



GAME DEVELOPER MAGAZINE

NOVEMBER 2003





GAME PLAN

LETTER FROM THE EDITOR

Fixing a Leaky Valve

It's like that archetypal dream where you're taking a final exam for which you're not prepared. And then, as if things couldn't get any worse, you suddenly realize that you're sitting in class in front of everyone you know, stark naked.

Or at least, that's a pale shade of how I imagine the developers at Valve felt when they realized the HALF-LIFE 2 source code had found its way into the wrong hands — many, many wrong hands. As I write this the hacker's trail is still warm, and both Valve and Havok (whose physics code was also reportedly part of the stolen source) declined to offer me any official comments on the situation beyond Valve managing director Gabe Newell's posts to the www.HalfLife2.net forums, where he first confirmed the hack. Havok remains in wait-and-see mode.

Developers may grudgingly accept that cracked or leaked betas will appear online while they are still fixing bugs and tuning gameplay, but when news of the HALF-LIFE 2 source heist broke, it was hard not to feel the collective shudder wriggle up the spines of game developers the world over.

Of course no professional would lift chunks of Valve's code wholesale for use in a commercial product, but is there harm in just looking, poring over five years' worth of innovations and optimizations by some of the best in the business? Well, if you find a wallet on the street with \$5,000 in it and no ID, do you say, "Hmmm, act of God!" and go buy a mammoth new plasma HDTV while blessing your good fortune, or do you turn it into the police with the contents intact? Most of us like to think of ourselves as the kind of honest person who would turn in the money.

But what if you somehow knew that the wallet had been stolen ten minutes before it had been lost, and that the thief would claim it before the rightful owner? Is it still wrong to keep the money if you know the rightful owner

will never get it back anyway? It's another case of two wrongs don't make a right. Just because something is in your possession doesn't mean it actually belongs to you.

Don't download the HALF-LIFE 2 source. Don't keep it if you already have it. Don't condone others who do. Don't create the lame excuse that "everyone else is doing it" in the name of staying competitive. Valve's developers have already given more than most to the game development community, by speaking at conferences, writing articles, and posting to mailing lists on a variety of topics surrounding their work. Don't take more than they were already willing to give.

Welcome, Steve. Finally, and not on an entirely unrelated note, this month I am excited to introduce our new Artist's View columnist, Steve Theodore, whose career includes many years at Valve, where he worked on HALF-LIFE, TEAM FORTRESS CLASSIC, COUNTER-STRIKE, and TEAM FORTRESS 2. I have had the pleasure of knowing Steve for several years now, and you may have seen his name before in *Game Developer* next to various product reviews and last year's feature on animation blending, or perhaps you've seen him speak at GDC on one of his many areas of expertise.

As game art is becoming increasingly specialized, Steve's as close to a jack-of-all-trades that could be expected of a single human these days, and it's a privilege to have him batting for our team. That his fluency in Latin and Greek is almost on par with mine is just so much satin piping on the jersey. Steve takes on procedural textures starting on page 22. I hope you enjoy getting to know him over the next few months; he's got a lot in store for you.

Jennifer Olsen
Editor-In-Chief

GameDeveloper

www.gdmag.com
600 Harrison Street, San Francisco, CA 94107 t: 415.947.6000 f: 415.947.6090

EDITORIAL

Editor-In-Chief

Jennifer Olsen jolsen@cmp.com

Managing Editor

Everard Strong estrong@cmp.com

Departments Editor

Jamil Moledina jmoledina@cmp.com

Product Review Editor

Peter Sheerin psheerin@cmp.com

Art Director

Audrey Welch awelch@cmp.com

Editor-At-Large

Chris Hecker checker@d6.com

Contributing Editors

Jonathan Blow jon@number-none.com

Noah Falstein noah@theinspiracy.com

Steve Theodore steve@theodox.com

Advisory Board

Hal Barwood Independent

Ellen Guon Beeman Monolith

Andy Gavin Naughty Dog

Joby Otero Luxoflux

Dave Pottinger Ensemble Studios

George Sanger Big Fat Inc.

Harvey Smith Ion Storm

Paul Steed Microsoft

ADVERTISING SALES

Director of Sales/Associate Publisher

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

Senior Account Manager, Eastern Region & Europe

Afton Thatcher e: athatcher@cmp.com t: 828.350.9392

Account Manager, Northern California & Midwest

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

Account Manager, Western Region & Asia

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

Account Representative, Target Pavilion, Education, & Recruitment

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

ADVERTISING PRODUCTION

Advertising Production Coordinator

Kevin Chanel

Reprints

Terry Wilmot t: 516.562.7081

GAMA NETWORK MARKETING

Senior MarCom Manager

Jennifer McLean

Marketing Coordinator

Scott Lyon

CIRCULATION



Game Developer
is BPA approved

Circulation Director

Kevin Regan

Circulation Manager

Peter Birmingham

Asst. Circulation Manager

Lisa Oddo

Circulation Coordinator

Jessica Ward

SUBSCRIPTION SERVICES

For information, order questions, and address changes

t: 800.250.2429 or 847.647.5928 f: 847.647.5972

e: gamedeveloper@halldata.com

INTERNATIONAL LICENSING INFORMATION

Mario Salinas

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

EDITORIAL FEEDBACK

editors@gdmag.com

CMP MEDIA MANAGEMENT

President & CEO

Gary Marshall

Executive Vice President & CFO

John Day

Executive Vice President & COO

Steve Weitzner

Executive Vice President, Corporate Sales & Marketing

Jeff Patterson

Chief Information Officer

Mike Robert

President, Technology Solutions

Mikros Falettra

President, Healthcare Media

Vicki Masseria

Senior Vice President, Operations

Bill Amstutz

Senior Vice President, Human Resources

Leah Landro

Vice President & General Counsel

Sandra Grayson

Vice President, Group Publisher Applied Technologies

Philip Chapnick

Vice President, InformationWeek Media Network

Michael Friedenber

Vice President, Group Publisher Electronics

Paul Miller

Vice President, Group Publisher Software Development

Peter Western

Corporate Director, Audience Development

Shannon Aronson

Corporate Director, Audience Development

Michael Zane

Corporate Director, Publishing Services

Marie Myers



GamaNetwork

SAYS YOU

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

Diagnosing Addiction

In reference to the parents whose 22-year-old son seems addicted to EVERQUEST (“Says You,” October 2003), their son is not responding to the world in a normal fashion. There are a number of things that can cause addictive behavior, but an all too common one is a mental disease known as “affective disorder” to clinicians, or “clinical depression” to the layperson.

If this is his problem, then the issue is not with the game, it’s an organic imbalance in brain chemistry. If it weren’t this game it might be booze, drugs, sex, or anything else that would provide a respite from his inner turmoil. This is a potentially fatal disease as one of its documented symptoms is suicide.

By coincidence, I myself was diagnosed with this disease at age 22 and, with the help of a few pills each night, I have since gone on to graduate from college and have a highly successful 15 years in the technology industry as a prominent, respected, and professionally published software engineer.

Since these parents are concerned about their son’s future, I would implore them to stop looking for outside things to blame and get him to a good psychiatrist who can diagnose or rule out internal problems. If he is a depressive, then we aren’t just talking about his future, but potentially his very life.

Jeff K.
Santa Clara, CA

Accentuate the Negatives

Jonathan Blow’s “Using an Arithmetic Coder” articles (August and September 2003) were fantastic. I’m looking at trying to incorporate this idea into our title.

I wondered about the `value` and `limit` being integers. I am hoping to find a general solution that would work for almost any kind of number, and I



believe negative numbers would be problematic with the current solution. Would it be a better idea to offset my data to be all positive and take that into account by the unpacking caller? Or should I simply encode the sign bit as another bit and decode it on unpack automatically? Or is this technique only applicable in the specialized situation you described in your article?

Brett Bibby
GameBrains (via e-mail)

Jonathan Blow replies: *Actually, I don't cover the negative number thing in any of the articles, but maybe I should have. You're right that you can't just shove a negative number in there. What I do is, I know the minimum and maximum values for a number, and I offset them by the minimum. So, $value_{prime} = value - minimum$, $limit = maximum - minimum$, and that's what I put in the encoder. At decode time, you do the same thing in reverse.*

You can define a new function that does this so you don't have to think about it. I wanted to keep the article and the example code as simple as possible, so I left this out. You could also do something with a sign bit, but it would be less efficient.

A Distcc Second

I am writing regarding Justin Lloyd’s review of Incredibuild (March 2003). I agree with everything he said about the product. However, Justin writes that Incredibuild is only available for VC++ 6.0 and .NET, thus leaving out platforms such as Gamecube and Playstation 2, and more generally any project that does not (or cannot) make use of VC++.

I must point out the existence of distcc, a free GPL tool that works over gcc to allow for distributed builds. It functions similarly to Incredibuild in that it does not require distributed makefiles or any complex additions to the build process. You run a daemon on every machine that is to participate in builds and call distcc in place of gcc. Your machine will preprocess source files and send them out to individual hosts, specified in a local configuration file, for compilation.

Alban Wood
via e-mail

CORRECTIONS

In the October 2003 feature “Structuring Creativity,” co-author Wolfgang Hamann’s bio was incomplete. He also worked on THE HULK, SIMPSONS ROAD RAGE, and most recently, SIMPSONS HIT AND RUN. He was previously a record producer and personal manager in the music industry.

Also, in the October 2003 review of the NXN Alienbrain Integrator SDK, we overstated the cost of the minimum configuration due to a recent change in pricing. The clients now come with the server software, and are sold à la carte. The minimum price for a usable configuration is the sum of two \$690 developer clients, a total of \$1,380.



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



INDUSTRY WATCH

KEEPING AN EYE ON THE GAME BIZ | *jamil moledina*

Take-Two makes TDK an offer it can't refuse. After posting a third fiscal quarter profit of \$7.7 million, Take-Two announced that it will buy TDK Mediactive for \$22.7 million in stock. Take-Two's acquisition will give the GRAND THEFT AUTO publisher access to properties such as SHREK, PIRATES OF THE CARIBBEAN, and ROBOTECH.

Concerto for videogames. Led by conductor Andy Brick, the Czech National Orchestra performed videogame scores to a sold-out audience in Leipzig, Germany. The fall game music concert kicked off the GC Games Convention, marking the first such event outside of Japan. The concert featured music from Nobuo Uematsu (FINAL FANTASY), Olof Gustafsson (BATTLEFIELD 1942), Koji Kondo (THE LEGEND OF ZELDA: THE WIND WAKER), Andrew Barnabas (PRIMAL), Christopher Lennertz (MEDAL OF HONOR: RISING SUN), and Chance Thomas (QUEST FOR GLORY V: DRAGON FIRE).

Eidos raids lost profit, whips Gamecube. Eidos Interactive reported a £17.4 million (\$27.4 million) pre-tax profit for the year ending June 30, 2003, compared to a £15.3 million (\$24.3 million) loss the previous year. The company was buoyed by




A videogame music concert performed by the Czech National Orchestra at GC Games Convention.

the early release of TOMB RAIDER: THE ANGEL OF DARKNESS. Eidos CEO Mike McGarvey also stated that the company will no longer release titles for Nintendo's Gamecube, claiming that the console is "a declining business." A few days later, Atari also announced that it is canceling three Gamecube titles.

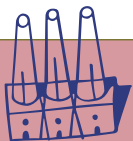
Over 150 million served. Nintendo announced that it has sold 15 million Game Boy Advance units in the U.S. since its release in 2001, and 150 million Game Boy units worldwide since the brand's introduction in 1989.

Playstation 2 levels up with Hard Disk Drive. Sony plans a March 2004 release for its Playstation 2 Hard Disk Drive, which will feature a 40GB drive; a media manager for games, MP3s, CDs, and photos; and Square Enix's

MMORPG, FINAL FANTASY XI. The \$99 internal peripheral will connect to the required Network Adaptor, enabling online gameplay and content downloads. Sony also announced that it has sold more than 60 million Playstation 2 consoles worldwide since its release in 2000.

THX is listening. THX launched a sound and visual certification standard for videogames, with EA Games as its first recipient. The certification program includes game mixing room analysis and standardization, and the ability for developers to test their work as they create it. THX director of advanced technology Mark Tuffy said that THX's goal is to "let the gamer see and hear what the game artist intended." 

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



THE TOOLBOX DEVELOPMENT SOFTWARE, HARDWARE, AND OTHER STUFF

ATI unveils final Rendermonkey. ATI released version 1.0 of Rendermonkey, an open, extensible utility for creating real-time programmable graphic shader effects. Developed in conjunction with 3DLabs, Rendermonkey 1.0 supports the DirectX 9.0 HLSL shading language. It has a real-time preview window that shows the current state of the shader, reflecting any code changes immediately. This final version also includes the Rendermonkey API. Rendermonkey 1.0 is available as a free download. www.ati.com

Araxis simplifies code review. Araxis announced the availability of Merge v6.5, the latest revision of its visual file comparison and merging utility for keeping multiple folder hierarchies synchronized. The utility compares text files in ASCII, Unicode, and MBCS character sets and produces detailed reports. New features include HTML reports of file comparisons, improved regular expression support, and a new Virtual File System API. Merge v6.5 is available for \$129-\$219. www.araxis.com



UPCOMING EVENTS CALENDAR

**STATE OF PLAY: LAW, GAMES,
AND VIRTUAL WORLDS**
NEW YORK LAW SCHOOL
New York, N.Y.
November 13-15, 2003
Cost: \$125
www.nyls.edu/pages/777.asp

**AUSTRALIAN GAME
DEVELOPERS CONFERENCE**
MELBOURNE CONVENTION CENTRE
Melbourne, Australia
November 20-23, 2003
Cost: AUD\$180-AUD\$800
www.agdc.com.au



BREW vs. J2ME: Round 2

by ralph barbagallo

MIDP 2.0

Sun has added many new features to MIDP 2.0, its first major update to the J2ME CLDC/MIDP in some time. These include new GUI components, a Mobile Media API for standardized sound playback, key polling for multiple-key-press, socket networking, push architecture, and some far-reaching new graphics classes in the form of the Game API.

The Game API. The Game API consists of a set of classes that govern the drawing of sprites and tiles. There's also a new `GameCanvas` object that can be used to poll keys for multiple-keypress. In the case of sprites and tiles, they are all derived from the new `Layer` object. Derived from `Layer`, the `Sprite` class features the ability to store multiple rectangle frames of a sprite in a strip or rectangle bitmap layout. The `Sprite` also performs bounding-box or pixel-accurate collision against bitmap `Image` classes, tiles, or other `Sprites`. The `Sprite` object also has support for animation in the form of frame sequences that define a list of frames to display in linear order. You can cycle and loop through the frame sequence before painting the `Sprite` for an animation effect.

The `TiledLayer` class is derived from `Layer` as well. This takes bitmaps in the same strip or square layout as the `Sprite`, but this time they are used as individual tiles in a tile map. The tile map is a series of "cells" inside the `TiledLayer` that, annoyingly, must be set on an individual basis. The `TiledLayer` also supports animation in an easy to use manner. Both `Sprites` and `TiledLayers` can be used with the `LayerManager` object to automatically manage the draw order of your bitmaps.

