with service pack 5. There is one installation for the programming environment and another for the Artist Pack, which contains the exporters and art conversion tools. The installation is straightforward, though it shows some rough edges: it places shortcuts to the Alchemy documentation on the Start menu without asking, yet it requires a manual setup of environment variables and shortcuts to frequently used executables, such as the art preview tool.

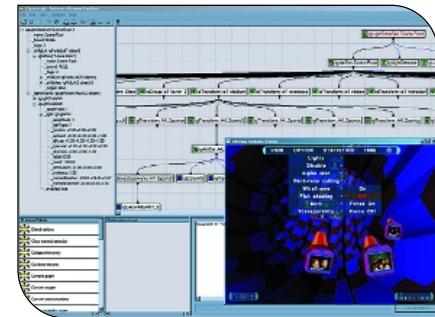
## Art Path

**A** lchemy ships with exporters for 3DS Max 3.0 and 4.2, Maya 3.1 and 4.0, and Lightwave 6.5 (the Max exporters are currently the best supported). Max can be configured to show a real-time, in-engine view of the model in one of its viewports, while Maya can launch the viewer from a menu. The Max exporter adds numerous rollouts to the interface, including the option to optimize the result for the PC, Playstation 2, or Xbox.

One of the exporter's convenient features is its integration with the preview tool Insight. A PC can be configured so that an exported model is automatically sent down to a Playstation 2 debug station via a USB connection, where it appears on the screen after a few seconds. From within the Insight viewer, an artist can rotate around a model, activate its animations, and enable or disable various rendering features to see their effect on frame rate. The viewer is equipped with on-screen counters detailing memory usage, scene graph traversal time, rendering time, and frame rate.

Animation is bone-based, using the tools built into the modeling packages. Alchemy supports skinned animation, with up to four bone weights per vertex. On the Playstation 2, models that restrict themselves to a palette of 16 bones are skinned on VU1; models with more bones are skinned on the CPU and VU0 at a greatly reduced speed. In addition to exporting keyframes from inverse kinematic (IK) animations in the modeling tools, Alchemy has its own run-time IK solver for two-bone chains.

The exporter exports .IGB files, which are collections of art assets in a proprietary format. An unoptimized .IGB file contains model data in a platform-independent representation. Alchemy can apply a sequence of optimizations at export time or as a postprocess to accelerate the rendering of a model for each platform. Examples include computing triangle strips, flattening the scene graph, or decreasing texture resolution. A sequence of optimizations can be saved as a script, so that a particular model is always exported with those optimizations applied. A GUI tool called the Finalizer allows an artist to modify optimization parameters interactively and view the resulting frame rate and memory usage in the Insight viewer. Alchemy also ships with a command line tool that applies an optimization script to an .IGB file, which is useful for batch processing. Applications can define their own optimizations, which are then available for use in these tools.



Alchemy's Finalizer, showing optimizations being applied to a 3D scene.

## Touring the Subsystems

**A** lchemy's PC version includes one renderer for OpenGL and one for DirectX 8. The Playstation 2 version is implemented through custom VU1 microcode. Applications can still send their own microcode to VU1 through an extension interface.

The current graphics state is stored in an object called the visual context. Invoking methods on the visual context changes the current drawing state; the model is very similar to that of OpenGL. The abstract visual context class handles settings common to all platforms, such as textures, lights, and material settings, while platform-specific subclasses allow more direct access to hardware features, such as the Playstation 2's quirky blend modes and mipmapping settings.

Alchemy stores model data in vertex and index buffers in the style of DirectX 8. The system supports dynamic geometry by marking in advance those buffers that will change, then retrieving direct pointers to the buffers, making modifications, and committing the changes. Naturally, dynamic geometry involves a performance penalty that static geometry does not.

Intrinsic's developers have tuned the performance of its Playstation 2 renderer considerably. One real-time demo shows 36 skinned characters of 2,500 polygons and 18 bones each animating at 60 frames per second, while another shows a texture transfer rate of 600MB per second. Given the Playstation 2's deserved reputation as a difficult platform to work with, this kind of performance represents a significant time investment. Starting from scratch, achieving these levels might take six to twelve months of dedicated development.

Alchemy's input system supports the Playstation 2 and Xbox controllers, including analog buttons, the multitap, and force feedback. An important omission is the debugging keyboard for the Xbox and the Playstation 2 (via USB). Applications can

either implement platform-specific keyboard classes or replace the input system altogether without disturbing other subsystems.

Alchemy implements audio through DirectSound on the PC and Xbox and a custom IOP module on the Playstation 2. The current implementation is rather inefficient on the Playstation, as all audio requests are immediately communicated to the IOP through a remote procedure call. Version 2.0 of Alchemy will include a deferred mode, in which audio requests are batched together for better performance. It supports both streamed and nonstreamed audio.

With Alchemy, Intrinsic has foreseen the need for application-defined memory usage. Without application intervention, objects are allocated from a default memory pool, which is a wrapper around `malloc()`. Applications are free to create memory pools that redirect allocations to a custom allocator. In addition to the default pool, Alchemy provides a fixed-sized memory pool and two variations of pools that work on the stack discipline (last in, first out). Console games in particular require this kind of absolute control over memory usage.

## Nuts and Bolts

One of the most distinguishing characteristics of Alchemy is its particular use of C++. The libraries themselves make use of a minimalist subset of the language; there is no use of multiple inheritance, run-time type information (RTTI), the Standard Template Library (STL), or the `new` and `delete` operators. This approach is something of a mixed blessing, as it ensures cross-platform compatibility and requires only a passing familiarity with the language, but places C++ aficionados in the uncomfortable position of programming outside of their usual idiom. Without `new` and `delete`, for example, programmers must instead call the special static method called `instantiate()` to create new objects. This can be more than a nuisance — it means that these objects cannot be declared on the stack, used as (nonpointer) member variables, or passed by reference. The Alchemy headers also make heavy use of templates and nested namespaces. Any project that needs a thorough understanding of Alchemy's architecture will require at least one programmer well versed in the nuances of C++.

