GAME DEVELOPER MAGAZINE

JUNE 2001

Return

Open Your SDKs

In recent years, the number of possible development platforms you can work on has gone through the roof. It's a bit crazy, but it certainly is nice to have options. Here's a short list of platforms you could create games for: Xbox, Playstation 2, Gamecube, Indrema, Game Boy Advance, Palm, Windows CE, WAP, BREW, Java for Wireless, Synovial, FOMA, WildTangent, Java for Web, Real Arcade, Macromedia Shockwave, Adobe Atmosphere, Windows, Macintosh, Linux, and there are many others.

The recent IDSA "State of the Industry Report 2000–2001" reveals that in the American market, roughly 60 percent of people regularly play interactive games. But most people probably only have one or two of the platforms above. How can you maximize your audience?

Choose. When planning out your new title, the first choice that you'll likely make is which platform category. Which platforms would be capable of providing the experience you envision, and how long will it take to develop for that platform? For example, if you're an independent developer you can probably write a Palm game on your own, but an Xbox title would require too much time to do well.

After identifying whether you're targeting handhelds, consoles, the web, or what have you, you'll consider money (these two decisions are actually interwoven). What is the licensing arrangement on the specific platforms you're looking at? How many units have shipped to the public?

Of course I'm greatly simplifying things, but after considering the platform category, licensing arrangements, and possibility for sell-through, you'll need to examine how difficult it is to develop for the platform as well as how easy it will be to make your title cross-platform. Making sure your game can be cross-platform is a serious consideration, unless you have a sweetheart deal with a particular platform manufacturer. Last month's CHICKEN RUN Postmortem is a perfect cross-platform example, with the game appearing on PC, Dreamcast, Playstation, and Game Boy Color.

Research the SDK. Doing research to determine the technology available on each platform is a challenge. For most platforms you can easily access the available SDKs. But if you're looking at consoles, you're blindfolded. What do the console SDKs offer? What technology is available in the form of libraries that you can take advantage of?

Holding the console SDK as a corporate secret keeps power in the hands of the console manufacturer, not the developer. How can you compare SDKs? For that matter, how might the SDK be improved by the console manufacturer when they're creating it without direct knowledge of what their competitors are doing?

The open flow of information encourages innovation and competition. If you're working with a console manufacturer, encourage them to share their SDKs more freely. You'll wind up with a better SDK as a result, and you'll be able to choose your platform with eyes wide open.

This Month

Keeping in this spirit, our primary feature this month details how to make sure your game will support Mac OS X. Tim Wood from The Omni Group has ported a number of marquee titles over to OS X. He shares his suggestions on how to make sure your game is easily portable to support the Macintosh.

I'd also like to introduce a new face to the magazine this month: Tito Pagán, senior 3D artist/animator from WildTangent, is joining us in the Artist's View column. Tito has many titles under his belt and a unique perspective on level design due to his architectural training. Mark Peasley has just entered crunch mode on his project at Gas Powered Games, so he and Tito are sharing the column for a while.

Our Postmortem this month is the much-anticipated PC title BLACK & WHITE. Get Peter Molyneux's take on what went well and what went poorly during the design of this massive game.

Mark

GameDeveloper

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

Publisher
Jennifer Pahlka *jpahlka@cmp.com*

EDITORIAL

Editor-In-Chief
Mark DeLoura *mdeloura@cmp.com*
Senior Editor
Jennifer Olsen *jolsen@cmp.com*
Managing Editor
Laura Huber *lhuber@cmp.com*
Production Editor
R.D.T. Byrd *tbyrd@cmp.com*
Editor-At-Large
Chris Hecker *checker@d6.com*
Contributing Editors
Daniel Huebner *dan@gamasutra.com*
Jeff Lander *jeffl@darwin3d.com*
Tito Pagán *tpagan@wildtangent.com*
Advisory Board
Hal Barwood LucasArts
Noah Falstein The Inspiracy
Brian Hook Independent
Susan Lee-Merrow Lucas Learning
Mark Miller Group Process Consulting
Paul Steed WildTangent
Dan Teven Teven Consulting
Rob Wyatt The Groove Alliance

ADVERTISING SALES

Director of Sales & Marketing
Greg Kerwin e: *gkerwin@cmp.com* t: 415.947.6218
National Sales Manager
Jennifer Orvik e: *jorvik@cmp.com* t: 415.947.6217
Senior Account Manager, Eastern Region & Europe
Afton Thatcher e: *athatcher@cmp.com* t: 415.947.6224
Account Manager, Recruitment
Morgan Browning e: *mbrowning@cmp.com* t: 415.947.6225
Account Manager, Northern California
Susan Kirby e: *skirby@cmp.com* t: 415.947.6226
Account Manager, Western Region, Silicon Valley & Asia
Craig Perreault e: *cperreault@cmp.com* t: 415.947.6223
Sales Associate
Aaron Murawski e: *amurawski@cmp.com* t: 415.947.6227

ADVERTISING PRODUCTION

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

GAMA NETWORK MARKETING

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

CIRCULATION

BPA INTERNATIONAL
Game Developer is BPA approved

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

SUBSCRIPTION SERVICES

For information, order questions, and address changes
t: 800.250.2429 or 847.647.5928 f: 847.647.5972
e: *gamedeveloper@halldata.com*

INTERNATIONAL LICENSING INFORMATION

Mario Salinas
t: 650.513.4234 f: 650.513.4482
e: *msalinas@cmp.com*

CMP MEDIA MANAGEMENT

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

GamaNetwork
A DIVISION OF CMP MEDIA LLC

WWW.GAMANETWORK.COM

Return

SAYS YOU

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

Writer Has Suggestions for Dual Specialty

I very much appreciated Richard Rouse's recent Soapbox article in *Game Developer* ("What Ever Happened to the Designer/Programmer?" April 2001). Rouse brings up a point I seldom even hear mentioned these days, to my considerable sadness. Our industry definitely does very little to encourage the designer/programmer dual specialty — in my own early career I experienced considerable resistance to even the idea, let alone the reality. I can remember being told on my first day at my first industry job that as a programmer I would never do design work at the company — that I eventually proved this prophecy incorrect was the result not only of years of hard work but also of considerable dumb luck.