A lot has happened since last year's BREW vs. J2ME showdown ("Product Reviews," May 2002). Markets have matured, handsets have come and gone, and a real mobile-game business model has solidified. Both technologies have undergone extensive enhancements in the area of game development APIs. In the case of J2ME, Sun has added their Game API in MIDP 2.0, allowing for native support of sprites and tiles. BREW 2.0 has introduced many new advances, including a similar sprite and tile architecture. This review will compare the two, as well as highlight the major enhancements of each platform.

BREW 2.0

BREW 2.0 has been available for well over a year, and although BREW 2.1 is available, 2.0 handsets are just now rolling out in the U.S.

The biggest enhancement is native support for sprites and tiles. Using the new `ISprite` interface, it's possible to draw sprites and tile fields with bitmaps restricted to four sizes: 8×8 , 16×16 , 32×32 , and 64×64 . BREW 2.0 also allows for the transform of bitmaps, allowing you to flip and rotate images (though only in 90-degree increments on the LG VX6000). The newly-introduced `IDIB` and `IBitmap` interfaces replace the old `IImage` and native bitmaps from BREW 1.0. Through these two interfaces one can convert any format to a device-native bitmap, including com-



A screenshot for CALIFORNIA SURF, a game developed with Brew 2.0.

pressed PNG and BCI images.

Real-world performance report. The LG VX6000 — featuring a large, 16-bit color display, a built-in camera, gobs of memory, and 1X 3G network speeds — is the first BREW 2.0 handset being released in the U.S. However, the handset has some less-than-impressive performance characteristics. I found this device to be slower than even Motorola's old 1.1 T720, released last year.

Although BREW 2.0 has some great new APIs for game developers, the performance benefits from using them are nonexistent when compared to the old ways of `bitblt` and native bitmaps, at least on the LG VX6000. But the pixel-level access to the frame buffer will help some applications.

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



Real-world performance report. It's hard to discuss the actual performance characteristics of MIDP 2.0, since at press time, there are no commercial MIDP 2.0 devices on the market, though a few devices may begin trickling out by the time you read this. Sun's Wireless Toolkit emulator runs at a horrifically slow pace, slowing down into the single-digit frame rates when **TiledLayer** is used. (Sun attributes this to a bug, and plans on fixing it.) However, emulators are never a good way to gauge the real-world performance of any mobile API.

The one MIDP 2.0 handset we were able to get hold of for this article sports some impressive performance characteristics. However, it is a much higher-end

PDA-style phone. MIDP 2.0 is designed for a much broader class of devices, this includes mundane consumer handsets and expensive Smartphones. Thus, there really is no comparison to the much cheaper BREW 2.0 devices.

The Verdict

Until we see more MIDP 2.0 devices on the market, it's hard to make a definitive decision. Both APIs compare favorably, as they have the basics in the sprite and tile department as well as other odds and ends. MIDP 2.0 is still much more restrictive given its sandbox security model. However, new code-signing features allow for the use of APIs

which are not bound to these limitations.

Some of the barriers that made BREW an expensive proposition have been removed — Qualcomm has provided support for the free GCC compiler for use with both BREW 1.1 and 2.0. Now, the only costs associated with BREW are the Verisign certificate service and application certification fees. MIDP 2.0 still has a wide variety of free tools, including an updated toolkit from Sun. However, MIDlet distribution remains a mess of interlocking carrier agreements, portals, and so-called “aggregators.”

From a pure development perspective, it can go either way; you either like Java or you don't. BREW has all the pain and heartache of native embedded development, including a lack of on-device debugging, weird alignment issues, and sometimes-severe bugs in just about every phone out there. Java has a deceptively friendly development environment, however once you get up and running on an actual handset you may find that some VMs don't exactly work as advertised.

Once again, there is no clear winner. For small shops, BREW's simplified billing and carrier distribution can't be beat. If you are looking for worldwide market penetration you can't avoid Java. So the war rages on — you'll have to drop by next year to see if there will be a clear leader of the pack.

Corel's Painter 8

by sean wagstaff

Corel's Painter has long been recognized for its great natural-media painting features; brushes and materials that mimic camel hair, ink pens, oils, inks, watercolors, canvas, and the rest of a master painter's toolkit. It has also been criticized for the density and complexity of its unfamiliar interface. Previous versions were downright hard to master for die-hard users of Photoshop and other common painting tools, even while its luscious features offered plenty of allure.

Painter 8 starts on a clean canvas that will make it far friendlier as a production-oriented tool in game studios where there is a strong need to paint artistic textures,

MIDP 2.0 ★★

STATS

Sun Microsystems
Santa Clara, Calif.
800.555.9SUN
java.sun.com/products/midp

PRICE

Free

SYSTEM REQUIREMENTS

Java 2 SDK, Standard Edition (J2SE SDK), version no earlier than 1.4, plus one of the following: Microsoft Windows XP or Microsoft Windows 2000, Microsoft Windows 98/Windows NT (unsupported), Solaris™ 8 (unsupported), Linux (unsupported).

PROS

1. Animated tile system is actually quite novel.
2. Pixel-accurate collisions done for you.
3. Standardized socket and music support (finally).

CONS

1. Not currently on any consumer devices.
2. Sprite and tile architecture can be rather restrictive.
3. J2ME GUI is still terrible.

BREW 2.0 ★★

STATS

QUALCOMM
San Diego, Calif.
858.587.1121
www.qualcomm.com/brew

PRICE

Free, but Verisign certificates are still required

SYSTEM REQUIREMENTS

Windows 2000 or Windows XP, Visual Studio .NET or GCC, Verisign Class 3 Certificate

PROS

1. Actually available on hardware in consumers' hands now.
2. Sprite and tile engine less restrictive — leaving more implementation up to the developer.
3. GCC support now makes BREW development costs less of an issue.

CONS

1. Performance of early 2.0 handsets is equal, if not less than, 1.1 devices.
2. Organizing sprites and tiles into “pages” is rather inconvenient.
3. Despite a camera in the first 2.0 handset (VX6000) — no standardized camera support until 2.1.

backgrounds, and marketing art. Corel has finally embraced the fact that Photoshop has become the industry standard for storing and manipulating multi-layered bitmap image files, making Painter file-compatible with Photoshop, including support for layers, multiple masks, and alpha channels.

Game artists will find great uses for Painter wherever there is a need for paint and other real-world art materials. Texturing building interiors, for example, can be greatly enhanced by the use of brushes that simulate sponges and rollers on stucco, while exteriors can benefit from being able to easily lay down layers of stones, foliage, and other elements into layers of a texture image.

One of the features missing in previous versions was a way to easily mix and blend colors the way all painters do when working with a limited palette of starting colors. However, with Painter 8's new



Painter 8's Mixer palette.

Mixer palette, you can now plop down swatches of any color and then mix them together by stirring and swirling bits of two or more colors; you can then pick from among the resulting shades to get

your own pure brush colors.

Painter 8 ships with more than 400 new brushes, but one of the best Painter features for texture artists is the Image Hose, which lets you paint with bitmapped images, such as photographs of rivets, stones, or reptile scales. These brushes can be set to use multiple images, including alpha transparency, as well as to swap images in response to stylus pressure, stroke direction or other combinations of input. A couple of minutes setting up a custom Image Hose can save hours of painstakingly painting details into a texture or background. However, a feature I'd like to see is the capability to paint different images into different layers for easy creation of bump and specular maps to accompany painted Image Hose textures.

One thing Painter 8 doesn't do is paint with or store a depth alpha along with color information. While paint can react appropriately to an underlying can-

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

vas texture and on multiple layers, paints are not rendered using lights as they are with Deep Paint. On the other hand, Painter offers faster, better manipulation of masks and channels — you can easily create a manipulated grayscale version of a painted texture on a separate layer, for instance.

Painter 8 won't replace Photoshop, but budget developers could get away with Painter as their primary texture-painting tool. For the price, it's a great box full of art tools to have around.

Painter 8 retails for \$299 with upgrades priced around \$149. It is available for the Windows 2000/XP PC platform, and OS9/X for the Mac.

★★★★★ | **Painter 8**
Corel
www.corel.com

Sean Wagstaff is a freelance 3D artist. You can reach him at www.wagstaffs.org.

Line 6 PODxt Pro

by *todd m. fay*

The PODxt Pro is an amp/cab modeling processor that allows an audio producer to make their instruments sound like they're being played through multiple different guitar amps (which all have their own particular sonic characteristics).

The PODxt Pro fits nicely into your studio recording chain, hooking up via its digital or analog outs to your recording interface or mixer, or directly to your computer through its USB port. Your guitar plugs in the front panel into an unbalanced, mono jack, which features a pad for signals hotter than your standard guitar (or those with active pickups). At this point you're given access to an impressive array of digitally modeled amplifiers, cabinets, effects processors, and positioned microphones.

Line 6's Point-to-Point modeling technology offers a wide range of sonic options. They chose the best individual models in an amplifier series down to the year, in order to capture the classic tones we've come to identify entire genres of music with. There are also eight original

models that drive guitar tone in directions that actual tube and solid-state amplifiers cannot.

Having 32 amplifier models and 22 cabinet models at your fingertips means searching for the perfect guitar sound has never been so easy. The PODxt has tons of effects built-in and even a tube preamp model for warming up other sources besides guitars, such as voice, keyboards, and drums. The PODxt also includes 46 stompbox effects, such as Octavia, Univibe, Vibratone, as well as four microphone models. And it's compact, taking up only two rack space units.

This unit has great factory presets, many of which are named for familiar songs with which you'll instantly associate the guitar sound. These presets make getting a particular classic "out of the box" tone simple. All of the unit's parameters are controllable via MIDI. You can change them on-the-fly using a footboard (great for when you're recording and performing alone) or from your sequencer.

The Pro comes with some studio-oriented advantages that a normal PODxt doesn't, such as input/output, indicators, and routing options. These include AES/EBU and S/PDIF connections for keeping your sound in the digital domain. You've also got access to two line-level inputs in the back, which are great for taking signal from a keyboard or mixer for feeding clean recorded guitar tracks through the POD for "re-amping."

Introducing a PODxt Pro into your game audio production studio grants access to just about any guitar tone that you could possibly want in one unit. Listed at \$979 retail, it's extremely cost-effective and the sound is incredibly accurate. With it, audio teams will be able to whip up authentic-sounding metal mayhem, smokey blues, or film-noir jazz (à la Peter McConnell's GRIM FANDANGO soundtrack). 🎸

★★★★★ | **PODxt Pro**
Line 6
www.line6.com

Todd is a producer and consultant in the music, games, and TV industries. Contact him at todd@audiogang.org.

Bright Lights, Big Payoffs

Leading Edge Design's Larry DeMar

Larry DeMar's involvement with videogame entertainment can be traced back to his 21st birthday in 1978, when he received an Atari 2600 system. After college graduation, he got a job offer from Chicago-based Williams where he got to work alongside such notables as Steve Ritchie, Eugene Jarvis, and Ken Fedesna on classic titles that include DEFENDER, its follow-up STARGATE, ROBOTRON: 2084, and BLASTER.

Sensing a lull in the market, Larry switched his energies to his other passion, pinball, and helped design and create such titles as BANZAI RUN, HIGH SPEED, and one of the all-time best-selling pinball games, THE ADDAMS FAMILY.

In 1999, with the pinball field declining in popularity and having been turned on to video-based casino slot machines, DeMar struck out on his own, forming Leading Edge Design (www.ledgaming.com). The company just released its award-winning MULTI-STRIKE POKER. Why would someone tackle the video-based casino market? Try hundreds of thousands — or even millions — of dollars in revenues.

How has Larry tied in his experiences with videogames, pinball, and casino gambling design together? Read on to find out.

Game Developer: How many lines of code are in ROBOTRON: 2084?

Larry DeMar: I have ROBOTRON: 2084's source code printed on a 3-inch-thick ream of standard foldout paper. It contains roughly 15,000 lines of 6809 assembly code.

GD: In today's videogame market, where game goals and missions are usually so well-defined, what makes a game like ROBOTRON still so addictive?

LD: I agree that the game is more tactical as opposed to the deeper strategies in play today, but ROBOTRON combined these tactics with overall play strategies. The things that keep the players coming back are adrenaline and accomplishment.

GD: What design principles carry over from videogames to pinball design to gambling games?

LD: In both pinball and videogames you want to have a graspable play mechanic with rules that can be picked up in a few plays. The most important things are for the player to know what his mission is and to provide ways to measure progress toward that mission. In the most successful games, these metrics are passively passed to the player through the presentation.

In slot machine design, the first and foremost foundation of a successful game is the math and the "gambling ride." A key factor is the volatility of the game, which measures the wildness of



Leading Edge Design's Larry DeMar has a lot to be happy about.

swings on both the winning and losing side.

GD: How do you view the reward system as it relates to videogames, pinball, and slot machine design?

LD: I don't think that rewards are the key to pinball or videogames. The most successful games define a mission and give you an understanding of that mission, the tools to complete that mission, and most importantly, good feedback about your progress. Rewards are just one form of this feedback. In slot machines, you are creating a roller-coaster of ups and downs in a player's cash balance. Rewards are king [for slot machines]. A successful game combines this with an interesting play mechanic and attention-grabbing presentation features.

GD: What benchmark do you use to consider a game successful?

LD: The most fun way to play the game should always be the most rewarding. If there are different fun approaches to play the game — whether it be a videogame, pinball, or a slot machine — then the reward of each approach should be balanced by the relative risk. It's important that there not be ways to "rape" the game, which are typically caused by a badly balanced (and usually badly thought-out) way to get a higher score with very low risk.

I think that ROBOTRON's continued popularity, 20 years after its creation, is due to a near perfect balance, among other things.

GD: What feedback do you look for from your games?

LD: When we watch a game's first test or trial, I am less interested in how many people approach the game and more interested in seeing how many of them play it again right away.

GD: You have said that "there's one great game in there, but 99 bad ones." What do you mean by that?

LD: There is not a more important aspect to game design than this principle. The best designers understand it well. Other designers and most management personnel usually don't get it.

Given a broad definition for a game (for example, "Let's do a scrolling space game where you track the enemies on a scanner and protect humans on the surface of a planet") there are an infinite number of ways to construct the game rules and mechanics. Most of them will be truly awful. Another large quantity may be as good as mediocre. Only a very few will be from good to great. The best designers can figure out how to develop and balance in this range with some type of consistency (I would consider 30 percent or higher to be excellent). The rest of the pack will hit from time to time, if at all. 🍀

Piecewise Linear Data Models

In a high-performance networked game, we want to compress our data to make efficient use of bandwidth. In previous articles I have discussed the use of an arithmetic coder for compression (August and September 2003). Last month I showed how adaptive compression could be achieved in the face of unreliable communications, as when using UDP (October 2003). However, these articles all assumed that the transmitted data would be built from sets of discrete symbols, like the alphabets used for text compression.