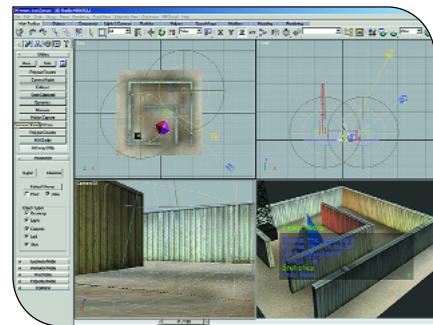
Alchemy's object system is based on a Scheme-like class description language called ODL (Object Description Language). In the header file for each class that's derived from the base `igObject` class, a section of ODL code appears at the top, describing the name, type, and default value of each of the class's data members, called fields. All of the ODL class files are preprocessed into a single source file, which is then compiled like normal C++ code.

A C++ class description language will appear both strange and familiar to game developers. The approach is similar to scripting languages in some games, with similar advantages. Repetitive code such as `get` and `set` methods can be generated automatically by the preprocessor, fields are automatically persisted along with an object, and code can enumerate field names and values at run time, much like the Reflection API in Java. This kind of flexibility is critical in tools, which typically expose hundreds of settings to level designers. Simple projects will have no need to extend Alchemy's ODL class hierarchy (or even be aware of ODL), though doing so is straightforward and well documented. It is, however, a bit inconvenient. The custom build step required under Visual C++ for ODL preprocessing is convoluted, and it isn't even possible to do this within the Codewarrior IDE, requiring a manual prebuild step (for example, through a makefile). New ODL classes are required to add scene graph node types or traversals, so most commercial projects will need to understand the language and extend the class hierarchy.

## Support

Intrinsic's product support is generally excellent. A small full-time support staff answers questions quickly and knowledgeably. Because support engineers are located in the U.S., Europe, and Japan, replies are possible anytime during the day. Questions that stump the support engineers are passed on to the product's development engineers with minimum delay. A training course is included with Alchemy's licensing fee, which will help familiarize a project team with the design and implementation of the libraries.

Alchemy comes with a fairly complete set of HTML documentation (also avail-



Alchemy's Insight viewer running within one of 3DS MAX's viewports.

able in PDF format, suitable for printing). All of the major subsystems have thorough, plain-language descriptions and small code examples. There is also a comprehensive set of documentation for every C++ class in the system, which is automatically generated from the source code.

As a rule, Alchemy does not come with source code. Source is included for the exporters and the Insight viewer, but not the core itself. The .JGB file format, which contains all art assets, is undocumented. And some of the plug-in optimizers come with source, while others don't. Intrinsic is willing to make some source available for support purposes on a case-by-case basis.

## The Bottom Line

Alchemy is clearly still in its first generation, but even at this stage, it's worth a look for certain projects and should be a strong contender with the 2.0 version. By the time you read this, at least one major title using Alchemy should be announced. The library, like the company itself, is new and unproven, and it shows in the scaffolding that is visible in places throughout the code and tools. Sometimes, though, it takes a new entry to push the cutting edge.

With some planning, it's certainly possible to pick and choose the parts of Alchemy to integrate into a project. As a practical matter, though, project teams are better off swallowing Alchemy whole, accepting at a minimum its renderer, object model, scene graph, and art path. For some developers, restricted access to the source code for such a large part of a project would be a scary proposition indeed. But for others, Alchemy is well worth the time saved. Middleware is usually best suited to developers who are in a hurry or who want to leave the details of the hardware to someone else; Intrinsic's Alchemy is no exception.

# Numerical Design's NetImmerse 4.0

**N**etImmerse is an open game engine or, more accurately, a game toolkit. By supplying a wide array of features and technologies out of the box, game toolkits don't attempt to eliminate, but do significantly reduce, a developer's coding needs. NetImmerse 4.0, the latest version of this already well-established product, continues to improve upon this design philosophy. Boasting enhanced multi-platform support and many cutting-edge features, it looks very promising but, does it live up to the expectations?

## Architecture and Features

**N**etImmerse is a scene-graph-based toolkit, providing a set of primitive nodes and traversal algorithms to access the data in an efficient manner. The scene graph itself is a Directed Acyclic Graph (DAG), where each leaf node represents world entities. These entities can be triangle (normal and stripified) meshes, portal and particle systems, skinned characters, and so on. Organizing the data into a graph structure allows hierarchical tests to be performed, and irrelevant data can be efficiently discarded on the fly.

The core engine is object-oriented, written in standard C++. The distribution comes with source code (and convenient Visual C++ project files), so it is possible to add new primitives to the scene graph. All you have to do is derive new classes from existing ones to create the new functionality. File access is managed by an I/O abstraction layer, and rendering is done through platform-specific renderers, which can access Gamecube, Playstation 2, Xbox, and PC hardware. For the PC, both DirectX and OpenGL are supported, with DirectX 8.0 being the preferred option. By using these renderers, NetImmerse simplifies porting your title among the different platforms. Some features (especially eye-candy-style features) aren't supported in some systems, but in case of discrepancies, the offending feature is simply ignored, so the game is still

playable. Besides these minor glitches, games built with NetImmerse should be rapidly portable.

As with most scene graph management tools, graphics is the driving force behind NetImmerse. The engine boasts a rather impressive feature set: triangles and triangle strips (which can be generated with a handy bundled processor), portal rendering (both the strong/convex and weak/non-convex visibility approaches), mesh skinning, particle systems, a configurable texture engine (supporting diffuse, specular, gloss, shadow, environment, and bump maps), and many more.

NetImmerse includes an interface that wraps on top of the Miles Sound System, which must be purchased separately. Thus, you can expect all state-of-the-art audio features: positional sound sources, environmental audio, MP3 playback, and so on. As far as physics are concerned, NetImmerse sports a dynamic collision detection library, but it doesn't include a full-fledged physics module. Still, good synergy exists between NetImmerse and the Havok physics toolkit. Some customers are already using both packages together, and as version 4.1 of NetImmerse ships in January, we can expect an improved interface to Havok.