From my own experiences I can also mention an additional reason I believe designer/programmers are a rare breed these days — because more senior mono-specialty programmers and especially designers perceive young would-be designer/programmers as a threat. Even at companies founded by successful designer/programmers this can be a problem. I concur with Rouse's assessment: a young designer/programmer with talent in both specialties has great advantages over his peers and is bound to advance rapidly. This potential creates "political problems" which can be quite intractable.

Another quite difficult problem: how to successfully train young developers to the dual specialty? I can suggest that young programmers take college courses in the humanities — history, writing, and so on, and that young designers teach themselves C++. On the job I can also suggest mentoring by senior developers in each specialty. At my own company, Big Huge Games, we encourage all of our programmers to learn and exercise their "game sense" and all of our designers to do as much coding work as possible. Not a panacea, perhaps, but at least a step in the right direction.

These problems remain conundrums, but the potential reward is considerable. Thank you once again for bringing this issue to everyone's attention!

Brian Reynolds
President, Big Huge Games
via e-mail

Rating System Flawed

In regards to Mark DeLoura's editorial (Game Plan, "Rating Systems," March 2001), I thought I would tell you about our own experience with the ESRB. When we were preparing to release KING OF DRAGON PASS, we got a "T" rating, which seemed reasonable (the game has mature themes that probably wouldn't appeal to younger players, for example, getting married). But they gave as the content descriptor "Animated Violence." We were astonished, since KING OF DRAGON PASS uses no animation. We decided to appeal the rating and figured that it was easier to get a harsher rating than a tamer one (and didn't have the time to go through a protracted process), so we asked for a "Realistic Violence" descriptor, which we were granted. I believe the ESRB scheme has some major flaws, and this is one of them.

David Dunham
A Sharp
via e-mail

Kudos for "Heads"

Gavin Moore's article in *Game Developer* ("Talking Heads," March 2001) was truly excellent. I've been working closely with Keith Waters (one of Moore's references) on the LifeFX facial animation system, and we were both impressed by the flexibility of the Talking Heads system and the speed with which Sony engineered it.

Dan Teven
Systems Architect, LifeFX
via e-mail

Send e-mail to editors@gdmag.com, or write to *Game Developer*, 600 Harrison St., San Francisco, CA 94107

Kludge by Tiger Byrd and Daniel Huebner

Hey, you're back from E3! Did you have a good time? Did you see anything memorable?
Memorable?
You spent the whole time drunk and you can't remember anything?!?!
My swag shirts assure me I had a good time.
Booth Babe Inspector

Return

INDUSTRY WATCH

daniel huebner | THE BUZZ ABOUT THE GAME BIZ

Japanese Xbox plans. Microsoft chairman Bill Gates used an address at the Tokyo Game Show to finally shed light on some of the company's plans for its upcoming Xbox console. Microsoft announced a long-term publishing deal with Sega that will bring several future Sega titles to Xbox. A total of 11 Sega games will premiere on Xbox, with new installments of JET GRIND RADIO FUTURE, PANZER DRAGOON, and SEGA GT leading the slate. Sega has already made similar arrangements for Playstation 2 and Game Boy Advance.

In addition to the landmark Sega deal, Microsoft outlined some of its plans to make the Xbox a success in Japan. The company announced that it is forming an Xbox Japan division that will focus on managing third-party relationships with Japanese game companies as well as handling Xbox sales and support in Japan. The new division will also contain an Xbox Games Production Group with the goal of developing and publishing original games under the Microsoft label. To help Japanese games take advantage of the console's online capabilities, NTT Communications will work with Microsoft to create an Xbox network. The two companies plan to launch the broadband service sometime in 2002, with trials of the online gaming service set to start later this year.

Microsoft also unveiled a unique version of the Xbox controller for Japan. The alternative controller, which will ship with every Xbox in Japan, is slightly smaller than its North American and European counterparts and features repositioned buttons to accommodate Japanese gameplay styles.

Indie games winners. The Independent Games Festival presented six awards over two nights at the Game Developers Conference to honor the cream of the independent game development crop, but it was a single title that garnered most of the honors. The first four awards were presented in the IGF "craft" categories, recognizing games for achievement in specific technical categories. Best Audio went to Denmark's Space Time Foam for its title CHASE ACE 2. HARDWOOD SPADES, from Silver Creek Entertainment, took the prize for Best Visual Art. Best Game Design and Technical Excellence honors

Nexon's SHATTERED GALAXY, winner of the IGF Best Game Design, Technical Excellence, Audience Award, and the Seumas McNally Grand Prize.

went to Nexon's SHATTERED GALAXY, the game that nearly swept the awards by also taking home the Audience Award as well as the Seumas McNally Grand Prize. The company walked away with $1,000 for the Audience Award, and $10,000 and an Intel Pentium 4 workstation for winning the Grand Prize.

Sega chairman dies of heart failure. Sega Chairman and CSK founder Isao Okawa has died of heart failure at the age of 74. Mr. Okawa established CSK in 1968 and acquired Sega Enterprises in 1984, in time turning it into one of the largest videogame companies in the world. Okawa became Sega's president last summer when Shoichiro Irimajiri stepped down to take responsibility for poor Dreamcast sales. Just last January, Okawa said that he would help prop up the sagging company with $730 million of his own money.

Yoshiji Fukushima, a Sega board member since 1996 and the chairman of CSK Corp., becomes the new chairman of Sega. Sega has also named chief operating officer Hideki Sato as its new CEO. Sato, who has been with Sega since 1971, has been responsible for the day-to-day operations of the company as the company's co-COO, a title he shares with Tetsu Kayama.