In a modern game, much of the data we're transmitting will not be alphabet-like. An alphabet has the property that neighboring symbols are distinct and largely unrelated. So if you are transmitting compressed text and the next symbol is "R," that means it is not anything like "S." This influences the data structure design for these systems; to store symbol probabilities, we tend to use arrays, with a separate and independent probability value per symbol. This data structure design then influences the final algorithm design.

I used the alphabet paradigm in earlier articles for two reasons. First, it helps to keep things simple; I was able to focus on the issues at hand, without juggling too many new concepts. Second, most of the arithmetic coder-related information you'll find on the Internet assumes the data is built from alphabets; so there's an accord between those articles and other sources the interested reader might seek.

This month, I want to step away from that alphabet paradigm. In games we often manipulate values that are ideally continuous, or values where neighboring elements possess strong semantic similarities. In an earlier article I called these "coherent values." Some examples of coherent values are a game character's health or his position in the world. Sometimes these types of values have very high resolutions; building probability arrays for them, as one does with alphabets, would waste a lot of memory. We want a probability representation (data model) that exploits the data's coherency. If we represent the data model as a smooth continuous function, we can use a piecewise linear approximation of that function as an efficient way to store and compute probabilities, hence the title of this article, "Piecewise Linear Data Models."

Health

I'll now look more closely at the example of player health. I'll discuss this in the context of a persistent-world online game, though the basic ideas apply to many types of games.

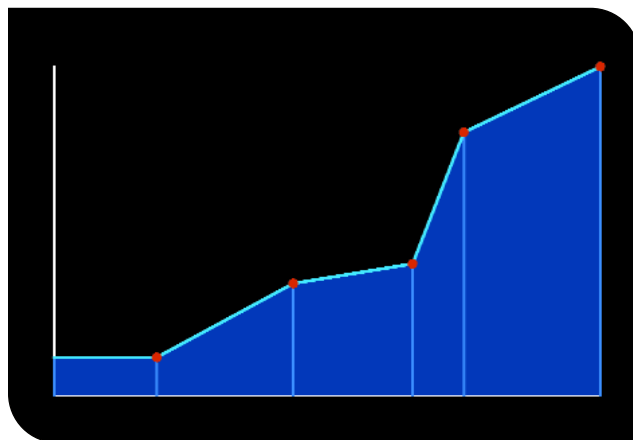


FIGURE 1. A coarse piecewise linear approximation of a probability distribution function for player health. The X-axis represents health values; the Y-axis represents the relative probability of each value.

Suppose we store a player's health as a floating-point value between 0 and 1; 0 indicates that the player's dead, 1 indicates perfect health. For the purposes of network transmission, we quantize this floating-point number into an integer at an arbitrary resolution (see "Packing Integers," May 2002). We could now transmit this integer directly, but we would be ignoring the fact that some health values are more likely than others; thus we would achieve poor compression.

The best way to figure out your data probabilities is to measure them from actual gameplay; but since we're talking about a hypothetical game, let's speculate about health. Players want to feel safe, so if they have the ability to rest or be healed in some other way, they will do so when their health is low and they're not preoccupied (with fighting, for example). So if a player's health is low, it will likely not be low for very long. In fact, once health falls below a certain point, chances are the player's in an overly difficult combat session and may be killed soon. Thus we have what statisticians call a "survivorship bias" that drives the average health value even higher. (Assume that players who are



JONATHAN BLOW | Jonathan is hanging out in Baltimore, but he hasn't eaten any crab. Send admonishments to jon@number-none.com.

killed will lie on the ground for a short period of time and then be reincarnated in a faraway safe area). Given this, we may expect the function for health values to look like Figure 1. This figure depicts a piecewise linear approximation; the function is divided into a small number of segments, and the value of the function is determined by linearly interpolating the control points. Control points are defined at segment boundaries.

Recall that to pack a number into an arithmetic coder we need to define an interval of $[0, 1]$, which means we need two things, the interval's size and its starting point. The interval size corresponds directly to the probability of the value we are packing. The starting point tells us which actual value it is; essentially it's just the ending point of the previous interval. (See "Using an Arithmetic Coder," August and September 2003).

Given a piecewise linear function like Figure 1, the appropriate interval width is just the function's y value at the desired x coordinate, divided by the total area covered by the function graph. (Dividing by the area just normalizes the values to make sense in $[0, 1]$). The interval's starting point is the total area of the function to the left of that x coordinate, divided by the area of the entire function (see Figure 2).

How the Encoding Math Works

So that we can successfully decode our data, the math we use to encode must be precise and reversible. Thus we store our piecewise function in integer coordinates and perform our area computations using only integer arithmetic.

Our piecewise function is defined as a collection of segments, and each of these segments is a right trapezoid. The main thing we need is a function for finding the area of such a trapezoid. I derived this by treating the right trapezoid as a rectangle with a triangle on top, finding the area of each, and summing them. Then I looked at www.mathworld.com and saw that they have a diagram and area derivation for a right trapezoid (see Eric Weisstein, "Trapezoid," <http://mathworld.com/Trapezoid.html>).

The area of the entire trapezoid is $Area(x) = 1/2 \Delta x(y_0 + y_1)$. However, what we really want is the area of a piece of this trapezoid. In Figure 3b, we let $x_0 = 0$ to simplify the math, and we find $Area(x)$, where x is between 0 and Δx . After a little bit of algebraic substitution, we find that $Area(x) = y_0x + 1/2(\Delta y/\Delta x)x^2$. This result will often not be an integer, but that is not really important if we minimize the rounding and compute this function consistently so that rounding happens the same way. To minimize the rounding, we group the terms like this: $Area(x) = y_0x + (\Delta yx^2)/(2\Delta x)$. Now there is only one divide operation, and it happens as late as possible, and it does not become magnified by any subsequent multiplications. Thus the rounding error will always be less than 1.

Earlier I said that the width of a coding interval was determined by its y value in the piecewise linear function. Ideally that's true, but in order to make the interval sizes always fit perfectly together, and to be consistent with rounding, it is best to define the interval width in terms of the trapezoid area function.

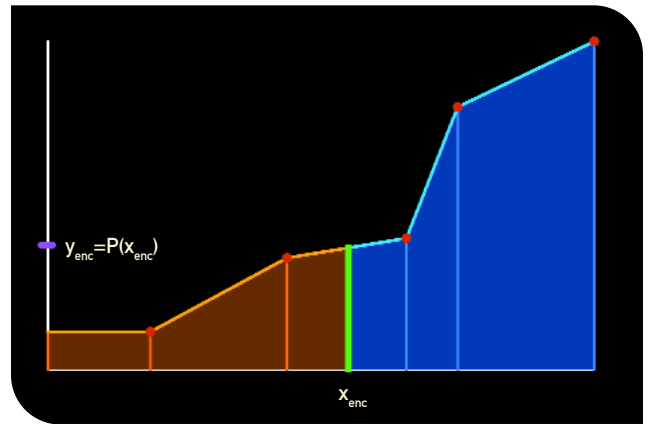


FIGURE 2. We wish to pack the value x_{enc} with an arithmetic coder. The interval width is determined by the function's value at x_{enc} (the height of the green line). The interval starting point is determined by the area of the function to the left of the green line (colored in orange).

So, the width of the interval for x is $Area(x + 1) - Area(x)$.

By letting $x_0 = 0$, we have been computing the area of each trapezoid in isolation. But it's easy to iterate over all the trapezoids at preprocess time and find their areas. Then we store an integer value with each trapezoid, telling us the area of all previous trapezoids in the function. We add that value to the $Area(x)$ computed above.

The only remaining task is to find the correct trapezoid for a given x . In this month's sample code I implemented a simple-as-possible solution, where I just iterate over the trapezoids from left to right until I find the one that spans x . This is fine if you assume that your functions aren't built from too many trapezoids. If you need more speed, you can always binary search or use a caching scheme.

How to Decode This

When decoding the message, the decoder provides us an area value which we will call A ; we want to map A through the inverse of our area function to find x . A will be somewhere between $Area(x)$ and $Area(x + 1)$. One simplistic but slow way to decode it is to iterate over all values of x , computing $Area(x)$ and $Area(x + 1)$ until we get results that straddle A . (Hey, it's not pretty, but it was the first thing I implemented and it worked the first time. Don't underestimate the value of simplicity when you are initially getting a system running.)

For a more elegant solution, we can find an equation for the inverse of the area function $Area(x) = y_0x + (\Delta yx^2)/(2\Delta x)$; one way to do that is by applying the quadratic formula. The quadratic formula contains an annoying plus-or-minus, but we can discard the minus since it represents a solution we're not interested in (outside the left-hand edge of the trapezoid). So $x = -y_0 + (y_0^2 + 2\Delta y/\Delta x A)^{1/2}(\Delta x/\Delta y)$.

We need to manipulate this equation to minimize rounding as before. That square root operation is obviously going to be a source of rounding, which is then exacerbated due to the multiplication by Δx . To fix this we factor the Δx into the square root (valid since $\Delta x > 0$) to yield: $x = -y_0 + (y_0^2 \Delta x^2 + 2\Delta x \Delta y A)^{1/2} / \Delta y$. Now there are two sources of round-off, the square root and the divide, but they compound directly and the error will always be less than 1.

There's another rounding issue that's a little more subtle: when solving the quadratic we treat the area as a continuous function on the domain of real numbers; but in reality, it is a discrete function on the domain of integers. In other words, we're not accounting for the fact that the area we compute at encode time gets rounded down to the nearest integer. We can rectify this by taking the x given to us by the quadratic formula, checking A against the lower and upper bounds of the forward area function for that x , and bumping that x upward or downward if necessary.

Since the inverse function divides by Δy , it's undefined when $\Delta y = 0$, which is a totally reasonable input case. But then, we can just go back to the area function and write it as $Area(x) = y_0 x$, and thus at decode time, $x = A / y_0$. Now, this new function is undefined when $y_0 = 0$, but that should never


happen, since it means the values in that trapezoid have probability 0. That would mean it's not possible to transmit them in the first place (you should use an escape sequence at times like that, as we did in the earlier articles). We don't want to trust network input, though, so if we encounter this case, we want to catch it and perhaps reject the entire message (since this is a sign that someone is screwing with us).

So far I have said that A is the area value gotten from the arithmetic decoder, but I have glossed over the actual computation of A . Let $B = ((code - low) / range) * A_0$. The $((code - low) / range)$ gives you a value in $[0, 1]$ corresponding to the remainder of the message; let A_0 be the area of the entire function. Since we want to use integer math, we refactor the terms to perform the multiply before the divide: $B = ((code - low) * A_0) / range$. So B is the amount of area from the left-hand side of the function up to x . We find the segment of the function that contains B , then we say $A = B - A_{prev}$, where A_{prev} is the sum of all the previous trapezoids' areas. Now A tells us how much area in the current trapezoid is used by the message, and we apply the quadratic formula from there.

In this article I have focused on the details of encoding with a piecewise linear function. In principle any type of function will work, so long as we can evaluate it, integrate it, and invert it.

One might try to use a higher-order function than linear (such as quadratic or cubic splines), though in practice it can be very difficult to manipulate these with integer operations on native machine registers without overflowing. On the other hand, if a piecewise constant function yields a good enough approximation to your probability distribution, the math is simplified greatly. You can perform the basic techniques in this article, but the area function becomes trivial and there's no need for the quadratic formula.

Sample Code

This month's sample code (available at www.gdmag.com) applies piecewise linear data models to player health and position. In order to accommodate the math from this article, some of the code uses 64-bit integer operations. There are some limits on the resolution of the domain and range of the piecewise function (necessary to avoid overflow), but these limits are high enough that they likely won't be a problem. For details, see the README packaged with the code. 

Procedural Textures

Procedural textures have a bad name among a lot of artists, particularly in the game industry. Partly this is because they're difficult to use: after all, procedural textures are closer to programming than painting. Renderman, the best-known name in synthetic texturing, is a full-blown programming language, and the various graphic front ends that their packages provide are almost as mysterious as a shelf-load of C++ reference manuals. Procedurals also arouse a lot of suspicion on artistic grounds. Hackneyed and almost inescapable, veiny marbles and fractal noise are the CG counterparts of Powerpoint clip art. With this kind of a buildup, you may be wondering why I'd devote a column to procedural textures. Here are three good reasons for giving procedural textures a second glance:

Tool changes. In the past couple of years the major vendors have added functionality that allows procedural textures to be unwrapped into standard UV bitmaps. Any fancy procedural that could be used in a cinematic render can now be converted into a bitmap and run in real time.

Workloads. Traditional hand-texturing techniques are increasingly stressed as asset lists and texture budgets grow. A 1,024 texture may not be 16 times more work than a 256, but it is still more work. Add in the need for separate bump and specular maps and the burden gets worse. Procedurals are good at filling space, and they also excel at coordinating bump, specular, and effects maps.

Generativity. If somebody decides at the last minute that the scales of your hand-textured dragon are too small, odds are you'll have to redraw the entire texture by hand. A procedural dragon skin, on the other hand, can be resized with little fuss. Parametric variations can also generate the whole families of related textures quickly and cheaply.

Working with Procedurals

It's much easier to understand what procedurals can do with a concrete example. To help break down some preconceptions about what procedurals are good for, we'll skip the usual rocks and walls and go straight to a complex organic subject: a human head. Each step in creating the texture will also illustrate a fundamental technique that can be applied to any project.

Working with procedurals, while far from simple, is not as difficult as it appears. Texturing with procedurals is in fact surpris-

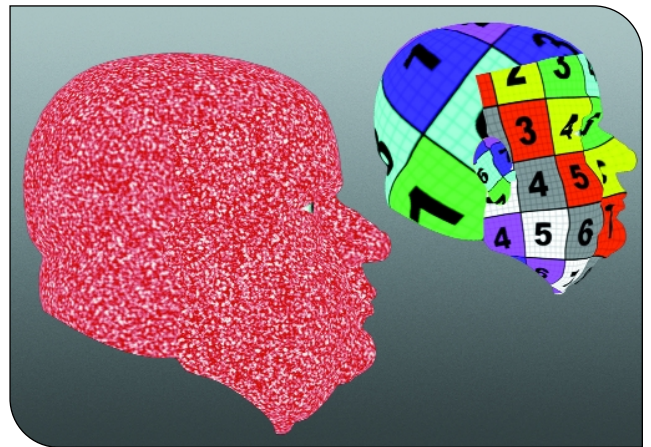


FIGURE 1. 3D noise helps disguise the differences in underlying UV map densities and orientations.

ingly similar to creating a layered Photoshop texture. If you were painting a wall texture by hand, you'd begin with a base color and then build up detail by adding layers for weathering, rain stains, cracks, and so on. Texturing the same wall procedurally amounts to very much the same thing, except the "layers" in a procedural are usually mathematical functions rather than bitmaps (although hand-painted maps are often vital parts of a procedural texture). The texture for our example will be built up in a manner similar to the physical structure of real skin, with underlying bone and fat and surface details like pores and hair.