NetImmerse can work with data extracted from 3DS Max or Maya. All systems are supported via plug-ins that make editing and exporting art assets quite straightforward. You can assign texture maps, tweak UV coordinates, and create animation loops all within your tool of choice. Then, all you need to do is use the plug-ins (which include real-time previews of what your scene will look like) to create .NIF files, the NetImmerse file format. In some cases, specific tools are required to tune the data for the various uses, be they terrain, LOD data, animation keyframes, or others. The supplied tools are well designed and documented, but familiarizing yourself with the conversion process can take some time. Some tools are command-line only, so the learning curve is rather steep.



Totally Games' *STAR TREK BRIDGE COMMANDER*, a space adventure, was developed using NDL's NetImmerse.

## Documentation and Samples

**T**he documentation for NetImmerse comes in digital form only, and consists of both manuals and samples. The documents come in Windows Help File format. Manuals are also provided for artists, covering their specific subjects and tools: plug-ins, modeling tips, system limitations, and other topics.

The code samples are organized in three categories. First, dozens of test scenes showcase the art production pipeline. Second, the tutorials, which are small test applications, are a great place from which to borrow code while you're getting started. Third, several advanced demos are made up of lengthy pieces of code that combine all elements together to create more involved applications. Some of these demos are quite sophisticated and really give you a glimpse how NetImmerse can be used to create a cutting-edge title. In an ideal world, including the source code from a completed commercial project would be a nice touch, granting users some perspective on how NetImmerse works within the whole production pipeline. Even with the dozens of demos available, the gap between a demo and a full game becomes pretty wide, and more help and documentation in this respect would certainly be appreciated.

## The Production Pipeline

**N**etImmerse is a well-built and feature-rich package, but what's it really like to work with it? What should your company expect?

To begin with, man-hours will shift from brute-force programming to soft-

ware engineering and analysis.

NetImmerse certainly frees you from many time-consuming programming tasks, offering tools and building blocks that would require months, if not years, to build from scratch. NetImmerse is a mature package, and component quality is generally top-notch. Still, a clear vision on how to make these bricks work together is fundamental. Spend some time reading the documentation and understanding the potential of each component. Don't be tempted to code something in-house when the desired feature may already be somewhere in the toolkit.

When you've read the documentation, the fun part begins: translating your game design in terms of what NetImmerse offers and figuring out how to get things done using the existing tools. Staying away from a keyboard and relying on the toolkit might be hard to get used to, but in the end it's certainly worthwhile from a time and results standpoint.

From an artist's perspective, NetImmerse is great. All game content can be created with your modeler of choice, and little or no training effort is required. Both the supplied plug-ins and the artist-specific manuals and samples do a great job of keeping the artists in a familiar environment where they are most productive. But there still exists the no-man's-land that affects art tuning and importing: Who will take care of using the different tools (stripifiers, keyframe tweakers, terrain cachers) to fill the gap between the art package and NetImmerse and ensure that art assets make it safely to the main engine? In today's games, where assets are counted by the thousand, this gap can be a potential problem, so you should face it from day one. Having dedicated personnel handling that incoming datastream is a wise move, as the tools to be used are likely beyond the skills of the average artist. It all boils down to the type and amount of data to be gathered and the technical expertise of your team.

From a production perspective, your company will experience a faster deployment phase. Once the game is working on one platform, deploying it on others is far faster than having to port it by hand. All you need to do is test the code on the new system and fix all glitches and anomalies

caused by each platform's nuances. The only caveat here is that you need to pay additional license fees for each platform, but these are certainly less costly than a full port and, luckily, multi-platform licenses are available at reduced cost.

Assuming that your game is to ship on all four platforms (that's PC, Xbox, Playstation 2, and Gamecube) the overall license fee is \$200,000, compared to the \$100,000 you should pay to make your game available for just one of the game consoles (or \$75,000 for PC-only titles).

## Where It Takes You

The benefits of using NetImmerse are clear and proven. Having cutting-edge features available from day one allows for smaller teams, faster prototyping and development, and more time dedicated to creating better content. Additionally, using a scene graph model gives the developer creative freedom over the game design. Still, your mileage may vary, as not all game genres will take equal advantage of the toolkit's features.

A media-rich game that uses simple AI and physics (such as an action/adventure game) can be created easily using this package. The geometry, collision, terrain, indoor, and animation components will likely fit your needs and save lots of engineer-hours. All you have to do is understand NetImmerse's layout and concentrate on creating the proper AI and game logic to ensure smooth gameplay. Still, if you are creating something like an RTS title, you will experience a bumpier ride. In these types of projects, media is usually secondary, as you need to free as many CPU cycles as you can for your game's AI. Two issues will certainly arise. First, you will only use the basic features of NetImmerse's scene graph, and the advanced functionality will remain untouched. After all, RTS titles with shaders, environment maps, and inverse kinematics aren't that common, right? To complicate things further, after the graphics engine is in place, you will have to write the AI system from scratch. NetImmerse does not provide the finite state machines, scripting languages, pathfinding solvers, and other AI systems that you will certainly need to complete your game.



OPEN TENNIS, a fast-action sports title from Montreal-based developer Miroids, was developed using NetImmerse.

So it seems NetImmerse is biased toward media-rich games (action, RPG, adventure, simulation, and sports) rather than behavior-rich games. (Clearly, crafting something like THE SIMS or BLACK & WHITE with NetImmerse would be complex.) Outside factors, however, may influence your purchase decision. Many developers doing action titles find it preferable to create their own graphics engine and will thus be uncomfortable using NetImmerse. On the other hand, a studio doing a RTS game may have a great team of AI coders but no graphics programming skills. In this situation, even if the tools provided by NetImmerse are not especially designed to do an RTS title, having a toolkit that saves the burden of creating a graphics engine will be a blessing for the team building it. So you really need to examine your team, understand its skills (and thus where can NetImmerse help), and analyze just what kind of game you really want to build.

All in all, NetImmerse offers a host of features and time-saving utilities that can shorten your development path. Once you've passed the initially steep learning curve, NetImmerse can help you concentrate on gameplay and content and free you from the burden of crafting a new engine. The latest release keeps up very well with the times, offering state-of-the-art features such as programmable texturing and hardware skinning to keep the wow factor high. Still, this is not a package for the faint of heart. NetImmerse may give you powerful building blocks, but your hard work is still required to build a great game. ☺



## Pseudo Interactive's

# CEL DAMAGE

**KEVIN BARRETT** | Kevin is the project director and lead designer at Pseudo Interactive. CEL DAMAGE is Kevin's first videogame release, though he has been an active designer in the adventure game industry for more than 15 years.