ATI acquires FGL Graphics. ATI has acquired Sonicblue's FGL Graphics at a cost of up to $10 million. FGL Graphics, formerly part of Diamond Multimedia, develops and markets the Fire GL brand of OpenGL-based graphics accelerators for NT and Linux workstations. Under the terms of the sale agreement, Sonicblue will receive $2.7 million in cash and is eligible to receive further financial consideration of up to $7.3 million, depending on FGL Graphics achieving future performance targets. ATI will acquire FGL Graphics' current contracts as well as the right to use the Fire GL brand name and 35 employees at FGL Graphics facilities in Munich, Germany, and San Jose, Calif. ATI expects the acquisition of FGL Graphics to have a minimal effect on earnings this fiscal year.

ATI posted an adjusted net loss of $26.1 million in this year's fiscal second quarter. The loss, which translates to 11 cents per common share, came on sales of $232.4 million and is in line with projections released by ATI on March 1.

Infogrames posts first-half 2001 loss. Infogrames reported a loss for the first half of fiscal 2001. The company attributed its poor performance to slower game sales caused not by the console transition that has plagued the company in recent quarters but to console shipment delays. The company posted a net loss of $20.7 million, an improvement from a loss of $80.4 million in the same period a year ago.

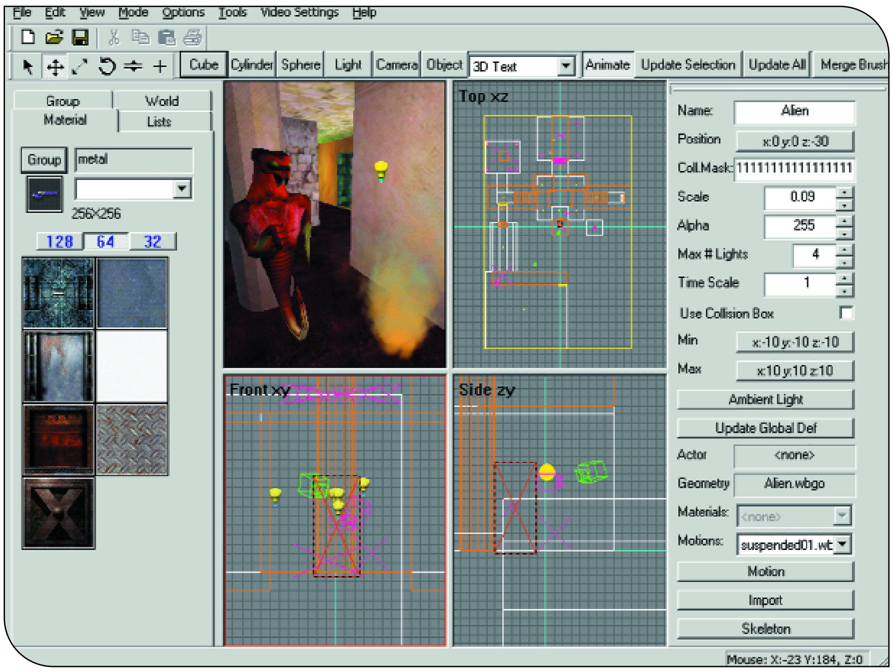
Sales for the company reached $325.4 million, an increase of 26 percent. Research and development costs more than doubled to $50.9 million as the company prepared more next-generation titles. Infogrames Inc., the unit formerly know as GT Interactive, finally showed a profit in the second fiscal quarter.

CEO Bruno Bonnell said that the remainder of this year will be spent turning its newly acquired Hasbro Interactive unit around and explained that the company faces restructuring costs related to the Hasbro deal of about $20 million.

UPCOMING EVENTS
CALENDAR

SIGGRAPH 2001
LOS ANGELES CONVENTION CENTER
Los Angeles, Calif.
August 12–17, 2001
Cost: $40–$950 (member and student discounts available)
www.siggraph.org/s2001

Return


File Edit View Mode Options Tools Video Settings Help
Cube Cylinder Sphere Light Camera Object 3D Text
Animate Update Selection Update All Merge Brush
Group World Material Lists
128 64 32
Top xz
Front xy
Side zy
Ambient Light
Update Global Def
Motion
Import
Skeleton

Group: metal
256X256

Name: Alien
Position: x:0 y:0 z:-30
Coll.Mask: 1111111111111111
Scale: 0.09
Alpha: 255
Max # Lights: 4
Time Scale: 1
Use Collision Box
Min: x:-10 y:-10 z:-10
Max: x:10 y:10 z:10
Actor: <none>
Geometry: Alien.wbgo
Materials: <none>
Motions: suspended01.wt

Mouse: X:-23 Y:184, Z:0

WildTangent's WT Studio interface.

WildTangent's Web Driver 2.0

by tom meigs

Alex St. John, the man behind DirectX, has a vision for the future of web content. Whether or not his company, WildTangent, will become the authors of the delivery mechanism and creation tools of choice for 3D on the web will depend on how they direct their efforts right now.

For those of you unfamiliar with the WildTangent Web Driver, it's definitely time to give it serious consideration for your web game needs. With Web Driver 2.0 now officially released, we can all expect to see a new wave of content hitting the web developed using this springboard technology.

O.K., What Is It?

Web Driver 2.0 is a complete "all-in-one" system for web-based content development. Simply put, you can build games (our primary concern) along with many other types of content including, but not limited to, audio visualizers and some surprisingly engaging interactive web ads. As we can all see from some of the high-profile games constructed using the WildTangent technology which now reside on Microsoft's Zone, naysayers will find proof that this SDK has the power to deliver CD-quality content over the web. Heresy, you might say. Well, kind of.

Actually, the "magic" is accomplished by an impressive streaming system that allows for those objects and support code which are needed first in a scene to come down even on a limited bandwidth connection while the remaining dependent assets stream down in the background. The ability to control this process via scripting in Java or JavaScript makes for pretty easy handling on the coding side. And it no doubt helps that the WildTangent developers include former members of the venerable team behind DirectX, providing them with an intimate understanding of this multimedia paradigm.

What Are the Components?

Web Driver 2.0 SDK Full (about 16MB). This is the complete development package available on the WildTangent web site. It includes tools, documentation, demonstration code, and tutorials about how to create content for the Web Driver.