We'll start with a layer that represents subcutaneous fat. More than anatomical detail, what we really want from this layer is a basic rhythm for the skin texture. This is very important for this model, since the scalp and back of the head are mapped at a much lower resolution than the face. The fat layer is made with a 3D noise tinted red. Since the noise is in 3D space, independent of the UV mapping, it will help tie the different map regions together and disguise the differing pixel grains (Figure 1).



STEVE THEODORE | *Steve started animating on a text-only mainframe renderer while working on a Ph.D. in ancient history. Since then, he has worked on games including HALF-LIFE and COUNTER-STRIKE. He can be reached at steve@theodox.com.*



FIGURE 2. Crude textures work fine for underlying skin structure.

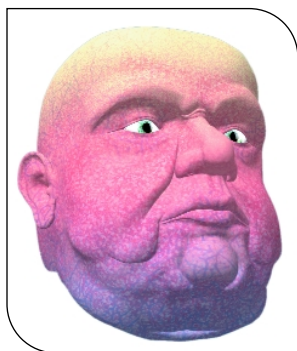


FIGURE 3. A cylindrical projection provides a nice color wash.

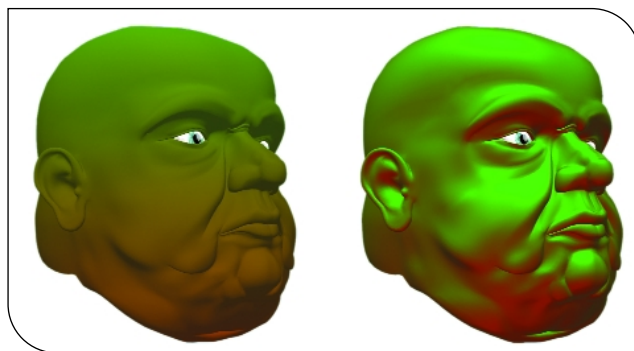


FIGURE 4. Two methods of applying the same gradient ramp: a spherical projection (left) and an environment map (right).

Masks

Now we need a way to represent bone or cartilage in areas like the ears and nose. The easiest way to do this is by painting an alpha mask that tells us where cartilage should replace fat. For simplicity's sake, the mask is very crude. To make it subtler we can multiply it against a 3D noise texture before applying it (a common Photoshop trick). Combining noise with masks is a timesaving compromise between pure randomness and the burden of painting too much detail.

A similar mask is used to represent the blood vessels under the skin. Since this texture is destined to be rendered to a bitmap and hand-retouched, it doesn't make sense to spend a lot of time hand-placing veins — the suggestion of structure is all we're after — so we use a generic veiny bitmap applied with a cubic projection and multiplied against noise. If placement were really critical, this layer could be hand-painted for accuracy. We've now got a complex foundation, with multiple overlapping rhythms that will give the skin a lot of local variation (Figure 2).

Projections

The job of the next layer will be to tie the final texture together. In a 2D illustration, we'd probably airbrush in the traditional portrait painter's trio of golden forehead, red cheeks, and blue chin into the skin color. Unfortunately, a gradient that would be very simple to create in a 2D image is very hard to apply in a 3D paint package, especially if the underlying UVs are complicated. To get around this we apply a gradient ramp with a 3D projection that's independent of the UV mapping on the geometry. When we collapse the final texture down to a bitmap, all of the magic necessary to get that smooth wash into UV space happens automatically.

Combining multiple projections into a single texture is extremely valuable. You can, for example, paint low-resolution masks using planar projections from whatever angle is convenient, without needing 3D paint software. You can even combine multiple hand-painted planar maps into a single texture without old-fashioned cut-and-paste hassles. An excellent aid to this procedure is a fresnel or "falloff" mask, a procedural that returns a grayscale value dependent on the facing ratio of a surface. If each planar projection is masked by a fresnel procedure with the same

orientation, the projections will naturally fade out wherever they begin to smear or stretch.

Our color wash is made with a procedural gradient ramp running from yellow to red to blue and applied with a cylindrical projection. Since all of these colors are pretty high in value and low in saturation, the range of RGB values is small. To prevent banding, the gradient has a very high-frequency noise in it, which acts like an old-school dither pattern. As you can see in Figure 3, the color wash looks artificial, even when the underlying noisy layers are allowed to peek through. Some of this is because the color wash is too cartoony, but the real problem is that the wash is too independent of the underlying geometry. This is what we wanted from the separate projection — but the effect goes too far.

Environment Maps

To bind the texture to the geometry, we'll add a bit of pseudo-shading. While higher polygon counts and better lighting models are gradually making false shading less important to texture painting, small doses of fake shading can still be very helpful. To get the effect without hand-painting, we'll use an environment map instead of a standard projection, because environment maps apply textures according to the facing direction rather than the spatial position. Figure 4 clearly shows the difference between a spherical projection and a spherical environment map. This is a great hack to get a bit of trompe-l'oeil shading or fake global illumination without hand-painting.

The environment map projects a gradient ramping from a very pale pink on top to a purplish red on the bottom. This will accentuate the creases and undercuts in the geometry. Keeping the variation mostly in saturation (rather than in value) makes it less likely that the trickery will be exposed by a moving light source or an extreme animation pose.

Combining the faux-shading layer with the color wash presents another problem. Again the RGB range is fairly small, so a simple blend will show a lot of banding. By blending the shade layer onto the wash using a very high-frequency noise texture as an alpha mask, we can dither them together. Dithering masks work best with unstructured or Gaussian noise types similar to the "Add Noise" Photoshop filter. Fractal noise, similar to Photoshop's "Add Clouds," would produce a patchy look instead of a smooth blend. Figure 5 shows the composited results.

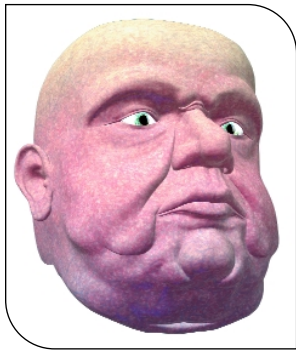


FIGURE 5. Blend between a color wash and an environment map.

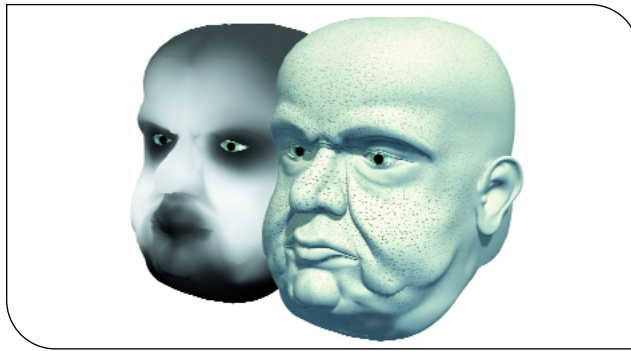


FIGURE 6. A grayscale map (left) controls both the density and the size of the pores (right).



FIGURE 7. Final image with procedurals converted to textures.

Parametric Variations

In our post-*Spirits Within* world, no skin texture is complete without pores, stubble, and other imperfections. Pores are pretty simple to imagine: they're just spots in the color map that also show up as indentations on the bump map and opaque blips in the specular. Unfortunately, the size and distribution of pores in the face varies a great deal. Pores are most prominent on the cheeks, the saddle of the nose, the chin, and the forehead; around the eyes, ears, and mouth they are both smaller and more tightly packed. Hand-painting individual pores would be maddening, but plain random spots, even with a hand-painted mask, won't reflect the structure correctly.

To manage the placement of the pores we'll use a parameter map to control the procedural, which generates them. The pores themselves are generated by a bubble noise texture, similar to the one we used for the fat layer. In this case the density is turned way down so that the bubbles appear as isolated spots instead of clumping together into blobs. We'll need to tweak the parameters to get reasonable looks for the two extremes of large, sparse pores and small, tightly packed ones. Once we know what the extreme values should be, we paint a quick grayscale mask that says, "big pores here, little pores there." Then we connect the mask's output to the parameters of the noise procedure, remapping the grayscale values to the numbers we had earlier tweaked out (Figure 6). The same trick with a second mask manages the distribution of hair on the scalp and stubble on the cheeks.

Parameter mapping is an extremely powerful technique, a graphically accessible version of the kind of wizardry that used to be the exclusive province of Renderman programmers. Unfortunately (like programming) it also sports a wicked learning curve and demands a lot of trial-and-error in the learning stage.

Endgame: Collapsing and Converting

Any procedural texturing project eventually reaches a point of diminishing returns, where the effects to be added are so subtle that they will no longer repay the effort of creating them, or where it simply becomes easier and faster to convert the procedural network to bitmaps and add the remaining details by hand. Our test case is going to need refinements such as wrinkles, lips, eyebrows, and so forth that are so closely

dependent on the geometry that handling them procedurally is only masochism. Knowing when to stop is an important aspect of working with procedurals — the process can become slightly addictive. Your audience will care if it looks good, and your producer will care if it's done quickly, but nobody will give you brownie points just for completing a texture without ever opening a paint program.

When converting the procedures to bitmaps, you may prefer to handle the final compositing in a paint package, or you may try to get the texture network as close to complete within your package as possible. The former method has the advantage of familiar tools and easy retouching; the latter allows more iterating and less hopping back and forth between applications. One useful trick, which was used in the completion of this image, is to keep a faint noise layer separate from the rest of the skin procedural. After the hand-editing is done this noise layer can be converted to a separate bitmap and composited over the retouched texture as a "soft light" map. This adds a small amount of grain to the final texture with a consistent 3D scale, a great aid to uniting the different UV mapping regions and keeping the manual edits from looking too simple and out of place.

Because the procedure generated such a strong base, the final image required only a couple of hours' handwork to add eyebrows, wrinkles, and some warts for believability (Figure 7). Most of the final image is a straight conversion from the procedural (with bump and specular maps derived straight from the pore map).

Expanding the Palette

As models and textures reach ever-higher resolutions, we're going to be forced into greater reliance on techniques beyond painting. Tool vendors can certainly do a much better job of making these tools more accessible and friendlier to experimentation; however, we can't afford to wait until that day comes — the demands on us are going to increase regardless. Working with procedurals is not rocket science; in many ways it's just an extension of tools and techniques most texture artists already know. If the process seems intimidating, it's worth remembering the introduction of Photoshop layers or the early days of UV mapping to realize that many tools that seem unnatural at first quickly become second nature. 🎨

Shhhh!

Be Vewy, Vewy Quiet

Sound is a critical element in film, television, and video-games, but silence is just as important as the sounds we do hear. To instill a feeling of isolation and vulnerability in a game, try removing all music and background sounds. The human ear will adjust to the silence, trying to make out any sound. A soft scrape in the rear speaker will definitely have a player's full attention and be far more effective in maintaining that mood of fear than a gunshot.

A balancing act. It's often more difficult to create a sound environment that doesn't include gunfire and explosions. While it's no more complex to add the sound of a bird than the sound of a gun, it's much harder to impress an audience with a subtle ambient sound environment than with an earth-shattering explosion. For example, in developing the sound for *JURASSIC PARK: OPERATION GENESIS*, everyone on the team loved to hear the giant roars of the dinosaurs and the mighty stomp of the T-Rex. But it was the tiny sounds that were the most problematic to blend, such as the sneeze of a little dinosaur, or the sound of grass moving when a dinosaur sat down. These sounds had to be balanced carefully so that they were audible only at close range. The team enjoyed the final audio environment, but they often could not describe the exact reasons they thought it was effective; it just felt right.

It's often the quiet, so-called unimportant sounds that are responsible for the sound environment seeming more real. The audience doesn't notice most of these sounds individually, but they contribute to the player feeling as though the game world is real. When adding a new sound to a game I take a slightly counterintuitive approach: I actually make it much louder than it should be. That way I am sure to hear it, and I can check if it's in the right place. Then as I play the game I reduce that sound each time until I can barely hear it. There are



Quiet, natural sounds vary with proximity to dinosaurs in *JURASSIC PARK: OPERATION GENESIS*.

sounds in *OPERATION GENESIS* that can only be heard when a player zooms in as far as possible on a dinosaur. If you want to hear a dinosaur's skin creak when she sits down, you've got to get really close.

Similarly, loud footsteps can disrupt a game's reality. When was the last time you noticed your own feet thundering along beneath you? Yet so many games balance the footsteps as if every character were a 40-ton mech. By comparison, imagine the impact when you first become aware of your character's steps echoing as you move across a metal walkway in an abandoned warehouse.

Focusing in on sound. Sound can be moved in and out of focus in the same way pictures can. For a battlefield scene, the sound designer can place emphasis on a particular event in an obvious way by making the sound louder. Alternatively, a similar impact can be achieved by suddenly reducing all sounds on the battlefield except for the whistle of the incoming shell. Borrowing from the visual domain, this is like changing the depth of field to focus on a single sound event. The player will be acutely aware that

something significant is about to happen. Another way to achieve this heightened sensitivity is to reduce the background audio but leave the main sound focus on the character's breathing and muddy footsteps as he skulks through soggy trenches. This keeps that isolation button pushed down, even though there may be many other characters on the battlefield. When sound is implemented in this way, a gunshot played at "normal" volume inserts itself into the player's world with such dramatic contrast that the player is instantly aware that the enemy is within striking distance.

Selective use of music. Many games make use of ambient music tracks that accompany general gameplay. When there isn't anything significant happening, ambient music plays in the background to highlight the world's environment. It seems, however, that as a legacy of early games where short music themes looped eternally, many people expect that music should always be present in a game. I once received a bug report from a QA tester listing a bug with the music system because music didn't always play. Music can also benefit from a "less is more" approach. If the ambient music is implemented with periods of random silence between the tracks, say between one and two minutes, the impact of the music can be far greater. The music works less as sonic wallpaper and more to highlight whatever is happening.

Well-placed silence can be truly effective in immersing a player and is very cheap on memory. And while the gentle sound of leaves moving in the breeze may not shake everyone's world, players will appreciate it, even if they don't notice exactly what it is they're appreciating. 🦖



STEPHAN SCHÜTZE | Stephan is the lead audio designer at Blue Tongue Entertainment Ltd. (www.bluetongue.com) based in Melbourne, Australia. He enjoys working as both sound designer and composer for anything that needs noise. Taiko drumming is his newest outlet for transmitting large quantities of sonic energy.

Tapas-Down Design

I recently hosted a dinner for game designers at a Spanish tapas restaurant that offers a wide variety of hot and cold appetizers to build your meal. Talk turned immediately to how we should order. Several ideas were thrown out in typical brainstorming fashion — “Let’s order one of each and then get more of the ones we like.” “Let’s vote on three top choices and start with those.” “Let’s order one meat, one fish, and one vegetarian dish first.”

Lively discussion about the merits of these and other algorithms ensued, complete with consideration of how we’d keep track of costs, the relative fairness of different methods, and occasional vehement interjections such as “I hate eggplant!” Eventually hunger prompted us to end the debate. With a group of designers contributing not to a meal but to a game design, one danger is that the debate will continue until the project is cancelled — and then everyone goes hungry.