**JOHN HARLEY** | John is PI's quality assurance lead. He joined the CEL DAMAGE team earlier this year. Active in the online FPS community, he is also content director of XMedia, out of xenoclan.com.

**RICH HILMER** | Rich is the development lead at PI, which is a suitably broad title for the diverse role he fills. He combines programming, level building, and design and generally avoids bossing other people around.

**DANIEL POSNER** | Dan is PI's code lead, a role made easy by the quality of the code team. He believes anything can be accomplished with teamwork, passion, and optimism.

**GARY SNYDER** | Gary is the art director at Pseudo Interactive. Responsible for a talented team of artists and animators, he has the added bonus of getting to watch cartoons whenever he wants in the name of "research."

**DAVID WU** | David is PI's president, director of technology, and the creator of the CEL DAMAGE game engine. He is on a mission to prove that in humankind's endless pursuit of happiness, good physics is just as vital as good coffee.

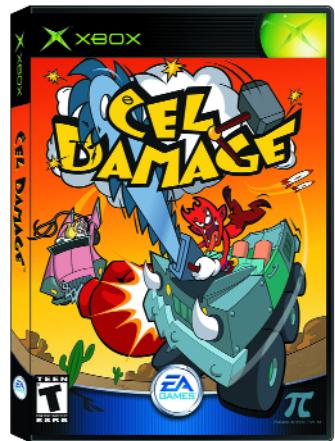
The story behind CEL DAMAGE is long, winding, and harrowing, but ultimately uplifting. And because CEL DAMAGE is our first published title, its story is also the story of our company, Pseudo Interactive. Based in Toronto, we began work on the technological core of the game four years ago. A demo of our driving-combat physics engine at the Game Developers Conference in 1997, PI's first year of operations, received a warm reception. Shortly thereafter, PI struck up a relationship with Microsoft's Entertainment Business Unit (EBU). Over PI's first two years, we started up and killed a few projects. However, with the coming of Xbox, we found a proper niche for our emerging technology.

The physics engine that PI president and technology director David Wu was developing lent itself well to console applications. EBU recognized this, and an early alliance was formed between PI and the embryonic Xbox team. A high-profile Microsoft producer came to PI with a vision of where PI needed to take its game technology, and a new project was born. At that time, the project was called CARTOON MAYHEM and was primarily a car-based racing game with ancillary gag and weapon features. As we struggled with the demands of Microsoft's vision for IP development, rendering, and weapon effects, we realized that the game engine, which was a patchwork of two years' worth of diverging

demands and evolution, would need a complete overhaul.

For better or for worse, we undertook that overhaul. So it was that just as we were getting into CARTOON MAYHEM's development, our engine, and our ability to iterate content in playable builds, went down for over eight months. This was a crucial time for Xbox and its first-party developers. Microsoft was allocating its resources to those teams with proven track records and those showing steady progress. We were obviously lacking in both areas. Microsoft cut PI, along with our Xbox title, at the end of 2000. Though this was a disheartening development for us, by this time we had the game engine back up and running, and we were suddenly able to produce good demo levels. It wasn't long before we drew interest from several other publishers.

We had a quickly evolving technology and a ton of assets ready to go. The demos we put together enabled us to land a new publishing deal with Electronic Arts. Switching publishers allowed us to prepare some great new material, including an internally developed IP, extra gameplay features, a new renderer, and a new title: CEL DAMAGE. We realized we were going to make the Xbox launch, and we were going to do it with our own property and the backing of the world's largest third-party publisher. These three facts alone made all the work of the previous several years worthwhile.



#### GAME DATA

PUBLISHER: Electronic Arts

NUMBER OF FULL-TIME DEVELOPERS: 16

NUMBER OF CONTRACTORS: 12

ESTIMATED BUDGET: \$2 million

LENGTH OF DEVELOPMENT: 2 years

RELEASE DATE: November 1, 2001

PLATFORM: Microsoft Xbox

DEVELOPMENT HARDWARE USED: 600MHz

Pentium III's with 256MB RAM, 30GB hard drives, and Nvidia GeForce cards

DEVELOPMENT SOFTWARE USED: Microsoft Visual Studio, 3DS Max, Photoshop, Illustrator, Winamp, SourceSafe

NOTABLE TECHNOLOGIES: pitaSim, Vtune, Microsoft Visual C++  
PROJECT SIZE: 800,000 lines of code

## What Went Right

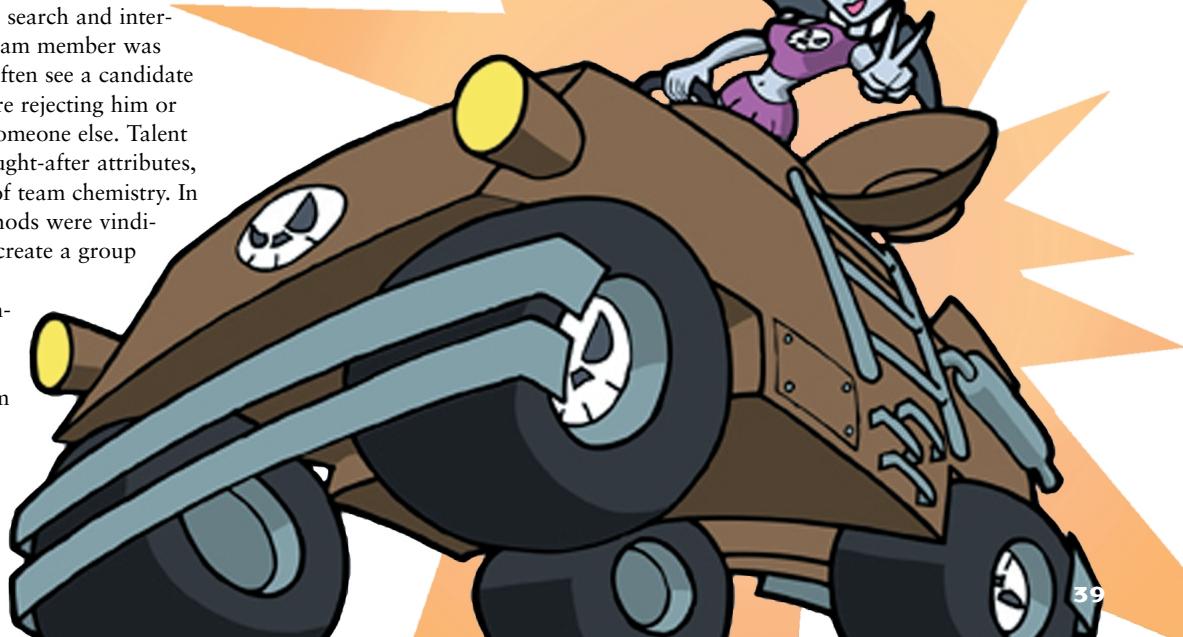
**1. Staffing.** Two years ago, when we started work on our Xbox title, we had a core group of about eight people. It was apparent that if we wanted to develop a console game, whole cloth, in time for the Xbox launch, we would need more staff in every department. We hired more team members as we progressed through development. We were fortunate in that we were able to find very talented and motivated people who were also able to contribute to our corporate élan. We brought our staff in from all over North America, and although none of us had console development experience, each new member brought a rich skill set to the company. The search and interview process for each team member was exhaustive. We would often see a candidate two or three times before rejecting him or her and moving on to someone else. Talent and experience were sought-after attributes, but not at the expense of team chemistry. In the end, our hiring methods were vindicated. We were able to create a group of friends who enjoyed working with one another and were deeply devoted to the project.