WTStudio (about 3MB). This is a world builder/editor allowing the user to create familiar BSP geometry to construct a scene.

3D Studio Max Exporters (165KB). These exporters, when copied into your 3DS Max plug-ins folder, will export geometry and motion information into a format native to WildTangent and can then be loaded into WTStudio or passed off to your Java game engine.

WildCompress (512KB). When creating content for a scene with various types of media (such as models, textures, and sounds), you can use this compression utility to convert assets for use with the Web Driver.

Our Web Driver Content Experience

Almost a year ago, our team started looking for ways to expand the kind of 3D game content being done anywhere on the web. After some quick research, we set out to "test the legs" of the Web Driver by developing a series of applications, each an internal stepping stone to the next.

We built a small-scale first-person exploration title intended to test all aspects of a production cycle using the WildTangent technology. WTStudio handled the interiors quite well, with some fantastic visuals available via procedurals. Interiors are a strong point for this tool. At times, since we were trying to re-create the visuals of a Disney property faithfully, we really needed the ability to get into the geometry and grab control points to accurately build certain organic shapes that could not be done very well using merged brushes. In its present state, WTStudio has fairly limited primitives with which to craft a scene, and importing huge actors created in 3DS Max (such as domes or porticos) is a recipe for slow motion.

✱✱✱✱✱ excellent
✱✱✱✱ very good
✱✱✱ average
✱✱ fair
✱ don't bother

Game example created with WildTangent's Web Driver 2.0 SDK.

For our first deliverable, we needed to get a hero character up and running through an indoor/outdoor environment with simple switch-based puzzles and pickups driving an inventory. This cycle would include tests on the Web Driver, WTStudio, and other tools still being completed as we began along with the driver's performance on minimal bandwidths (28K) and its performance viability through America Online. Working with WildTangent to try to shoot down bugs fast, yet competing constantly for internal Java engineering support, this exercise proved to be our first valuable lesson.

We struggled a bit initially to hand off art correctly through the 3DS Max 3 exporters. There were some obvious bugs, some more subtle but no less irritating. Mapping information on models was not always retained and exported correctly, and our models would often import with lighting values blasting away surface information.

Finally, based on the reasonable success of our first "demo" title and the successful presentation of our next title running through America Online and, incidentally, loading faster than some of our Flash-based content, we gained management approval to build out a complete title. Would our 3D application still load faster than the Flash content? Yes.

Our Experience with the Tools

WTStudio is one of the best stand-alone interior level editors any member of our team has ever used. It has a great lighting system, nice particle abilities, and includes on-the-fly scaling of cut brushes. In the unanimous opinion of our team, further development of this tool in particular needs to continue full-force as a critical point of strategy for WildTangent. For content-driven companies like ours, becoming the 3D answer to Flash on the web would help both WildTangent and developers alike.

Having Java and JavaScript code hook into the Web Driver lessens the demand for heavy and constant C++-based engineering support and is a smart move for a web development environment. This forms an advantage for web and game hybrid developers, and the exposure of WildTangent's games through partnerships with Microsoft, Toyota, Radio Shack, and others demonstrates the kind of stability and presence many companies look for when deciding to build content using WildTangent.

However, there were significant delays in shipping the final Web Driver, and WTStudio remains in a state of constant "feature flux." For content to be rapidly prototyped and thrive in a development atmosphere, there is too much dependence on having experienced Java game engineers available. A stronger, more feature-robust WTStudio with more built-in Flash-like "drag and drop" scripting abilities and a variety of geometry prefabs would allow content providers to build simple game and interactive functionality with minimal code support. For those with strong Java resources available, this built-in scripting ability could be enhanced, extended, and customized into entire feature libraries.

Competitors in the general 3D-web arena, such as Adobe, Viewpoint, Pulse, and Cult3D, are making inroads with strong 3D character- and product-based tools and current exporters supporting multiple 3D packages. If WTStudio could be enhanced to handle very basic character setup, include the ability to tag product models with hot spots, and implement mouse controls for viewing, WildTangent would more successfully be able to fend off competitors as a complete solution. Right now, WildTangent only supports 3DS Max 3 with no Maya exporters. Some competitors already support 3DS Max 4 and Maya.

Bottom Line

WildTangent is a solid choice for bringing 3D content to the web. They are a first-rate technology company actively engaged in pursuing multiple venues for their technology. Hopefully, this multi-attack approach will not fracture their ability to become a premier 3D web content solution. From a game developer's perspective, a more powerful and robust WTStudio will enable developers to make the kind of content that will give people a reason to acquire the driver and pass it on to others via sites that use WildTangent. To be sure, some fantastic games can be made, and there is already good evidence of this. Beyond that, the "adver-tainment" potential of the driver and its toolset are perhaps one of the most interesting growth areas to watch. Get familiar with WildTangent now, as they stand a very good chance of leading the way.

WEB DRIVER 2.0 SDK ✱✱✱✱

STATS

WILDTANGENT
Redmond, Wash.
(425) 497-4500
www.wildtangent.com

PRICE

Free until developers or publishers post or distribute content in a commercial format. Then technology licenses are available on a per-usage or per-product basis.

SYSTEM REQUIREMENTS

PC: 233MHz Pentium, 32MB RAM, Windows 95/98/2000 or Windows NT 4.0; 16-bit color display adapter; 28.8Kbps Internet connection; Internet Explorer 4+ or Netscape Navigator 4+; DirectX 5.0 or higher.

PROS

1. WildTangent has solid developer support and well-placed strategic partners.
2. WTStudio could and should become the web's 3D answer to Flash.
3. Java entry into SDK opens game crafting to a wide audience.

CONS

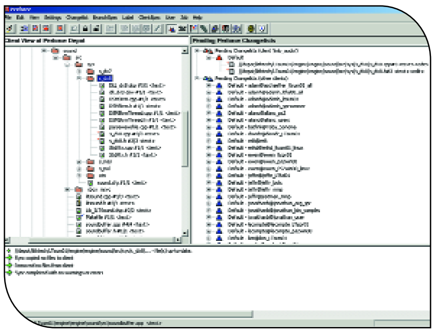
1. "Feature flux" in WTStudio makes content feature promises unreliable.
2. Strong competition from rivals.
3. WildTangent's web focus may be spread too thin.