Rule: Provide a consistent, single vision throughout the game development process. My March 2003 column (“Evolving the 400”) alluded to this rule, which I believe is one of the most important ones of all the 400. There are many ways to arrive at a single consistent vision, but without it, a game is very likely doomed from the start, or at best will be a mediocre jumble of conflicting ideas.

The rule’s domain. This is a meta-rule about the process of game design, and applies to all games.

Rules that it trumps. This rule trumps the well-meaning but misguided rule, “Let everyone contribute to the design equally.” It is in fact a good, even vital rule that everyone on a project may contribute and should have their contribution acknowledged — but having a single, consistent vision lets people understand which of their own ideas can help enhance that vision and which are really better saved for a different game.

Rules it is trumped by. “Let the game adapt to the player.” This rule will be cov-



Valve’s *HALF-LIFE 2*: can the cabal approach succeed again?

ered in a future column, but in short, there are times when it is best to allow a game to appear to be a different game to different players, even if those different games appear to have separate visions. The best solution is for the game to appear to have a single, consistent vision in each player’s experience.

Examples and counterexamples. There are many ways a single, consistent vision can be achieved. One is the auteur method, where a single person with a strong personal style exercises creative control, but shared visions are increasingly the rule as project sizes grow. An approach that worked well on two *INDIANA JONES* games I co-designed was to let that established, strong character franchise provide the template by which game design decisions could be measured. It was easy to ask, “Would Indy do that?” It’s interesting to note that the *Indiana Jones* movies are the result of a shared creative vision of two filmmakers with distinct styles.

Then there’s the fabled cabal method

that Valve used to create the original *HALF-LIFE*. In such an approach, an entire team can share a vision without strong personalities imposing their will, but it’s a risky undertaking. A method to ensure this vision that has worked for me is to put out a vision statement early on, capturing the experience the player will have in a high-concept-style sentence or two. This then becomes the standard by which all ideas are measured, and if they don’t expedite that vision, they are tabled.

Here’s an example of a vision statement of a hypothetical mobile phone game in which players exchange semi-autonomous “digital celebrities” who gain fame by being traded multiple times and become tokens in a simple pattern-based game:

CELEBRITAIRE combines the simplicity and compelling repeatability of SOLITAIRE with the emergent complexity and social play of POKÉMON.

As you can see, it’s not enough to build the game, but it can serve as a touchstone to measure whether the game in progress has captured the right feeling.

Dessert. Perhaps you’re wondering what vision prevailed at our designer’s dinner. We each decided to order the dish we liked and then pass them around, then split the bill. It worked fine until it became apparent that one end of the table (yes, it was my end) seemed to have a bottomless appetite and was running up the tab. And even this potentially divisive problem worked out. A generous designer/studio head surprised us all by grabbing the bill and subsidizing us all. This shows that in the real world a flawed vision may need to be saved by an executive with money. If only all game projects had such happy endings! 🍷



NOAH FALSTEIN | Noah is a 23-year veteran of the game industry. His web site, www.theinspiracy.com, has a description of *The 400 Project*, the basis for these columns. Also at that site is a list of the game design rules collected so far, and tips on how to use them. You can e-mail Noah at noah@theinspiracy.com.



As games have become more complex and visually demanding, the increasing need for animation data storage has also become a difficult and important issue. Since animation data takes up a huge chunk of memory map, game developers need to try to reduce animation data to the minimum.

Squeezing

Illustration by Dominic Bugatto

the Animation



STEVEN BATISTE | Steven is co-owner and lead programmer at Genuine Games. He is currently developing an as-yet-unannounced fighting game for Vivendi Universal on the Playstation 2 and Xbox platforms. He can be reached at ste@genuinegames.com.

The Memory Squeeze

Previously, most games only needed to store bone rotations, as that was all that was required to make models animate in a basic manner. The first wave of animation engines in games would play basic animations, usually using Euler angles stored in degrees or some other similar format of fixed-integer-based numbers. These animations would then usually be exported, so the keys per second rate would match the final game's frame rate. In the days of games running at 20 or 30 fps, this wasn't too much of a concern.

However, as games evolved, so did animation engines. By 1996 many game developers were starting to change or had already changed from using angles to quaternions for rotations. With quaternions, people could take their animations exported at 20 or 30 keys per second (kps) easily and slerp them to run at any speed without all the problems of angles. The major downside of switching to quaternions was the increased memory usage. We went from storing three 16-bit angles to storing four floating point numbers, that's six bytes to 16 bytes.

Animation in games now is expected to look more like movie-rendered CGI, and animators are requesting translations and scale data stored in the animation. Such a system allows more realistic skeletal features, such as spine compression that offers not only bones but bones with spongy cartilage. This system also helps keep the skeletal rig simpler, because the bone chain twists can be aided with translations.

In order to have this animation system to last for the next five years, it became clear that I needed to support rotations, translations, and scales for our current and future projects. This animation engine, if used for characters or who-knows-what in the future, has to be able to stand next to the best games at E3 2008 without looking inferior in the slightest way.

Opening the Raw Data Cookbook

As animators and producers increase the total seconds of in-game animation, developers need ways to compress the data while still maintaining speed. With our game's skeletons consisting of 89-plus bones, I didn't fancy trading memory for speed.

Before thinking about compression techniques for animations, I researched the existing methods without really find anything. The same was true when I looked through various 3D game programming books. Most articles would describe the various methods behind animation, but the memory used was rarely mentioned.

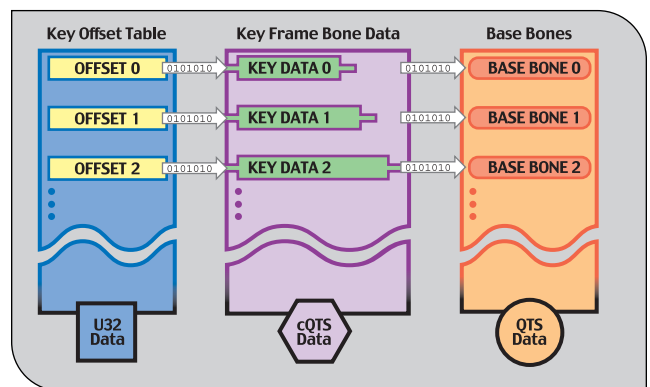


FIGURE 1. The relationships between the various data structures.

I started to realize that I would have to come up with something on my own.

My research indicated that the most common way of storing animations is as raw data. Motion-captured data is the least ideal: the percentage of memory it takes is simply too large. This is mainly because in order to get a good capture of the marker data, you need a fairly high capture frame rate. This is then converted into bone data that remains at a high key rate.

As stated earlier, our animation engine needed to use and store scale, rotation, and translation bone information, with the rotations stored as quaternions. To make animations visibly play the same in the game as they did when they were created, we need to keep a fair amount of precision in the animation data. This means that we cannot simply reduce a float down to an 8-bit-signed number and ignore the slight visual differences.

When I was thinking about ways to compress animation, there were a few constraints that I needed to follow. First, animations should be able to play at slower, faster, and normal speeds; second, animations should be able to play in reverse as well as forward; and third, animations should visibly look as they do in Maya or Max.

I identified the first two constraints because they allow some nice replay and camera effects. Games no longer just require a camera module; if we are to give players the more cinematic experiences they demand, games also need a director module that sits atop the cameras.

The third constraint is for a simple reason: animators like to see their animations looking as they animated them. They hate spending their time on making their ideal animation, only to find out it look less fluid in the game. Also, I thought it would be nice if we could keep the ability to stream animations from hard drive, CD, or DVD.

LISTING 1. BaseBone data structure.

```

struct BaseBone
{
    HASH    Name;
    U32     Frames;
    U16     Flags;    //bit 1 specifies if bone removed from key data
    S16     Quat[4];
    VEC3DF  Tran;
    S16     Scale[3];
}
    
```

Compressing the Data

The first simple compression method is to reduce the number of keys per second. This is a task best left to the artists, as they will best know which animations can be reduced to a certain degree. Based on a game frame rate of 60 fps, we found that 30 kps worked well for most animations, and a few could go down to 20 and some even 15 kps.

When you play the animation back, you will need to interpolate between the keys. The standard lerp-ing-and-slerp-ing techniques can be used to interpolate the in-between keyframes. Although doing so may lose some of the acceleration or deceleration that an animator puts in his or her animations, most of the time it won't be noticeable.

I began the second phase of compression by looking at the input key data. I first noticed that scales are usually 1.0, 1.0, 1.0 and 0.0, 0.0, 0.0 for the translations. This gave me the idea that I could somehow flag the key data to say, "Instead of storing the data, store a flag saying to use an identity quaternion, translation, or scale." Secondly, I noticed that for any given bone in an animation, the same key data would be repeated many times over throughout the frames. For this reason, I stored a base bone that has the most popular quaternion, translation, and scale. Then, like the identity flag, I would use a bitfield flag to say "Use the base bone" rather than "Use data present."

The `BaseBones` has the name of the bone stored as a hash number, which is useful for applying one animation onto multiple different skeletons (Listing 1). It also allows the number of bones in an animation to be less than the number of bones in the skeleton, which can save a significant amount of memory.

If all the key data for a bone in the animation is the same, then the `BaseBone` has a flag to say use the `BaseBone` data for all keys for that bone, and that the key has been stripped from the animation.

The key stream data always start with a `U16` flag for each bone. The data following the flags depends on the flag data.

The first four bits state the quaternion data type, the next four state the translations data type, and the last four state the scale data.

For all the quaternion, translation, and scales, the four bits represent:

LISTING 2. C-style pseudo code to retrieve quaternions, rotations, and scales for compression.

```

ANIM_KEY *GetKeyData(ANIM_KEY *key, ANIM_BASEBONE *BaseBone, QUATF
&quat, VEC4DF &tran, VEC4DF &scale)
{
    S16 *sptr=(S16 *) (key + 1);

    if ((key->Flags & 0x00f) == 0x002)
        quat.xyzw = (float)(*sptr++) * (1.f / (float)(1 << 15));
    else if ((key->Flags & 0x00f) == 0x001)
        quat.xyzw = (BaseBone->Quat.xyzw) * (1.f / (float)(1 << 15));
    else if ((key->Flags & 0x00f) == 0x003)
        quat.xyzw = Identity;

    if ((key->Flags & 0x0f0) == 0x020)
        tran.xyz = *(float *)sptr++;
    else if ((key->Flags & 0x0f0) == 0x010)
        tran.xyz = BaseBone->Tran.xyz;
    else if ((key->Flags & 0x0f0) == 0x030)
        tran.xyz = 0.f;

    if ((key->Flags & 0xf00) == 0x200)
        scale.xyz = (float)(BaseBone->Scale.xyz) * (1.f / (float)
(1 << 12)), sptr++;
    else if ((key->Flags & 0xf00) == 0x100)
        scale.xyz = (float)(BaseBone->Scale.xyz) * (1.f / (float)
(1 << 12));
    else if ((key->Flags & 0xf00) == 0x300)
        scale.xyz = 1.f;

    return (ANIM_KEY *)sptr;
}
    
```

0 – unused

1 – use `BaseBone` data (no key data present)

2 – use data in key

3 – use identity (no key data present)

The data after the flags is always in the order of quaternion, translation, and scale. Any, if not all, of these could certainly be stripped out.

Because the key data is being stripped, we ended up with variable key frame sizes. This is fine for streaming animations. However, since I wanted to be able to play animations at varying playback rates, I need to be able to retrieve any key data without parsing through the animation from the beginning of the file. For this, I stored an offset table in the animation file that stores for each key the offset into the data stream. Figure 1 (page 29) helps illustrate how the offset table, key data, and `basebones` relate to one another.

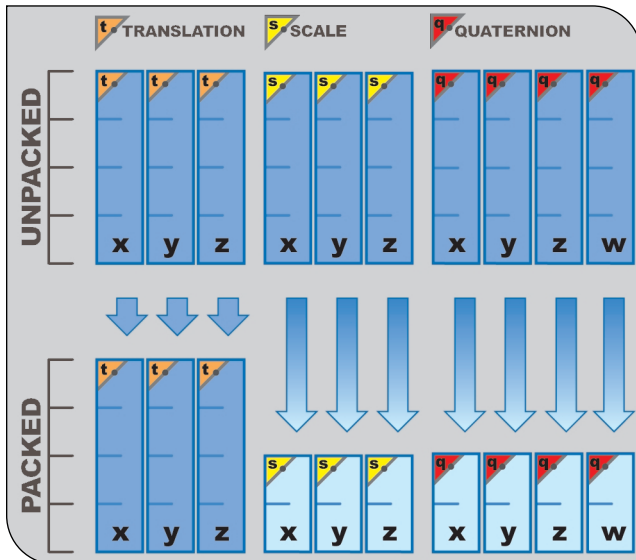


FIGURE 2. A comparison between a packed and unpacked key.

Lastly, I used fixed-point math to store the quaternions and scales (Listing 2). The quaternion is always a normalized quaternion with sign, for which I used 1:14:1 fixed-integer format. This gives us a range of ± 1.0 with a precision of $1 / 32,767$. On the other hand, the scale always remains positive. Therefore, I used 0:4:12 fixed-integer format, which means the maximum scale size can go up to 16.0 with a precision of $1 / 4,096$. This gives a 5-to-3.5 compression ratio just on the raw data storage. Figure 2 shows the packed/unpacked data side by side.

On the Flipside

For every right-handed move our current project has, there's a left-handed move that mirrors its counterpart, so it seemed like a good idea to make animations able to be playback-flipped. To accomplish this, I negate the translations and quaternions in the X-axis after decompression. I also need to apply any right-side bones onto the left-side bones and vice versa. In our animation rig, the animators have the ability to map the left- and right-handed bones to each other. This then

LISTING 3. C-style pseudo code for converting nonflipped into flipped bones.

```

if (fFlags & ANIM_FLAG_X_MIRROR)
{
    bone_index = bone_index_fliptable[bone_index]
    translation.x = - translation.x;
    quat.x = -quat.x;
    quat.w = - quat.w;
}

```

is stored as a table in the game's animation manager when the animations are loaded. We store each of our **BaseBones** with a hashed bone name, and our table stores the relationship between flipped bone names and the non-flipped bone names (Listing 3). This is a simple table lookup, but for this article, I will omit how our hashing system works. For the purpose of keeping the example code simple, it shows the bones as indices rather than names.