Our approach to team communication went hand in hand with our approach to staffing. We found that weekly

full-staff meetings, individual weekly lectures or presentations to the entire staff, and regular departmental reviews greatly improved all team members' understanding of how their co-workers contributed to the project.

We also held an ace up our sleeve. We formed a strategic alliance with a local technical college that offered a diploma course in 3D visual arts. Through the school, we instituted an internship program in our art department. We integrated top students into our team, which was a very successful exercise that we will maintain during our next project.

Moral: Wait for the cream to rise, then scoop it off the top.



**2.** **Early development with Xbox.** As a new entrant in the highly competitive console market, the Xbox group was looking for game experiences that would make their console stand out. As Microsoft pointed out so often during the Xbox design period, “Great technology does not sell game systems, great games sell game systems.” Picking up on that mantra, we started our development when the console was little more than an optimistic dream championed by a charismatic team of visionaries. Chief among them was their bold Advanced Technology Group manager, Seamus Blackley. We were converts to his ambitious plans for the Xbox. With the promise of a stable, RAM-packed, hard-drive-enabled computational powerhouse, we were confident that we could deliver the breakthrough game experience that Microsoft was seeking.

Our game grew and achieved its focus as the Xbox did the same. Knowing that CEL DAMAGE would be held to standards set by second-generation Playstation 2 titles during the 2001 Christmas buying season, we were spurred on to utilize whatever technology the Xbox team was stuffing into the system. We believe that through this evolving relationship, we’ve managed to create an innovative and highly entertaining title. It’s also worth noting that CEL DAMAGE probably would not exist today were it not for the support and inspiration provided by Seamus and the rest of the Xbox team. They stood up for our project and pushed as hard as any of us to make CEL DAMAGE a reality.

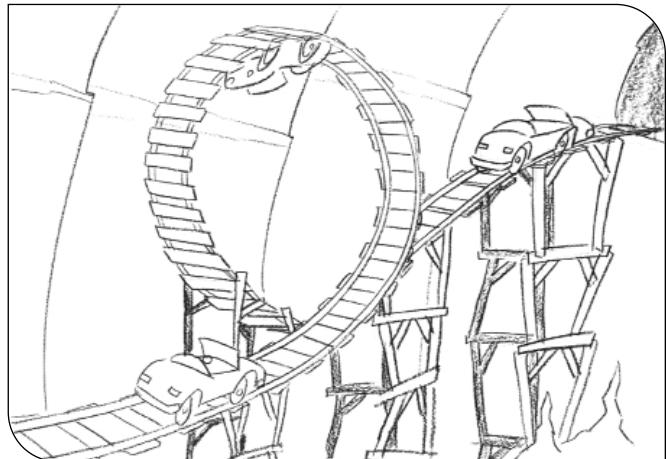
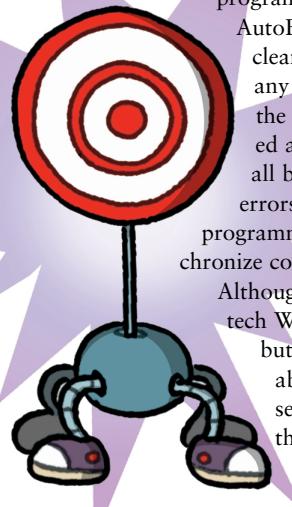
**Moral:** It's all about whom you know.

**3.** **Synchronization tools.** PI grew a great deal over the course of the project, and we knew it was important to keep everyone synchronized. The increasing size of the team, combined with the growing mountain of content and code, made regular updates more difficult and time consuming. The process of creating a build became a black art that only one or two people could do correctly.

The first step toward synchronization came fairly early on with the creation of an automated code-compilation process, dubbed AutoBuild. We investigated a few different automated build programs, but none was as flexible or complete as a home-brewed batch file (or rather, a collection of batch files and supplementary programs). Each night, or whenever necessary,

AutoBuild could check out all source code to a clean directory tree. It then built and executed any code generators, built all binaries, copied the output to a shared directory, and generated an e-mail report containing a .ZIP file of all build output, along with a summary of errors and warnings. Whenever convenient, our programmers could run another batch file to synchronize completely.

Although we implemented AutoBuild with low-tech Windows commands and utilities, this one-button solution proved to be extremely valuable. Each build that the process generated served as the absolute point of reference for the current code base. Even with six people



Early in development, CEL DAMAGE was more race-themed. Here's an early concept for a loop-the-loop road gag in the desert theme.

working simultaneously on the same source code, we were able to keep inconsistencies and problems to a minimum.