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

PERFORCE

by james boer

Software configuration management systems, more commonly referred to as source-control systems, are a necessity in today's software development projects.


Perforce is split into a file tree display, a multi-function display, and a scrolling status window.

With a complex 3D game engine being developed simultaneously for three different platforms in quarterly release cycles, engineers at Lithtech wanted an industrial-strength but simple-to-use tool. The product that was deemed best suited to our particular demands was Perforce, which happens to be both the name of the product and the company that creates it.

Because we are developing an engine for multiple platforms, it's a requirement that any source-control system be multi-platform. Perforce handles this in spades, as the list of supported operating systems is quite impressive. The Perforce server runs on Unix or Windows NT, and client programs are available for most flavors of Windows, Unix, Linux, Macintosh, QNX, BeOS, and OS/2.

Fortunately, just because this is a cross-platform product doesn't mean you have to suffer with an antiquated command-line interface. While there is a command-line client available, most users will be using the Windows-native (P4Win) or web-based (P4Web) user interface, greatly reducing the learning curve. The Windows UI is comparable to many other source-control products, both in functionality and in ease of use. The main window is split three ways, with the file tree displayed on the left, a multi-function display area on the right, and a scrolling status window on the bottom.

While the Windows client is a huge step up from a command-line interface, there are a few annoyances, as well as some obvious holdovers from Perforce's command-line heritage. When setting up client information, paths connecting the file depot (the database) and the local hard drive must be set up in a nonintuitive command-line fashion. The worst part of this is that Windows users may not be aware of the fact that Perforce treats all directories and file names as case sensitive, an obvious requirement for Unix and Linux development. But there are no clues for the hapless Windows user as to why the paths they're typing in don't seem to work.

But despite a few small nitpicks, Perforce really shines where it counts. File operations are blazingly fast and the database is reliable and space-efficient. In addition, Perforce works in a different manner from simple file-based systems. Files are grouped together in numbered "change lists" consisting of modifications, additions, and deletions of individual files. This makes it easy to track single feature-based changes or bug fixes instead of having to scan the entire database for similar check-ins.

In addition to a friendly Windows front-end, Perforce has several other options for use. It integrates with Microsoft Visual Studio, Metrowerks' Codewarrior, IBM VisualAge, and others for quick access to basic functionality from within your work environment. It has a clever web-based interface as well, meaning that it supports nearly every platform that can display a web page. Frankly, I haven't used the web interface for any real work, but I did check out a sample database on Perforce's web site, and it seemed fairly simple to use. The integration with Visual Studio is generally solid, but there are annoyances here as well. Perforce insists on altering project and workspace information when no changes have been made, and clicking through warning dialogs becomes tiresome.

Overall, despite some minor issues, Perforce is a robust and solid-performing source control tool for game developers, particularly if you have multiple projects with shared code or do any sort of cross-platform development work. Perforce is $600 per user for 20 or fewer users and costs less with more seats purchased.

★★★★ | PERFORCE
Perforce Software
www.perforce.com

CALIGARI'S TRUESPACE 5

by david winter

TrueSpace 5 is the latest in a line of mid-range 3D modeling and rendering packages developed by Caligari. The interface is far different from your typical 3D graphics application. Modeling tools are provided via a large number of "fly out" button bars. You can use a four-view screen as used in other 3D packages, or you can use the traditional TrueSpace workspace. After fumbling with the version 5 configuration for a short while, I found myself changing back to the version 4 configuration.

Since version 4, TrueSpace has offered modeling in wireframe, or solid view via Direct3D. Version 5 has improved the performance of this feature significantly. It is now just as fast to use the solid view as it was to use the wireframe view in version 4. I am currently using TrueSpace 5 with DirectX 8 running on a Windows 98 Pentium II with an ATI Radeon card and have had no performance issues.

Most of the modeling tools in TrueSpace are designed to create smooth, detailed objects by increasing the polygon count. These tools (NURBS, Quad Divide, and Meta-Balls) are almost never used for building game models. In general, the best way to create polygonal game character models is to draw a flat polygon and then sweep it. Scale the top plane, then sweep it to the next contour and repeat. Using this method, you will easily be able to build game models in the 600 to 1,200 polygon range.

TrueSpace offers a bones system and a jointed IK system for animating. Add in the Puppeteer plug-in and you have a reasonably good set of tools for animating.

TrueSpace does not offer game-developer-friendly texturing tools. It does offer a 3D painting system and standard UV mapping, such as planar, cube, and so on. However, it is very difficult to create textures optimized for games. If you use the 3D texture painting tools, each face is saved as a separate bitmap. Imagine your game engine trying to load 800 bitmaps to texture one character. The ability to position a face's vertices in a 2D texture bitmap while updating the 3D display should be a must for TrueSpace 6.

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

File formats are TrueSpace's second biggest fumble. You can export to Microsoft's .X format. That's great, but the problem is that TrueSpace 5 was released early in 2001 and Caligari is still using the format specifications from January 1997. There is no provision for skinned meshes; you had best build your characters with IK groups rather than with the superior bone or skin system. TrueSpace does not support animated 3DS Max files.

There is also a huge bug in their export tool in that it flips all the texture UV values. If your character wears jeans and a T-shirt, his fly will wind up down by his ankles, and the collar of the shirt will be around his waist.

Like 3DS Max, TrueSpace can use plug-ins. Fortunately there are plug-ins which can correct most of the .X export shortfalls.

In conclusion, most of my projects have budgets ranging from "nothing at all" to "not a heck of a lot." Purchasing packages like 3DS Max and Character Studio are out of the question — purchasing the 3DS Max tools would cost about $7,500 Canadian. It's important for there to be another option for development houses that don't have the budgets of Microsoft or Electronic Arts. At under $1,000 (U.S.), Caligari's TrueSpace 5 offers a very affordable alternative to the 3D Studio/Character Studio combination if you don't mind the need to work around some issues.