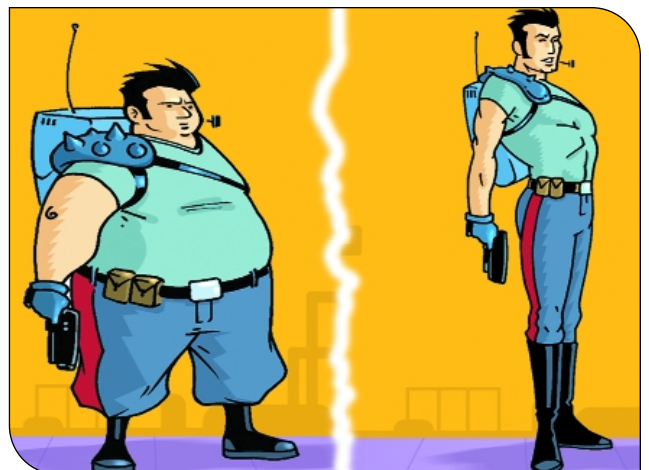
Results and Future Work

Overall, by using the techniques I've described we achieved a compression ratio of 30:1 for our current game. My target compression ratio before starting work on it was 20:1. Getting 50 percent compression on top of 20:1 was very pleasing. In fact, because the system works so well, we use it to store other types of movement in games, such as camera paths and AI rails. Treating data in this way makes it very easy to put together a non-interactive polygon sequence system that can stream from CD, DVD, or hard drive.

The bone translations and scales alongside the rotations really give the characters a fluid and natural looking movement. The sponginess the translations provide really adds a lot of weight to the characters. I just hope it'll add enough quality to the animations to allow my earlier E3 2008 statement to stay valid.

Future improvements could include reducing further the animation keys per second and store the animation tangents alongside the key data to prevent some of the robotic-looking artifacts when key frames are reduced. Also, rather than having just one quaternion, rotation, and scale stored in the **BaseBone** structure, multiple sets could be stored with extra bit flags to index them. The bone data could also index into a global quaternion, rotation, and scale table, rather than a local **BaseBone**.


In the future, animations with the inclusion of data to store the different modes of acceleration to get from one key to the next will reduce the amount of keys needed per second. Instead of linearly going from one key to another, algorithms may define a motion that cuts out the need for in-between keys. This means that more work will be placed on the CPU to calculate how to get from key *n* to key *p*, but this has the huge advantage of using less RAM, especially as processing power is increasing at a faster pace than memory growth. 🦄



Riffing on Tolkien:

**The conceptualization, production,
and dissemination of music in
Vivendi Universal Games'
LORD OF THE RINGS titles.**

CHANCE THOMAS | *This is where I'm asked to list my life's work in five or less lines, so here I am, categorized and compressed: Oscar, Emmy, Telly, Addy, Grammys. LORD OF THE RINGS, MIDDLE EARTH ONLINE, WAR OF THE RING, EARTH AND BEYOND, QUEST FOR GLORY 5, Men in Black 2 DVD, UNREAL 2. Vivendi Universal Games, Electronic Arts. SEK, AIAS, GANG, Game Developers Conference, E3, BSA, LDS, and USA. More details at www.HUGEsound.com.*



For Vivendi Universal Games' LORD OF THE RINGS titles, which include MIDDLE-EARTH ONLINE, WAR OF THE RING, THE HOBBIT, and FELLOWSHIP OF THE RING, the mandate was to create music that would approach Tolkien's idealized descriptions, music that would represent the relentless pursuit of quality evidenced in his books and that would also bind an entire series of games together over time and across multiple developers, composers, and platforms. I'll talk about creating an authentic music style guide for the franchise, producing high-quality music assets, and managing an innovative music implementation system at both the publisher and developer levels.

Hallowed Ground

Sacred texts. The Tolkien works are highly esteemed by millions of readers across the globe. For the fantasy genre faithful, the *Lord of the Rings* series nearly approaches canon. Daring to mingle our own mortal efforts with those of Tolkien was a risky venture and not a quest for the superficially inclined. This music needed to be drawn from the very pen of Tolkien's writings, ringing of truth to anyone familiar with its pages. The only way to avoid flaming out in the fires of Mount Doom was to know the literature completely, inside and out.

The Tolkien mind-meld. Thus it was that, over the course of five years, I logged hundreds of hours researching and annotating Tolkien's books for everything they had to say about music. I found passages describing specific musical instruments used by the various races. I found information about vocal tone qualities and inferred vocal ranges for the races of Dwarves, Hobbits, Elves, Men, and even monsters. I found more than 60 different songs in the books and studied them all, including song forms and styles. It was fascinating to read about the impact of music on characters, traits, and even the environment. As a result, my copies of the literature are dog-eared, underlined, cross-referenced, and yellowed — and not just from the gaggle of Post-It notes protruding from the pages.

Map of Middle-earth. From these notes, I created a Tolkien Music Style Guide to offer direction to the many composers who would be working with me on this game series. The style guide defines a specific palette of musical instruments for each race based on actual references in the text. It identifies specific voice types and ranges for each race, also based on references in the text. The underscore for each race is given harmonic, melodic, and rhythmic guidelines inferred from references in

the text. In addition, there are sections in the style guide discussing production quality standards, music design matrices, implementation alternatives, music delivery specifications, and much more.

For example, both Elves and Dwarves are known to play the harp. But unlike Elven harps, Dwarven harps are "strung with silver." We represent this in our scores with a rare wire-strung harp, recorded especially for our LOTR series by sample maestro Gary Garrigan. As another example, Hobbits' music is voiced by Celtic ensembles, based on the reference that Hobbits play "pipes and flutes." But they also played "horns and trumpets." You'll find them all in our Hobbit tunes. Also, Dwarves are reported to play "clarinets" and "viols as big as themselves," which we have also reflected authentically in our scores.

THE HOBBIT. A quick story from the development of THE HOBBIT (Christmas 2003, all console platforms) is illustrative. Composers Rod Abernethy and Dave Adams had been creating a wonderful collection of music to underscore Bilbo's adventures in Hobbiton. Almost everything was in complete harmony with the Tolkien Music Style Guide. But one piece of music seemed a little out of character. The arrangement was laced with marimba, a very cool instrument but decidedly out of place. Referencing the style guide I told the composers, "You see, there are no marimbas in The Shire!" There was a brief moment of silence, after which we all broke into laughter. The moment was comical in its self-importance, but the composers did make the change, and our adherence to the style guide successfully preserved a higher degree of authenticity in the score.

Other games served by the Tolkien Music Style Guide include WAR OF THE RING (Christmas 2003, RTS for PC), MIDDLE-EARTH ONLINE (Christmas 2004, MMO for PC), THE TREASON OF ISENGARD (recently cancelled), THE FELLOWSHIP OF THE RING (2002, consoles and PC), and several unannounced titles currently in development for console and PC.

Landmarks

The "five" towers. One of the most important recommendations in the Tolkien music style guide was that a series of main themes be written to reflect the essence of each key race in the story — Elves, Dwarves, Men, Hobbits, and the races of evil, represented by Sauron. These main themes would then be used in every LOTR game to lay the thematic underpinning for each game score. The themes would serve as musical landmarks in our games, tying all the scores together with a series of common

musical motifs and palettes. The task of composing these main themes fell to me.

I could have written 12 notes and said, “Here’s the Dwarves’ theme,” but in keeping with VUG’s vision that the series amount to a “leather-bound edition” quality, I proposed that each racial theme be showcased in an overture telling key parts of the LOTR story in music. Not only would this model the style guide’s recommendations in a broad range of potential gameplay situations, but it would also provide a plethora of multiple-utility music assets. These assets include dozens of high-quality music cues to implement directly in each game score, sectional stems (choir, strings, brass, and woodwinds) from the live recording sessions for integration in other composers’ scores, MIDI files to start each composer on the right track, and feature-length tracks appropriate for a music CD. To my knowledge, planning such a detailed musical framework in advance for an entire series of games has never been done like this before. VUG approved the main themes as outlined, and we were off to the races.

The Age of the Elves. As an example, let’s look at the structure for the Elves’ overture. This piece of music comprises several movements that showcase our four main Elven themes. The opening and closing movements, “From Across the Sea” and “Return to the Sea,” give us a feel for the immortal, solemn, and sad nature of the Elves. The middle three movements underscore the Elven strongholds in Middle-earth — Rivendell, Lothlórien, and Mirkwood — and reflect what the books tell us about the Elves in each particular region.

The Tolkien Music Style Guide defines the augmented 5th as a harmonic signature for the Elves, and the classical harp as a primary Elven instrument. The entire Elves’ overture is built upon these two constants, branching out with variations for each movement in ways that are completely unique and reflective of the various strains of Elvenkind.

In addition, each of the three middle movements shows two variations, which offer additional examples of Style Guide scoring. Thus, the yield from this single five-minute piece of music would be as follows:

- Eight examples of Style Guide scoring for the race of Elves.
- Five fully orchestrated PCM music cues for implementation directly in a game score .
- Dozens of sub-mixed music cues (harp, strings and voice, woodwinds and psaltry, and so on) from each movement for implementation directly in a game score.
- The source MIDI file for supporting composers to use as a starting point for their own scoring efforts.
- An adventurous overture for music lovers, which tells much of the Elves’ story.

Visit www.gdmag.com to hear some samples.

The Council of Elrond. This project was innovative and efficient music design at a global level. In order to ensure the broadest possible appeal and safeguard against my own potential biases, I



Ancient instruments, such as this dulcimer, lend authenticity to the LORD OF THE RINGS’ game scores.

composed the five thematic suites in cooperation with our developers, VUG management, and the other Tolkien directors. I sent everyone an MP3 of each draft and invite commentary, and many good suggestions came in. Kristofor Mellroth, an audio engineer at Surreal Software (TREASON OF ISENGARD), suggested we use some of the Black Speech in Sauron’s theme. Chris Pierson, one of the designers at Turbine Entertainment (MIDDLE-EARTH ONLINE), suggested specific lines of Dwarvish for the battle at Helm’s Deep. Daniel Greenberg, our creative director, helped steer me toward a better feel for Mirkwood. Even Vijay Lakshman, one of the VPs at VUG, got into the act, suggesting I beef up the drums in the Dwarves’ theme. It was a total team effort, and the end result was a collection of compositions we could all feel very good about. Time to move into production.

Producing the Themes

Middle-earth or middle-of-the-road? There was never any question in my mind that we would record the themes with as many live components as possible, striving for the highest possible quality standard. We simply had no other choice. Tolkien’s conceptualization of music was too idealized. He talks of musical instruments “of perfect make and enchanting tones.” He describes singing as “clear jewels of blended word and melody.” He refers to “power” in old songs, and even ascribes the ultimate creative power to music from the gods.

In addition, we had to consider the level of quality apparent in Tolkien’s writings. Careful attention to detail, painstaking effort in choosing words, great skill in painting verbal images of beauty and artistry, and a tireless thoroughness evidenced in all his books. It was clear that we must hold to the highest possible standards of excellence for our themes. And that meant finding a great orchestra, choir, and ancient acoustic instrumentalists.

And wither then...? There are dozens of orchestras around the world, but only a handful whose musicians have significant

experience with film, game, or television scores and whose facilities are suitable for recording. I quickly narrowed my list down to four — The Hollywood Symphony, The Northwest Sinfonia, The Utah Film Orchestra, and the Prague Philharmonic.

The L.A. group is the most experienced and claims “the best players in the world.” But they were the most expensive, even with the new AFM agreement negotiated by G.A.N.G. My recording budget would have disappeared all too quickly. *Scratch*.

The Prague Philharmonic was the least expensive. I could have recorded there all day every day for weeks. But to my ears, their performances come off sounding sharp, and their recorded sound has a brittle edge that I find aesthetically unappealing. *Scratch*.

That left Seattle and Salt Lake City. Seattle’s musician-per-hour rate is \$55. Salt Lake’s is \$50. Seattle’s Bastyr Church and Studio X are both more expensive than Salt Lake’s L.A. East Chapel, which goes for \$125 an hour. Both orchestras have tons of experience recording for media. Both groups have their share of good and bad stories to be told. I had recorded in Salt Lake City previously with good results, so in the end I went with the cost savings and experience. I chose Salt Lake City.

The Bridge of Khazad-dûm. Nothing focuses your attention quite like hundreds of dollars falling into the abyss every minute a large group of musicians is in the studio. And yet, nothing gives such a sick feeling as missing an ever so slightly out-of-tune phrase that could have been fixed with one more take. That is why producing a live recording is such a balancing act. On one side you have aesthetics: timing, tuning, dynamics, all those elusive ingredients that make emotive magic. On the other side there’s the budget: there is only so much money, and if you go over in one area, you generally must cut somewhere else. Under the incredible pressure of the moment, making those decisions well is the key to effective live orchestral production.

The Orchestra

These recordings would be used in current and future games, so I isolated and recorded the orchestra one section at a time — strings, brass, and then woodwinds. This approach yielded undiluted sectional “stems” which would work flexibly in any number of future arrangements, offering each game a chance to use the high-quality live recordings we made within the context of infinitely varied compositions. It maximized quality and flexibility in one fell swoop. We started with the strings.

Orcs and Elves. We didn’t exactly get off to a smooth start. Version incompatibility between my Windows XP drives and the studio’s Windows ME drives made it impossible to transfer my guide tracks. While the studio scrambled to find a fix, the orchestra grew restless. An outburst by a prominent member of the orchestra only added to the building tension. As we waited



Looking down the neck of a double bass from videographer John Pratt’s LOTR music documentary (see sidebar at www.gamasutra.com).

on the tech team, I watched the dollars slip into the chasm, and my blood pressure began to rise.

The control room finally called down with an interim solution. I rose to the podium and took a moment to gauge the atmosphere of the room. The players were unfocused, uneasy, and some seemed antagonistic. I was outwardly calm but totally rattled on the inside. This was no way to start a session, especially for my LORD OF THE RINGS score. So I did something I had never done in a session before. I announced to the orchestra that I was going to pray, and before they could protest I bowed my head and started talking loud enough for everyone to hear. I gave thanks for everyone’s talents and professionalism, I gave thanks for the rare privilege we had of making music for a living, and I asked for help in capturing a performance that would live up to the lofty standard of the literature. I said, “Amen,” picked up my baton, and started describing the story behind the first piece. Interestingly, those sessions gave us some of the best tracks I’ve ever recorded.

The Choir

In contrast to the orchestra, which benefited from some “divine intervention,” the choir was a hit right from the start. I contracted an ensemble of singers, most of whom perform with the Mormon Tabernacle Choir, and they far surpassed my expectations. Here are a few moments from our sessions together.

The Rise and Fall of Sauron. For Sauron’s theme, we took the inscription from the Ring of Power — “*Ash nazg durbatuluk, Ash nazg gimbatul, Ash nazg thrakatuluk agh burzum, Isbi krimpatuland*” — and set it to music. This is sung twice in Sauron’s theme, once with men only singing in a profoundly low octave, and the second time with full choir, including sopranos wailing on the top end. The singers were just way too good at this. I said to them, “What’s your choir director going to say when you tell him you’ve been singing the Black Speech of Mordor?”

The Song of the Dwarves. When we started this piece, the

singing was exceptional, but the feel of the caverns and the monotonous labor of the Dwarves was just not coming through. I asked the men's choir to march in place, and to sway from side to side for the next take, and their singing was totally altered. It was uncanny. As Tolkien videographer John Pratt later wrote: "To my amazement, simply excellent singing was transformed into the grandeur of generations of tireless hammers echoing into songs of celebration in the finished halls of Khazad-dûm!" A live producer needs lots of tricks up his sleeve.