Another low-tech solution, MakeBuild, filled our largest gap in synchronization, though its implementation came quite late in the project. MakeBuild consisted of our source game content, automatically compiled into run-time format by adding a few simple commands to the game editor, and a few batch files. By automatically running MakeBuild after AutoBuild, we had a brand-new build waiting for us each morning. Our daily build process kept artists and QA staff up to date without bogging down any individual with responsibility for creating the builds. MakeBuild accelerated the feedback cycle between content creation and gameplay review.

Of course, not all updates were visible in the build, and we made several other utilities to help keep everyone abreast of changes under the hood. Our CheckInReporter was a simple Visual Basic program that scanned the SourceSafe database for all check-ins over the previous 24 hours, and then created and e-mailed out an Excel spreadsheet report. These were especially helpful in tracking down regressions. We created another simple VB program that e-mailed out active bug lists to each team member once per day.

**Moral:** Spending a few days creating simple tools pays big dividends throughout the project.

**4.** **Internal bug tracking and QA.** We made sure that our daily build process was up and running before we built our internal QA department. The daily build mentality was instrumental in the iterative process and was QA's greatest ally. New art and game logic assets could be evaluated in-game within 24 hours of their creation, allowing broken assets and bad functionality to be identified immediately.

Asset pound-downs and targeted focus testing ran concurrently as soon as we had four functional game levels. As development progressed, focus testing generated reams of data, which was boiled down to nearly 400 gameplay and asset recommendations. This information provided an important perspective on what people

were interpreting as fun and fair. This feedback was very valuable, since we'd lost all objectivity toward the game and its difficulty level once we'd mastered the various weapons and gags.

The bug-tracking software that our internal QA used for the duration of the project was called PI\_Raid. This tool, designed and customized in-house, allowed us to stay on top of game defects, generate work items, and comment on evolving game features. We kept our bugs small and focused. While this approach often left each of us with a lot of bugs in our "bin," we were able to close out several per day, providing mini morale boosts throughout the project. Though some of the bugs that we logged might have been considered trivial, cumulatively tackling them had a dramatic, positive effect on the game and our level of polish.

**Moral:** Get fresh eyeballs on your game and efficiently iterate gameplay.

**5. Coordinated schedule.** One of the pleasures of working on CEL DAMAGE was the lack of a brutal crunch period in the final weeks of development. We also felt throughout the last year of the project that we'd be able to realize our desired feature set. A good schedule, coordinated with each department, helped us achieve this unique state. Our early work with Microsoft taught us the value of adhering to a schedule, and after we moved on from that relationship, we were able to maintain, and even improve, our scheduling skills. Our guess is that badly maintained



A before and after shot showing cel-shaded smooth groups on Violet's APC.

and poorly enforced schedules are the primary cause of game projects missing their ship dates, dropping features, and winding up with morale-busting, project-end crunch periods. Following are some schedule-related factors that worked for us:

**Estimating task duration.** No one can estimate with 100 percent accuracy. However, our leads and staff communicated constantly to refine delivery date estimates. If an asset looked as though it was going to run overtime, we would cut it or some of that person's later deliverables, from the schedule. If such changes created holes in the game's design, we would be flexible and design around the holes.

**Software.** We used Microsoft Project. If you've used it, you know it's not great, but it gets the job done. That was all we needed. Once we got used to Project's idiosyncrasies, it was smooth sailing to the end of development.



A before and after shot using the desert theme's General Store. Cel-shading conventions were prototyped in 3DS Max using the Illustrate! plug-in.

**Team-wide involvement.** We periodically printed the master schedule and posted it on a wall where everyone could see it. This helped in many ways. First, it demonstrated the interdependence of the departments. Each staff member could see that an asset he or she was working on was needed by someone in another department. Second, missing items could be identified more easily, since more eyes were looking at the schedule. Third, seeing the schedule updated gave people a strong sense of making progress. This progress contributed to team confidence and morale.

**Short, staggered crunches.** We crunched, but we did it early in small, manageable, prescribed intervals, giving us a buffer at



An early concept for a weapon pick-up, using one of the original CARTOON MAYHEM cast members as a scale reference.

the end of the project, after our feature set was complete. People were then freed up to work on visual weapon enhancements and level polish. At the end of the project, the team was playing full- and split-screen CEL DAMAGE during and after business hours. This intense play period helped identify exploits and balance the gameplay. This data wouldn't have been available to us if we had crunched long and hard at the end of the project.

 Moral: The schedule is your friend. Never let friends down.

## What Went Wrong

**1. Design on the fly.** Once we got our technology back online in December 2000, it began evolving very quickly. Feature sets for weapon and death effects, driving behaviors, gag functionality, and animations were growing every day. Because we were designing a game to the technology (rather than the other way around), we were throwing out design documents as quickly as they could be written. Art assets had to be revised, retextured, discarded, and rebuilt from scratch several times. As most readers will know from experience, this is a scenario for feature creep, obsolete tool sets, and blown deadlines.

While we were able to nail down our feature set four months before shipping, our evolving engine did cause other problems. Essentially, our strategic preplanning was stillborn. Every week, we had to revise our perceptions of what the game would really be, which frustrated our attempts to describe the game to prospective publishers at the beginning of 2001. Different staff members had different ideas of what our game would finally end up looking and playing like. Fortunately, once publishers and press played the game for themselves, the core of CEL DAMAGE's identity as a cartoon-based vehicular combat game became self-evident.

**Moral:** It's O.K. to design to an evolving technology, but institute hard cut-off dates for code development and features.

**2. Asset tracking and implementation.** Our initial efforts produced large amounts of art content to show off the Xbox's power. However, evolving performance specs for the Xbox and our game engine, along with a new IP introduced early in 2001, generated several massive content revisions. These revisions were necessary for level geometry, static world objects, gags, skyboxes, cars, characters, weapons — everything. In the worst cases, we saw at least 12 major revisions to individual assets.

While we had an established directory structure for storage at the beginning of the project, new workflows, staff, and management methods precipitated a patchwork of file-naming conventions and tracking methods. Final game meshes were inadvertently overwritten with geometric primitives. We "lost" assets on the server for days at a time. Other tracking problems cropped up as well. A bug in our game engine created duplicate textures that were difficult to hunt down and eliminate. Also, we had a problem with texture revisions that got wiped out on import to the game editor. To compound our headaches, objects were often used in several different levels, but if an optimization was made to one, that change was not automatically propagated through all levels.