For those on a really tight budget I would recommend purchasing the older version 4.3. It is under $400, and version 5 does not offer the game developer any features that are worth the extra cost.

Caligari has made it known that they wish to work with the game development community to enhance their product. Adding game-developer-friendly texture tools and the ability to export correctly to modern game formats would be a good starting point.

★★★ | TRUESPACE 5 |
Caligari | www.caligari.com

NVIDIA'S GEFORCE 3

by brian sharp

Along with a speed boost over existing cards, Nvidia's new GeForce 3 graphics card boasts an extraordinary number of substantial new features. It includes support for curved surfaces, programmable vertex shaders, more textures in more formats, increased internal precision, and expanded fragment-blending capabilities.

The curved surface support is a new feature for the GeForce 3, not present in the GeForce 2. The GeForce 3 supports the industry-standard Bézier patches, both triangular and quadrilateral. It accepts parameters independently specifying the number of vertices along each side of a patch, which makes continuous level-of-detail adjustment possible. As the parameter is specified in floating point, it helps curb "popping" in patches as they change density over time. The implementation seems well thought-out and eminently useful for games using curved surfaces, as Bézier patches are a very common curved surface representation, supported in most major 3D modeling programs.

Programmable vertex shaders are among the most impressive new features. A game sends microcode programs to the card at run time which are executed to process vertices. Programs can scale anywhere from doing nothing at all to applying a variety of application-specific effects using the orthogonal instruction set and temporary storage space. The main limitation on vertex program size is speed — the longer the program, the fewer vertices processed per second. Nonetheless, the card is fast enough to handle powerful tasks at good speeds. Programs cannot contain loops, but since the programs are short, looping isn't needed. The engine can neither create nor destroy vertices, which could limit clever schemes to generate vertices in the shader to reduce bus traffic. Overall, though, vertex shaders on the GeForce 3 are orthogonally designed, well implemented, and have a lot of potential for use in games.

The card sports more power in the area of textures: a single rendering pass can reference up to four textures and in some new flavors. For size, texture dimensions are no longer limited to powers of two, and as for formats, the card supports a signed RGBA texture format (where each component is an 8-bit fixed-point signed value) and the "HILO" format, either signed or unsigned, where each texel is made up of two 16-bit components. The latter is particularly useful for textures whose values represent arbitrary vectors needing greater precision or range.

The card supports up to eight stages of fragment blending, where each stage can apply one of a handful of operations to incoming values and pass the result on to the next stage. Operations include basic arithmetic as well as limited dependent texture reads (where one texel is used as a vector to shift the lookup into another texture) and dot products. Computations at each stage are done with increased precision and in floating point, which goes a long way toward reducing blending artifacts. With all of that, though, the fragment-blending pipe does leave a few things to be desired. Unlike the extremely general vertex shaders, the fragment-blending capabilities are still very fixed-function, and the specialized operation set and small amount of scratch space keep the fragment-blending pipe from being generally programmable. Where the vertex shaders offer fertile ground for experimentation, the fragment blending is nowhere near as versatile. For the set of effects it is capable of, though, it does them very well and is a significant improvement over other current cards' capabilities, and so while it begs to do more, it certainly does a lot as it is.

The GeForce 3 has other assorted additions and features, but the above are the most significant features to be aware of as a game developer. The card is not without its limitations and pitfalls, and as a developer it's important to be aware of them ahead of time to avoid any unpleasant surprises down the road. Those pitfalls are small detractions from a generally superb piece of hardware. The features, power, and versatility of the card give game programmers some serious potential to play with and the opportunity to take a substantial leap in our games' visual quality.

★★★★★ | GEFORCE 3 |
Nvidia | www.nvidia.com

Return

Louis Castle

Helping Westwood Studios Command & Conquer

Westwood's general manager, Louis Castle.

Louis Castle co-founded Westwood Studios in 1985 with Brett Sperry in Las Vegas, Nevada. They shared their first office in Louis' converted garage and have been profitable ever since. Westwood now has approximately 250 total employees, making it one of Electronic Arts' largest subsidiaries. *Game Developer* caught up with Louis at the Game Developers Conference, where he frequently gives lectures on company culture and the business of game development, to talk about his background and his philosophy of how to run a game studio.

Game Developer. When you were in college, you used to spend time converting cubist paintings into hexadecimal and entering the data into tables to re-create them on your computer. If you hadn't found developing games as your outlet for this unusual proclivity, what do you think you might be doing today?

Louis Castle. I suppose I'd be an architect who was very savvy with new computer technologies and movement through spaces to sell the concepts, probably running a firm and working in the high-tech side of architecture. But once I got bitten by the bug of doing programming, I found it so satisfying as an artist to be able to see my work appear and to animate things myself. I turned down quite a few scholarships to architectural colleges to pursue fine arts and computer science at a local university.

GD. Did you and Brett meet in college?

LC. No, we met in Las Vegas. He was working as a freelancer for an educational software company. I was selling computers and doing some freelance artwork and such, and we got to know each other there. There was a small community in Las Vegas of game developers who would get together and gripe about how most people running companies didn't care about the product. They were just looking to exploit the talent and turn a quick buck.

GD. That sounds kind of familiar.

LC. That's why we started Westwood. Brett and I were acutely aware that we didn't want to work for people who didn't care about the products they were building. Brett had a lot of visions of building the company and growing it to an enormous size, but all I really wanted to do was make sure that the environment that I came to work in every day I really enjoyed, and that all the people there had similar goals to mine. Even to this day, being the most profitable company is really not what drives me. I just want to make great games that people love.

GD. How did you arrive at defining Westwood's company culture? Was it something philosophical, or more trial and error?