The Ancient Instruments

Rare, antique acoustic instruments bring an ancient flavor and feeling to a game that nothing else in the world can. In truth, the sounds of these instruments are the only game elements that actually do come from another place and time. Getting some of these instruments into our *LORD OF THE RINGS* recordings was essential.

From Forochel to Belfalas. Ferreting out ancient instruments in the 21st century is an adventure game all by itself. You can wander unsuccessfully for days and still come up empty-handed. I was lucky to find Gael Schults, an ancient Celtic music enthusiast who knew just about everyone in the world of archaic instruments. She provided two of the instruments herself and put me in touch with several other great players. Some of the specialty instruments we used in recording our themes and songs included the hurdy-gurdy, viola di gamba, psaltry, penny whistle, recorder, mandolin, rebec, dulcimer, and even an arch lute (also known as a theorbo).

Getting in the Game

With the recordings completed, it was time to get the music into the hands of the developers and into our games. This process had great potential to fall apart, since all the developers had a different composer signed on for their individual game score. But our music design was developed for this precise circumstance, and it actually held up remarkably well through the first round of scoring. Let's examine some of the specifics.

The Forge and the Blacksmith. After bringing all the music tracks back to my Yosemite studio in five massive Pro Tools sessions, I started carving out the music cues. By way of explanation, music cues are game-useful segments of music that underscore a particular mood or game state, and are ready for implementation as a digital audio file. Every racial overture was broken down into five to ten such cues of the full orchestration, each lasting from 30 seconds to two minutes. I now had close to 35 usable cues from the main themes for all our developers.

Next, I went to work creating variations on these cues with different mixes. One section from "The Overture of Men" is instructive. From the movement entitled "The Life and Love of Men," I was also able to derive a brass-only mix, a harp and flute mix, a strings-only mix, a woodwinds-only mix, and a



The author scrutinizing and weighing every score against quality and budget.

strings and woodwinds mix. Each sounds remarkably different and conveys a unique atmosphere. Thus each is useful for a different scoring purpose. Suddenly the number of usable music cues was approaching 100.

I uploaded all of these music cues to VUG's FTP site and made them available to the developers. I also uploaded the original MIDI files and sectional stems, and I sent each developer an asset list with recommendations for using them. A personal visit with each composer followed, offering further instruction, encouragement, and clarification. A brief summary of the development of three *LOTR* scores will show how it all came together.

WAR OF THE RING. Composer Lennie Moore and the team at Liquid Entertainment have taken full advantage of every aspect of our design. Lennie used the MIDI files as a starting point for 75 to 80 percent of his compositions. He generally began by quoting one of the themes, working into a variation of the theme, then going off into a completely original idea. In producing the score, he made generous use of the choir stems, especially the phrases sung in Sauron's Black Speech. Each of the sectional stems has been utilized to add texture and definition to the score, and even the solo fiddle from "The Overture of Men" is mixed into one of his pieces. In addition, extra brass, pipes, voices, and Irish whistle sessions were contracted to record fresh material and some wild variations on Sauron's theme. Finally, music cues from the main themes were used under the movies and underscore some of the key game events and transitions. The result is an artfully complete score that is perfectly in harmony with the music style guide, sings the main themes with clarity and variety, and creates a unique identity for *WAR OF THE RING* within the body of Tolkien music.

TREASON OF ISENGARD. This game was cancelled, but that too can be instructive. Composer Brad Spear and the team at Surreal Software took a more selective approach with our design. Brad used the MIDI files generously but was more crafty in his quotes and quicker to move into variation and onto his own



Separate main themes were composed for each of the main races of Middle-earth, including Gimli's dwarves.

material. The TREASON score did not include any of the stems or music cues. Nevertheless, it adhered faithfully to the Tolkien Music Style Guide and quoted from the main themes reasonably enough to establish it as a VUG Tolkien game score.

MIDDLE-EARTH ONLINE. Instead of relying on MIDI files like the previous two games, developer Turbine Entertainment and composer Geoff Scott prefer sprinkling the game world with the ready-made music cues pulled from the main themes. These fully produced theme segments are perfect for an MMO, and MIDDLE-EARTH ONLINE is taking full advantage of them. As of this writing, there are at least 90 different cues that have been identified for implementation in the game. This abundant thematic foundation allows Geoff to concentrate on creating source music and specialty tunes for the game. In addition, he has contracted additional recordings on lute, solo woodwinds, and guitars, quoting some of the main themes by ear and offering grassroots variations for the score.

An Epic Journey

Great literature is a wonderful catalyst for the imagination, and very few works of literature inspire better than *The Lord of the Rings*. With our authoritatively documented Tolkien Music Style Guide, meticulously produced main themes, and successful franchise music design, each game score orbits tightly around an authentic Tolkien center, while offering its own unique adaptation and interpretation of the material. The result is a desirable union of individuality and continuity.

I am grateful for the chance given to write music from such a brilliant font of inspiration. In deference to LOTR fans I have been as thorough, scholarly, and authoritative as possible in adapting these works for music, and so have the various composers who have worked with me. I hope fans will find each game score evocative, authentic, of award-winning quality, and ultimately irresistible. In addition to the music cues available at www.gdmag.com, many of the examples I've discussed, plus video footage from the recording sessions and interviews with the creative team are available at www.LOTR.com, where VUG has built a web hub dedicated to our LORD OF THE RINGS music. 🎵



The composer's road starts on a blank page and finds its way to a full orchestral score.



GAME DATA

PUBLISHER:
Sierra Studios (Vivendi Universal Games)

NUMBER OF FULL-TIME DEVELOPERS:
30

NUMBER OF CONTRACTORS:
5 (including 2 testers)

LENGTH OF DEVELOPMENT:
Initial Development Phase:
22 Months;
Final Development Phase:
17 Months

RELEASE DATE:
September 2003

PLATFORMS:
PC Windows

DEVELOPMENT HARDWARE:
1-2GHz CPU PCs with 256MB-1GB of RAM and GeForce 2-FX cards

DEVELOPMENT SOFTWARE USED:
Visual Studio.NET, Perforce, Maya, Photoshop, Adobe AfterEffects

GEOFFREY THOMAS | Geoff holds a degree in film studies from Queen's University at Kingston, Ontario. He spent the last year as assistant producer on HOMEWORLD 2. Contact him at gthomas@relic.com.

STEPHANE MORICHERE-MATTE | Stephane holds a degree in software engineering from Polytechnic University in Montreal. He spent the last year as lead programmer on HOMEWORLD 2. Contact him at smmatte@relic.com.

JOSHUA MOSQUEIRA | Josh holds degrees in cinema studies and English literature from McGill University in Montreal. He spent the last 18 months as lead designer on HOMEWORLD 2. Contact him at jmosqueira@relic.com.

Developing a Sequel: Evolution, Not Revolution

Relic Entertainment's HOMEWORLD 2

When HOMEWORLD hit the scene in the fall of 1999 it met with immediate success in the industry. Receiving the Game of the Year award from *PC Gamer*, it was hailed as one of the finest RTS games ever made. The title broke as many conventions as it established while telling a compelling and engaging story. It was all that the then fledgling Relic Entertainment could have hoped for with their first title, both critically and commercially. It should therefore come as no surprise then that once the celebrations subsided, thoughts quickly turned to HOMEWORLD 2 and future glories. Flush with success, Relic and Sierra set out in the fall of 1999 to top the success of HOMEWORLD. In developing any sequel, the familiar temptation is to eclipse the original title in scope, innovation, and content. The perception driving this common desire is that more of the same is never enough, and the only way to succeed is to outdo the predecessor. It's surprising then that in the pursuit of this holy grail, developers quickly forget the lessons they learn from the original game and make the same mistakes over again. With HOMEWORLD 2, the original development cycle saw many core mechanics and systems radically changed in a race to re-revolutionize the RTS experience. What started off as a plan to deliver a sequel worthy of 1999's Game of the Year quickly ballooned in scope to a project beyond feasible development in the timeframe allocated. The studio learned the hard way not to try to reinvent the wheel. HOMEWORLD 2's subsequent voluntary six-month cancellation was a direct result of owning up to this lesson, which then became the vision that fueled the new team's phase of development. Building on strengths and what worked right, while reinventing what didn't was our focus.



What Went Right

1. Evolution, not revolution. The key to sequel development is to know when to push and when to polish. If the original title was successful, the last thing you want to do is fix

what isn't broken. On the contrary, you want to build on what worked and throw out what didn't. Invariably this doesn't boil down to deciding what features to add but rather where features can be cut. Developing a successful sequel is therefore the process of refining and improving what came before. If you want to dramatically change the core game concepts and add a slew of new features, you're better off with a new IP.

When HOMEWORLD 2 restarted, the team agreed upon a simple mandate: evolution, not revolution. This mantra guided us through the next 18 months of development. Time and again at meetings, in presentations, and in documentation these words provided the solid ground upon which we built HOMEWORLD 2.

Instead of pushing the envelope too far, we focused on improving those areas where HOMEWORLD fell short and built on the mechanics and story delivery methods that made it a critical success. In short, we tried to deliver more of what made HOMEWORLD great, taken to new levels of polish and refinement to make HOMEWORLD 2 the definitive sequel.

A simple concept to understand, the evolutionary vision permeated into every aspect of development. This concrete vision underpinned every important decision we made and gave us insight that proved critical when hard decisions needed to be made during development.

2. Scope reduction: The axe is your friend. HOMEWORLD 2 began its second phase of development in February 2002, at which time we planned the



development schedule based on the work already done. Unfortunately, this appraisal was at a significant disadvantage, as most of the team responsible for initial development had left the company. We didn't realize the scope of this issue until work began in earnest. A number of systems considered complete were, to our dismay, in bad shape. We knew they could be worked around and patched up, but only at considerable risk to game stability. As a result we made a hard decision: we removed these systems entirely and rebuilt them from the ground up.

As painful as this decision was, it was one of the best we made on the project. Rebuilding our collision, network, and rendering code gave us a leg up on optimization and helped us to weed out systems that were no longer required for gameplay (for example, the terrain system). With an extra three months worth of work being dumped onto the programming team, we had little choice but to cut features from the schedule in order to compensate. This unpleasant reality turned out to be a blessing in disguise, as it gave us the impetus to remove features and units that weren't essential to the game. One example of such a cut is the Commander mechanic, where unique units gained experience and shared bonuses with ships assigned to them. We thought it was cool, but the mechanic didn't serve to enhance the core gameplay. Cutting it gave us valuable art, design, and programming time that we put to better use on core features such as subsystems, strike groups, and AI.

3 ● A strong, experienced team. A project is only as strong as the people working on it, and in this respect *HOMEWORLD 2* was outstanding. We wouldn't have been able to complete this product if we didn't have a team of strong, experienced people. Out of 28 individuals (aside from leads), six had been leads on previous projects. Everyone had worked in game development prior to *HOMEWORLD 2*, and almost everyone had shipped a game before. One might expect bruised egos and sparks to fly, but this wasn't the case. Everyone on the team was focused on making a great game and understood that respect

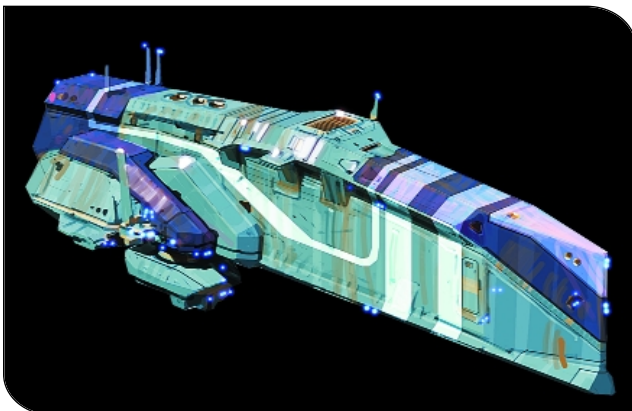
and cooperation were necessary parts of that goal.

Cultural diversity was another key factor. The *HOMEWORLD 2* team hailed from Canada, Australia, England, Holland, Japan, Mexico, and the United States. Each person brought unique perspectives and diverse game development experience, including role-playing, first-person shooters, sports, and real-time strategy. We all brought something to the table, and the net result was more than the sum of our parts.

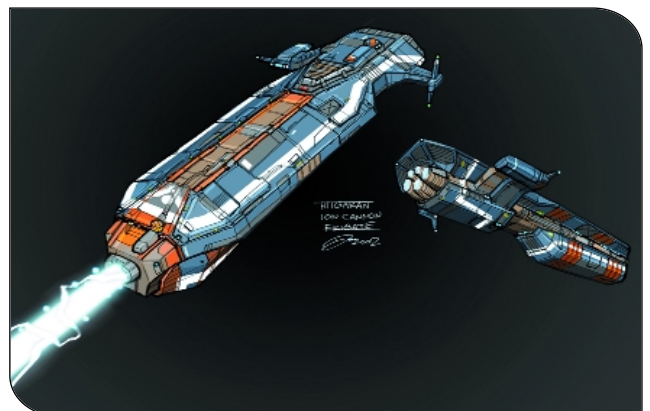
4 ● Art process. In a sea of RTS titles, the original *HOMEWORLD* distinguishes itself in large part due to its art direction. From the moment the game begins until the end of the final scene, *HOMEWORLD* is a seamless gestalt centered around a story-driven gameplay experience. With an engine capable of gorgeously rendering combat and action, the experience was unlike anything else available at the time.

The key then to developing a worthy successor fell largely on the shoulders of the art department — not an easy task, considering that only three of the artists who worked on *HOMEWORLD* contributed to the final development of *HOMEWORLD 2*. In the end, however, they not only created visuals that equaled the first game, they surpassed it beyond anyone's expectations. The key to their success lay in strong art direction fueled by a collaboration that adhered to the key lessons learned from *HOMEWORLD*. They quickly developed a strong collaborative process that gave each artist, from junior modeler to senior animator, equal input into what they created. While the art director set the stage, he thrived on constant input from his team throughout development.

The result was an iterative process where the art team shared comments with each other and members of other departments during texture overpaint sessions. Designers often sat in on modeling meetings to provide feedback and insight into unit function and how set pieces would fit with game mechanics. With each iteration, the result was a stronger, more refined vision of how each game element would look in its final form.



Early ship design, in which elements of the final Hiigaran Destroyer can be seen. At least, if you turn it upside down.



Very early sketch of the Hiigaran Ion Cannon Frigate. In this case, the concept was almost identical to the final game asset.