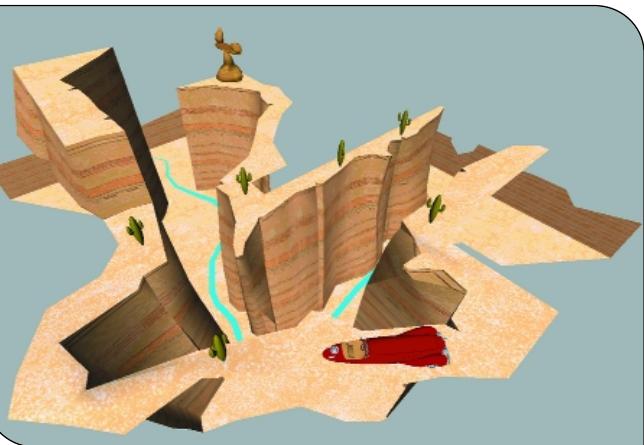
Obviously, we needed a tool to track our art assets and their properties and to update content in the game. We created the robust PI\_Asset for just such a use. Unfortunately, it was introduced too late in the project for full implementation. As a stopgap measure, artists began sending out dailies through e-mail. These reports proved useful in tracking what had been accomplished in the course of a day and what should be updated in the build, but data management was still a problem. PI\_Raid

showed us just how many holes our pipeline had in it. While the primary purpose of PI\_Raid was to track and resolve bugs, the artists and level builders found themselves using it as a means to provide a pathway to updated content. Through PI\_Raid, a person could know when an asset had been updated, where it could be found, and what had changed in it. Using our bug reporter to track game assets was not an ideal solution, but it did serve us well in a pinch.

While the build-discard-rebuild process hit our staff pretty hard, it created a sturdy springboard for future asset-tracking methods, and it also reinforced a better mentality for thoroughness in our development procedures. Our next project will definitely see better tracking and implementation methods.

**Moral:** Don't overwrite finished, textured building models with spheres.

**3. Single member over-tasking.** Due to our relatively small staff, we had to put managers in the critical path of day-to-day asset delivery. These same people held crucial, unique skill sets. As you know, this is a recipe for bottlenecks that hamper development.



Some early desert environment renders made to test scale, detail, and color. Our tests included fog, vertex lighting, and gradations with the goal of evoking a classic Warner Brothers style. These elements were actually dropped as the vision of our own house style came into focus.

One example among several was the role of our art director. We made this person responsible for overseeing the game art, scheduling his staff's workweeks while developing their technical expertise, modeling, creating the game's interface, producing our cutscenes, overseeing interns, and distributing hundreds of art bugs. The time needed for one person to do all these things just wasn't available every day. Fortunately, we were able to create an art lead position to handle the staff and bug tasks. However, not every instance of over-tasking could be fixed by adding a new body.

We had one texture artist, but three or four art staff members were generating meshes. Add to this the frequent discarding of textures and rebuilding of models, and the amount of work crossing the texture desk became enormous. We also had a single staff member who was responsible for updating level content every day. If you consider that on some days we'd generate 100 updated assets, and each one had to be imported, adjusted, and hand-tweaked in the levels, you can gain an appreciation for the bottleneck occurring there. With so many items funneling through one mouse, the balance between efficiency and human error was highly stressed. Ultimately we dealt with the regressions that cropped up, but it's clear that better integration tools for our next project will help a lot.

**Moral:** Spot bottlenecks early and divert the work as necessary.

#### 4. Last-minute implementation of crucial elements.

Our inexperience in console game development caught up with us about three months away from the end of the project. For the better part of two years, we had spent all of our efforts on developing in-game assets and gameplay. As our delivery date to EA came into focus, we realized that we still needed to get a fair bit of content underway, including a solid front-end interface, music, cutscenes, voice acting, foley sound, and sound effects. Once we had our budget in place, we scrambled to pull together a stack of contracted, out-of-house assets.

We drew up a shopping list that looked something like this: 13



cutscene scripts and storyboards, 12 pieces of in-game music, a theme song, interface music, 450 sound effects, 1,000 lines of in-game dialogue and 200 lines of cutscene dialogue to be read by seven different voice actors, six minutes of foley sound and cutscene music, and six man-months of modeling and animation talent. We also realized we needed a way to play back our cutscenes in real time through the game engine and renderer, even though these code elements were not designed to handle the task.

We took on the interface and playback tasks in-house, but farmed out everything else. Obviously, the work was finished on time, but to accomplish this we had to divert the attention of all of our in-house managers to get these items implemented. Spillover bottlenecking was unavoidable. Though we were ironing out implementation bugs until the day we shipped, the quality of the talent and assets we were able to find on short notice shone through in the finished product.

**Moral:** The last 5 percent of a game takes 50 percent of the effort.

5. Switching publishers. As we already mentioned, we switched to a new publisher halfway through development. Going from Microsoft to EA was a mixed blessing. While we were able to improve gameplay and develop our own IP, we lost both our financial backing and our internal focus at a crucial time in the project. We were also forced to reinvent all of our art assets to avoid an IP conflict with Microsoft. However, what could have been a project-wide meltdown actually hardened our resolve to get CEL DAMAGE on store shelves. Once we realized that the CEL DAMAGE property would belong to us, and that our mistakes and successes would be our own, the training wheels came off. We became more determined and professional. As a rite of passage, this publishing switch might have been exactly what we needed. In the end, perseverance carried the day, and getting dropped as a first-party title was a black eye from which we recovered.

**Moral:** When life gives you lemons, start drinking hard lemonade.

#### Damage Control

Now that CEL DAMAGE is out the door, PI's last monkey is off its back. We are published and moving ahead. There is plenty for us to look forward to now, not the least of which is CEL DAMAGE 2, which we will deliver for next holiday season. We are excited about the prospects for Xbox and hope to continue to exploit its strengths with network and team-based play in our next game. Fortunately, our experiences with CEL DAMAGE have shown us where we can improve our processes and strategic planning on our next venture. ☺



Fowl Mouth and Sinder get down to business in a CEL DAMAGE promotional scene created for EA.