LC. It's very deliberate and very philosophical, and not communicated clearly enough and frequently enough. I'm always reminded that we need to be better at communicating it. With so many people at such a large organization, it would be very easy to lose touch with people. It's a constant effort to make sure everyone knows they are appreciated and an even greater effort to make sure everyone is happy, and if not, why not. I don't want someone coming to Westwood, spending six months with us, then saying, "This is horrible, I can't believe this company after all the great things I've heard about it." I hate seeing someone have a bad experience in this industry, especially at my company, because this is an industry that's about entertainment. You should want to come to work every day and love it.

GD. That's the basis of your company culture?

LC. That's it. I want people to really enjoy themselves. Many years ago, my partner asked, "What do you want to do with the company?" And I said, "I think I just want to have fun." It sounds very simple, but if you think about it, that's a very hard ideal, if you try to push that through an entire organization and say, "I want everyone in our organization to have fun."

GD. But how do you create an environment where people feel empowered to have fun and also be able to function in their jobs?

LC. Westwood's a work in progress. There will never be a time when all 250 people are smiling and running into the office every day because they're delighted to be working, but if I can get as many of them into that mode as possible, it becomes very infectious.

GD. How do you communicate that to your employees?

LC. The way we do it is very organically, at social events, hanging out with the teams. People ask what you think, and you just go off on a little soapbox. If you don't make a conscious effort of it, once you get to a certain size, people just don't have enough touch time with each of the visionaries at the company.

GD. Without some framework, it's too easy to make game development feel like futility.

LC. I think that happens a lot. A lot of people feel frustrated even within our company. What I tell people is that if you're frustrated, say something right away and surface these ideas. Because the company as a whole really wants everybody to enjoy their position.

GD. No one wants to be the guy to crack and say, "I'm not having fun right now. Actually, we're on a flaming train wreck to hell."

LC. Then what happens is that when the train wreck occurs, everybody says, "Oh yeah, I saw that coming." Well why didn't you say something? But I agree that there's pressure. We have all of the same pressures that everyone else does.

GD. Everybody really seems to like you. Coincidence?

LC. I treat people the way I'd like to be treated. I care a lot about the industry and I want to see it become the next wave of entertainment. Maybe I saw too many Chris Crawford lectures when I was growing up in the industry [*laughs*]. Maybe that's why people find me an affable person to get along with. And I'm not afraid to say, "Hey, we don't get it right every time. We make mistakes."

The Era of Post-Photorealism

Vincent Van Gogh. *Starry Night* (1889).

"When you go out to paint, try to forget what objects you have before you, a tree, a house, a field, whatever. Merely think: here is a little square of blue, here an oblong pink, here a streak of yellow, and paint it just as it looks to you . . ."

— Claude Monet

We have been pushing towards this elusive thing called photorealism, yet we can't really blame the hardware anymore. We have more polygon throughput and greater pixel fill rate than ever before. We can use 32-bit textures pasted all over our imaginary worlds. However, it seems to me that, ironically, as game graphics get more realistic, they're not as interesting. We live in a photorealistic world. I'm used to walking around and looking at things in it — I'm just not sure that I want to play there.

I think it would be interesting to walk through a world as moody and full of emotion as Vincent Van Gogh's *Starry Night*. How would that turbulent sky move? What is over the next hill? How does it look inside the church in the village? Now that would be an interesting world to explore. *What Dreams May Come*, directed by Vincent Ward, gave an interesting glimpse of what it would look like to explore the inside of an impressionist world. This month I am going to explore methods for creating more artistic styles in a game environment.

From the Canvas to the Screen

The most obvious and easiest way to create a scene in a different artistic style is to create more artistic textures. This seems painfully obvious. However, programmers are often prone to solving all problems with technology. While oftentimes the creation of 3D graphics requires a technical solution, the most effective method for achieving the goal is to have a talented artist who understands the vision. While the technology used in a game such as Sega's JET GRIND RADIO enhances the style, the style in the textures clearly dominates the look.

I found this to be very apparent in my research on sketch rendering styles. When trying to create the look of a pen and ink sketch, often a simple texture achieves the look in a very efficient manner. There is also a variety of artistic filters for programs like Adobe Photoshop which can aid in the creation of artistic textures.

Last year some students at the University of Wisconsin modified the OpenGL version of QUAKE to support some artistic rendering styles. Some of the effects were achieved by drawing edge lines in a random fashion, much as I described in this column last month ("Images from Deep in the Programmer's Cave," May 2001). They also worked on a sketch style by combining the edge-rendering routine with some simple pencil sketch textures. The result

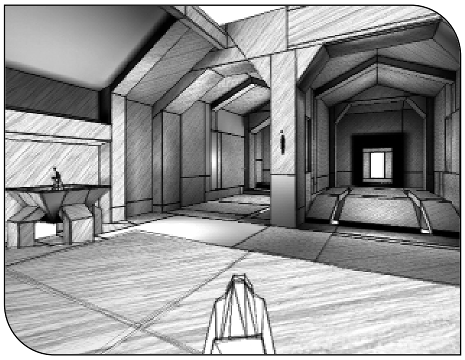


FIGURE 1. Sketch-style QUAKE.

JEFF LANDER | *Realistically, Jeff should be at his desk banging out code for Darwin 3D instead of gazing at the sunset lost in the swirling skies. If he ever snaps out of it, you can reach him at jeffl@darwin3d.com.*