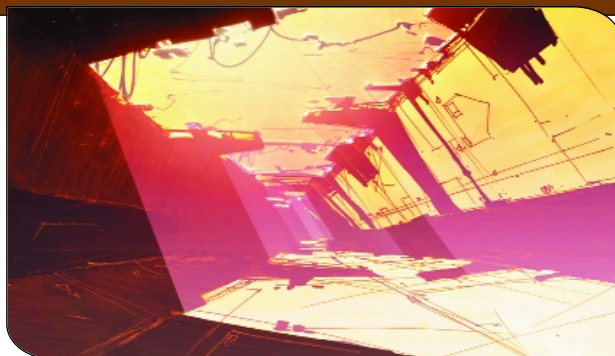


5 • Milestone Acceptance Tests. In order to track and validate our progress more effectively, we submitted our milestones to the publisher according to Milestone Acceptance Tests, or MATs. Deliverables for the next milestone were submitted as part of the current milestone's delivery, and acceptance of the milestone included acceptance of the next MAT. Both Sierra and Relic took this document very seriously; failure to deliver a single item would constitute failure of the entire submission. Putting into perspective what needed to be accomplished, the MAT was built from the schedule, and additional time for testing and build preparation had to be buffered into the deliverables in order to reach our goals.

It was only natural that we had an internal and external MAT for every milestone. The internal document tracked every deliverable scheduled to be completed, while the external document removed those tasks that had the greatest possibility of hitting snags or being otherwise delayed. Ideally this system meant that the team began working on the next MAT's deliverables a full week before the current milestone was finished. Fortunately this was the case more often than not, and the system worked very well right up to our beta milestone delivery. From our publisher's viewpoint, we had reached content and feature complete with our beta delivery and so we stopped writing any MAT documentation. In retrospect, it would have been better if we had continued adhering to internal MATs until much closer to gold.

What Went Wrong

1 • Collaborative design process. Much has been said in recent years about the advantages of the collaborative design process, and in many ways we benefited from this open approach to game development. *HOMEWORLD 2* is what it is because it's a gestalt. On their own the engine, art direction, and game design paint an incomplete picture of the final product. Only when seen as a whole does the game's true character shine, illustrating what can happen when all three disciplines of art, design, and programming work with a common vision in mind. However, collaborative design is not without its dangers, and we learned some painful lessons as a result. The first mistake we made was not establishing a clear delineation of authority over the script. This created two big problems.



Concept art illustrating some of the massive terrain pieces envisioned early on.

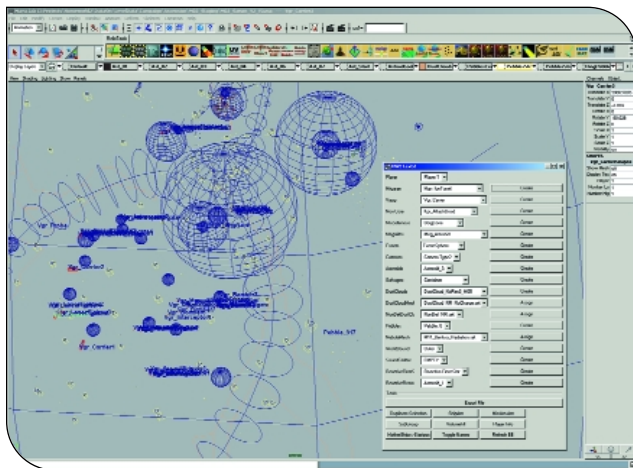
First, with multiple people having sign-off on script decisions, story meetings dragged on for hours — if not days — with no clear resolution. At times, though rare, the leads found themselves at an impasse, and invariably capitulation was the only solution. Inevitable lingering tensions between the art and design teams hampered productivity and ultimately affected production schedules.

Second, people who were responsible for areas of the game had little say when it came to making decisions that affected their work. Level designers needed to consult with a number of people before implementing even a small change to their missions. In the end, this overhead diminished their investment in their work.

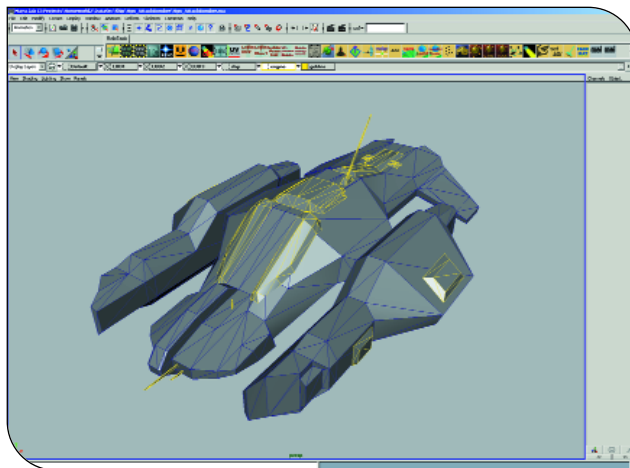
These issues caused tremendous grief during development. Throughout the entire process, what saved us was that every person involved not only acted professionally but held their peers and leads in the utmost respect. In the end it was the collaborative process that drove the team to resolve impasses and work as a whole.

However, our collaborative design process shouldn't have been a democracy. An open environment is critical to creating games, especially gestalt-driven titles such as *HOMEWORLD 2*. But ultimately someone needs to have the authority to make the decisions needed to move the project forward. The balance is in keeping the process from becoming a dictatorship while maintaining the authority and conviction to know when to say enough is enough.

2 • Late development iteration. To a person, *HOMEWORLD 2*'s art team was staffed by talented perfection-



Laying out a single-player mission map in Maya.



An untextured Hiigaran Bomber viewed in Maya.

ists. They're the reason the game is beautiful to look at and feels consistent from start to finish. They're also the reason that new content was being submitted well after we reached beta. In the pursuit of artistic perfection, scheduling and risk management unfortunately fell by the wayside. If an artist didn't feel that he had achieved an acceptable level of quality in the time allotted for a task, more time was allotted. This resulted in one of two possible outcomes: Either tasks yet to begin would have less time to reach the same level of quality, or the artist needed to work massive overtime to make up the deficit (which opens up a whole new can of worms).

It was an incredibly risky way to deliver content that endangered the art schedule, had a cascading effect on design and programming tasks, and nearly burned out a lot of very talented people. Obviously it's difficult to quantify what is by definition an issue of quality, but the schedule is there for a reason, and we should have made a greater effort to scale our expectations to fit within its borders. In our drive to make the best game we could, we too often ignored the facts, in favor of hoping for the best. Fortunately we didn't end up paying for this gamble in anything but lost sleep. It could easily have sunk the project.

3. No feature approval by design department.

When the programming team ramped up in late August 2002, a number of key design documents were incomplete due to a short-staffed design team, meaning that programmers started working on features before they had been fully designed. We had a strong idea of what had to be done, but without a formal design document the margin for error became unmanageable. We had many meetings to try to reconcile the most critical aspects of the game, but in reality programmers were completing features before their design had been finalized. The greatest impact, however, was the lack of a formal review process for new features. All too often a feature was deemed to be complete when it worked on the programmer's test map with default tuning values. As you can imagine, such assump-

tions came back to bite us time and again.

One feature, the ability to harvest resources from dust clouds, had to be cut when we realized too late that it suffered from significant mouse selection issues. Completed in February, our Unit Cap system wasn't put through the paces until June, a mere eight weeks before release. We had no choice but to implement the system used in *HOMEWORLD* or suffer setbacks to game balance and AI.

Any request made by the design department needs factoring-in time for a proper review or it's at risk of being cut in late stage development. Our lack of formal design review not only wasted art and programming time, it negatively affected gameplay implementation, since some features never made it into the game.

4. Parallel development. Parallel development almost crippled the project before it got started. For most of the last 18 months, all three departments grappled with an aggressive schedule while clamoring for more resources. In fact, the first six months of development occurred with a skeleton crew, frequently putting various departments behind the eight ball. For over three months the lead designer was the only designer on the project, resulting in an understaffed design team mired in conceptualization while art and programming started full production. This significantly delayed single-player and multiplayer game development. Certainly it was not a good situation to be in, but things would get worse before they got better.

With *HOMEWORLD 2* back in active development, Relic had for the first time grown into three full production teams. The largest of these was *IMPOSSIBLE CREATURES*, which was in its final stages at the time and taking most of the company's resources. Development of *HOMEWORLD 2* had been planned around the established pipelines, completed tools, and functional code base left over from the original development team. And so a small group would begin production with the idea that it would soon be reinforced by most of the *IMPOSSIBLE CREATURES*



The mission backgrounds were created in Photoshop and then applied to the inside of a sphere. Once the texture was applied, the background tool would turn it into perfectly Gouraud-shaded polygons.



The resulting look is very close to the original Photoshop image without using any texture memory. An artist hand-corrected the spherical distortion and seams in the polygons before exporting it into the game.

crew when that game shipped.

None of this turned out to be true. Many HOMEWORLD 2 systems considered finished needed substantial work and in some cases had to be scrapped entirely. This situation severely taxed the already understaffed programming team and turned an aggressive schedule into a nearly impossible one. And instead of IMPOSSIBLE CREATURES staff moving onto HOMEWORLD 2 within three months, many didn't arrive until six or seven months later, if at all. While this affected all three departments, design was the hardest hit.

Realizing the company was spread too thin to support three full projects, Relic made the responsible choice and shut down the prototyping group, moving the entire team over to HOMEWORLD 2.

5. Quality assurance. Most development teams probably have something to say about their QA, but for us a number of issues came up that turned a challenging situation into a horror show. Our QA team did an amazing job with the resources available to them, and the game would certainly never have shipped if not for their tireless efforts and unfailing dedication. That said, our introduction to the QA process began with our entire team being fired and then rehired soon after on temporary contracts (as a result of internal reorganization at Vivendi Universal Games). In the following months our QA lead was replaced three times, and in each instance project ramp-up took valuable time. Most problematic of all, we had restricted access to the test plan, which put us in a position where we had no visibility into when or how systems were being tested. For example, our compatibility testing didn't begin in earnest until the final six weeks of production. As a result, we were fixing OS compatibility bugs after our first gold candidate delivery.

There were a number of ways we were able to work around these roadblocks, however. The first was

to conduct focus group sessions for gameplay and UI. The first session was held two weeks prior to E3 and gave us a wealth of feedback that proved invaluable in polishing our E3 demo. We held two more focus sessions in the final months to identify which areas needed the most polish and which worked the best.

The second (and most valuable) way that we managed our QA risks was to run an internal bug-tracking system separate from our QA database. Game crashes were always dealt with as soon as they occurred, and the remaining tasks entered were dealt with expediently. It's an ill-advised solution, but in this case it was one that worked.

Evolve or Die

Hot on the heels of HOMEWORLD's success, HOMEWORLD 2 initially focused on reinventing the RTS genre all over again. Concepts like terrain, vector-based attack systems, even time modulation mechanics (creating a 4D RTS, as it were) came into play and in some cases were at the core of development. As a result, a lot of the things that made HOMEWORLD so much fun fell by the wayside, and HOMEWORLD 2 stopped being a sequel.

Six months later we went back into development with a new focus: evolve the

HOMEWORLD experience rather than revolutionize it. Every decision we made, right and wrong, came from this principle. On paper HOMEWORLD 2 was in development for nearly four years, but in many ways the game was created in the final 15 months. We set out to make a worthy sequel to HOMEWORLD 2, and that is exactly what we accomplished.

It's our third title, our first sequel, and has proved second to none as the best experience of our professional careers. 🙌



A pair of Hiigaran Pulsar Gunships assault a Destroyer

The Quiet World of Non-Disclosure

Words can kill. For game developers, the most costly weapon a publisher wields today is the word "non-disclosure."

Illustration by Steve Munday



use this information to negotiate better contractual terms. In other words, because non-disclosure agreements prevent developers from comparing their deals, they may be settling for less than what they might have effectively negotiated otherwise.

If game developers were suddenly able to compare their deals, publishers might be put in a difficult position, having in some cases to justify why more competent developers were being paid less than inferior ones. You'd be surprised at how often this happens.

David Cornwell, President of DNK Cornwell (a former NFL counsel and attorney for sports agent Leigh Steinberg), agrees. Cornwell believes that the public disclosure of salary information has been instrumental in negotiating higher compensation for sports players. By using salary information as the basis for their discussions, sports agents have negotiated record-breaking deals. For teams, the value of a publicly disclosed deal demonstrates their commitment to winning, and for players it's a badge of honor reflecting their contribution to a team's success.

How does this apply to game developers? Despite earning millions of dollars, today's hottest game developers may be leaving millions more on the table simply because they don't know what their competitors are earning. Recently I was speaking with the president of a leading independent game company about this. When I asked him how he would feel if he discovered that another company were earning a 2 percent higher royalty, he immediately understood the point I was trying to make. For him, a 2 percent higher royalty could mean millions of dollars in additional revenue. But without knowing what his competitors are earning, he has lost this strategic bargaining chip.

To help you understand what is really at stake, let's do a little back-of-the-matchbook math. Since our developer is the creator of a series that has sold more than 8 million copies, we know that his publisher's gross revenue is likely between \$150 and \$200 million. Based on this, our developer would have earned around \$15 million, give or take a million or so. Not bad, right? But had he negotiated 2 percentage points higher

continued on page 71

A non-disclosure agreement is a promise between a publisher and developer not to discuss the details of their agreement, and most assume that this is simply an assurance that developers will not leak game information to competitors or the press.

But the true impact has less to do with a developer's propensity to spill the beans and more to do with how a developer might

continued from page 72

initially, he would have put an extra \$4 million in his pocket.

Let's look at it another way. If his publisher generated \$200 million, one could reasonably argue that a competing publisher might be satisfied with \$180 million, right? So if our developer then signs his next deal with the second publisher and the product does equally well, then he's better prepared to negotiate for that \$20 million difference. But is it really possible to negotiate this kind of a deal?

A few years ago baseball star Alex Rodriguez signed a record-breaking \$252 million deal, making him the highest-paid player in baseball history. Rodriguez is receiving 10 times more money than the highest-paid player received a decade earlier. Why? Because Kevin Brown was earning nearly that much the year before, that's why. Because Alex Rodriguez's camp could argue that Alex was a more valuable player based on an existing benchmark. And the only difference between baseball players and world-class game developers is the existence of that benchmark. Sports players talk openly about their compensation, and teams compete with each other for the most talented players based on this information.

Today's hottest game developers may be leaving millions more on the table simply because they don't know what their competitors are earning.

It's highly unlikely that we'll wake up tomorrow to find the details of Electronic Arts' latest development deal posted on a web site. On the other hand, there are promising indications that one day we will be able to talk openly about compensation. Recently at ECTS I met with one of the top business development executives in our industry. Over a dinner of Texas-style barbecued ribs (yes, you can find them in London) we discussed the future of independent

developers. In confidence he admitted that when you consider their creative contribution and the risks they take on behalf of their publishers, independent developers should be paid more.

Given the competitive nature of our business, getting to the point where we can more openly discuss contracts may take more time. Then again, we all know that Atari recently purchased Shiny for \$47 million, based largely on the anticipated revenue of a single license. The real news about this deal is not the purchase price, but the fact that we all know about it. 🐾

DAN LEE ROGERS | *Dan is president of BizDev Inc., a leading business management firm in the interactive industry. E-mail him at dlr@bizdev-inc.com.*