# Learning Curves: The Present and Future of Game Studies

**"An academic program of study officially listed as focusing on gaming studies runs, I think, the strong risk of attracting people on the basis of prurient interest. I do not think we should send forth messages of this type if we wish to be a research university of the highest level of distinction."**

— academic official as quoted in "Gaming: Too Cool for School?"  
by Katie Dean (Wired News, January 15, 2001)

**G**ames have taken their place in the entertainment pantheon alongside film and television in a fraction of the time of their forebears. Nonetheless, from a perceptual standpoint, there still remains the notion that games are violent, primarily geared towards teenage boys, at best a waste of time, and at worst a breeding ground for high school snipers. Stereotypes tend to overshadow the facts: games are becoming more and more mainstream and reaching a broader audience than ever. But as the preceding quotation and definition demonstrate, prejudices continue to dominate in academia.

In many respects the controversy comes down to a simple generation gap. The primary audience for computer games is college students. But their professors (generally in their forties and fifties), tend to see games as a cultural scourge and an unwanted distraction from homework.

In spite of the fact that our colleagues find it profoundly distasteful, a small cadre of game scholars has been at large, propagating the new field of "gaming studies." Most of us have designed games, either in commercial-, art-, or research-based contexts. We tend to be media and cultural theorists, computer scientists, artists, or all of the above. Most of us look at game making as a model for the multi-disciplinary convergence of art, technology, and culture.

There are a range of challenges faced in creating a meaningful movement in game studies, and I want to take this opportunity to enlist anyone reading this, from either side of the fence, to help move some of these agendas forward as they see fit.

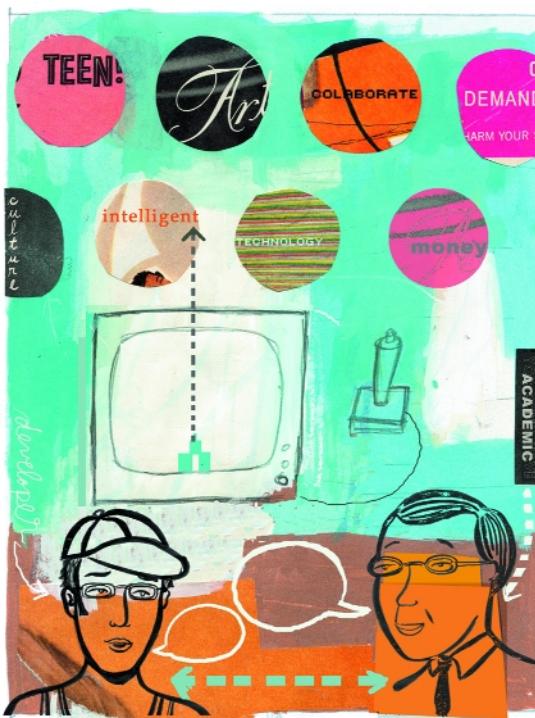
The first and foremost challenge that we face is the culture gap. While it's true that most academics frown on game culture, much of the game industry doesn't seem to hold academia in very high

regard either. The game industry is decidedly unacademic. Very few experienced game designers have advanced degrees. Most game companies look with suspicion upon a candidate with an MFA. But consider this: One of the biggest evolutionary leaps in the art of film was its introduction into the university curriculum. This industry has a lot to gain from supporting a well-developed and well-rounded game studies curriculum.

The second challenge has to do with the content of this curriculum. The best game studies programs try to merge theory and practice. We want to train people to be good game designers by industry standards, but we don't want our curriculum dictated by industry. Universities are not in the business of vocational training. We want to introduce a deep level of understanding, cultural critique, research, and discourse, a big-picture approach that puts game design in the larger historical and cultural context.

The good news is that people in the game industry are a lot smarter than people in the film and television industries. For one thing, game people understand the value of research. Research is vital to both the game industry and to academia, and is thus a fertile area for collaboration. Randy Pausch's students at Carnegie Mellon did the play-testing for DISNEYQUEST, Disney's location-based entertainment product. Both industry and academia agree that play-testing

is one of the best ways to learn about game design. (Many game designers, including myself, got their start in play-testing.) Plus, Disney got what it needed by donating to the university a fraction of what it would have spent on in-house testing. Last summer, Will Wright had some of Randy's interns doing research on play patterns for THE SIMS. These sorts of relationships are mutually beneficial for everyone.



*continued from page 56*

Applied research is another fertile area for collaboration. LEGO MINDSTORMS, for example, was developed originally at the MIT Media Lab with support from Lego. Researchers (and their students) such as Ken Perlin at New York University, Michael Mateas at Carnegie Mellon, and Bruce Blumberg at the MIT Media Lab have been developing ways to make more compelling autonomous characters and interactive narrative scenarios. Academic research is of huge benefit to industry because it's done without the pressure of economic justification. This freedom enables researchers to explore aspects of character and drama that might not otherwise be investigated when immediate profit is a pressing objective. There is a great deal of potential synergy in that equation that can be exploited by both parties.

In the end the controversy will be resolved by two things: demand and money. The demand for game studies is growing. For many young, smart, creative kids, game design has supplanted filmmaking as a cool career. Even art students are starting to treat games in the same way that experimental artists viewed video in

the 1970s and 1980s. The September 2001 issue of *Next Generation* magazine featured a cover story called "Video Games 101" with a guide to all the current game design programs. This kind of coverage is about as mainstream as it gets. As to the money part, well this is where you come in. At universities, just like anywhere else, money talks. If game companies come forth and support us, then all the arguments will quickly become moot. I anticipate that this will happen relatively quickly. When it does, the rest of the world will have no choice but to bestow game studies with the respect that it deserves. ☺

**CELIA PEARCE |** Celia is an interactive multimedia designer, artist, researcher, teacher, and author of *The Interactive Book: A Guide to the Interactive Revolution* (Macmillan). She is currently a lecturer at the Claire Trevor School of the Arts at the University of California, Irvine, and acts as liaison between the game industry and academia for the Education Committee of the International Game Developers Association ([www.igda.org/Committees/education.htm](http://www.igda.org/Committees/education.htm)).