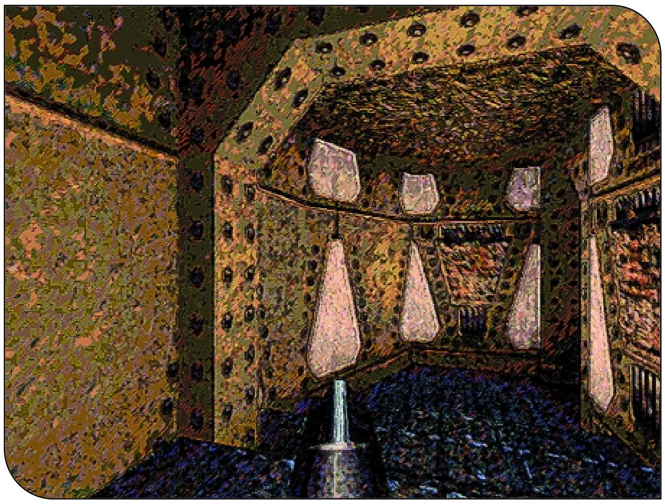
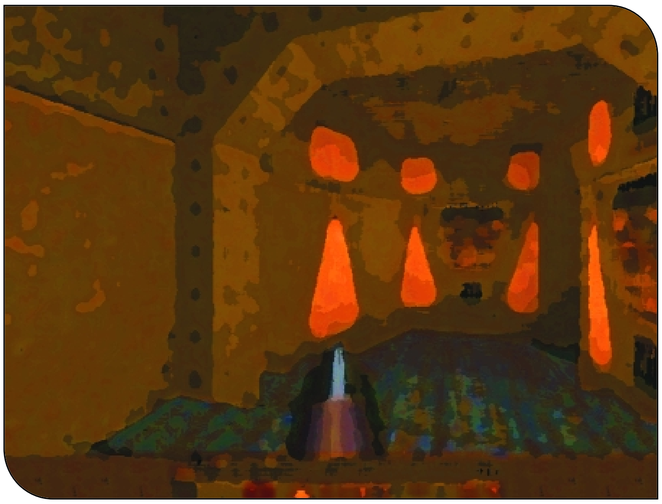



FIGURE 2 (left). QUAKE with Photoshop textures. FIGURE 3 (right). QUAKE with impressionistic textures.

was pretty interesting, as you can see in Figure 1.

The idea of modifying the look of a game by hijacking the rendering engine was pretty interesting. However, what interested me more was how much simple texture manipulation could change the feel of a game. To take this idea a step further, I passed the original QUAKE textures through a Photoshop brush filter. This simple change yields some very dramatic results, as you can see in Figure 2.

When interacting with an environment like this, though, it is apparent that the style is applied as textures to the mesh. The brush strokes align directly with the polygon. The size of the strokes is also a function of distance from the viewpoint. It looks very consistent as you move through the world, but it looks like a creatively textured world, not one that was painted.

Brush Strokes and Bullets

One thing that makes a painting different from a textured rendering is brush size. When we apply a texture of a brush stroke to a polygon and then move it through an environment, the brush stroke starts almost infinitely small when it is far away. As the texture gets nearer to the camera, the size of the brush stroke grows very large.

This is not usually true of a painting. Artists are actually limited to some extent by the fact that painting is a physical process. Paint must be applied with a brush or knife of a finite size. Paint takes up physical space. When a painter draws a tree very near the viewpoint it is usually composed of many brush strokes. When it is far in the distance, that same tree may only require a couple of strokes. This effect can be simulated through the use of level of detail and MIP-mapping. However, in order to actually capture the look of a painting, we really need to look at how an artist works.

Painting is done on a 2D canvas. It is only the artist's talent at perspective and our own minds that create this illusion of depth. In many cases the size of the brush strokes is almost independent of depth. This fact is why many of the 2D image-processing algorithms for simulating painting styles are so effective; take a look at another screen capture of QUAKE. This time, however, I am going to apply an artistic filter within Photoshop (see Figure 3).

You can see how the strokes give much more of an impression of being painted. The brush size is much more uniform, and color blends across polygonal boundaries. Creating continuity from frame to frame is a real challenge with this method though. Since the postprocessing method is applied on the rendered 2D image, there is nothing to keep brush strokes from changing dramatically from one frame to the next.

The people who created the visual effects for the movie *What Dreams May Come* dealt with this problem by tracking the motion of the camera as well as individual patches of color throughout the scene and then using this information to make continuous strokes. They have even released an Adobe After Effects plug-in called Video Gogh for applying this effect to video sequences.

The next crop of 3D graphics hardware may show some form of accelerated 2D image processing. Nvidia recently released a demo of some simple image processing on a 3D scene. The new GeForce 3 graphics chip supports rendering of four simultaneous textures as well as vertex programs and programmable pixel operations. By placing the same texture in each of the four texture units and using the vertex and pixel operations, a simple convolution kernel can be applied to an image. It may not be possible to do complex effects, but simple edge detection and sharpening is clearly possible. It is definitely time to start playing around in the Photoshop convolution editor to see what is possible.

Flowing through the Turbulent Skies

These 2D image-processing tricks are good for creating more artistic textures. However, I am not really any closer to finding a method for generating images like those seen in the *Starry Night* painting. To me, the painting does not express a texture. The work has a profound sense of motion. The sky in particular invokes a feeling of fluids moving and swirling across the sky. I am familiar with fluid animation techniques, and that seems like

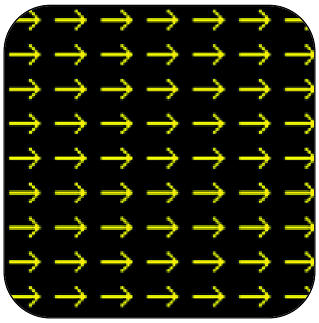
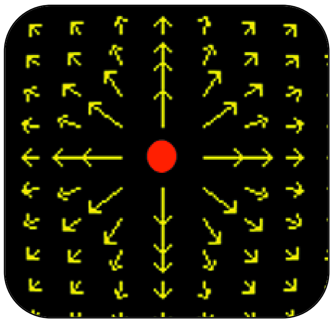
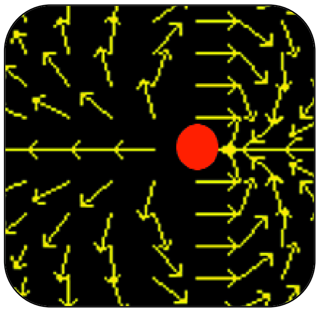
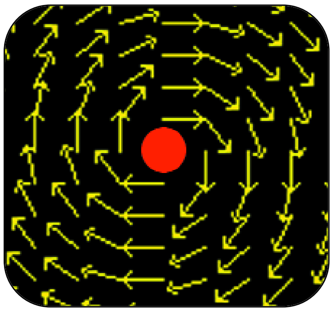
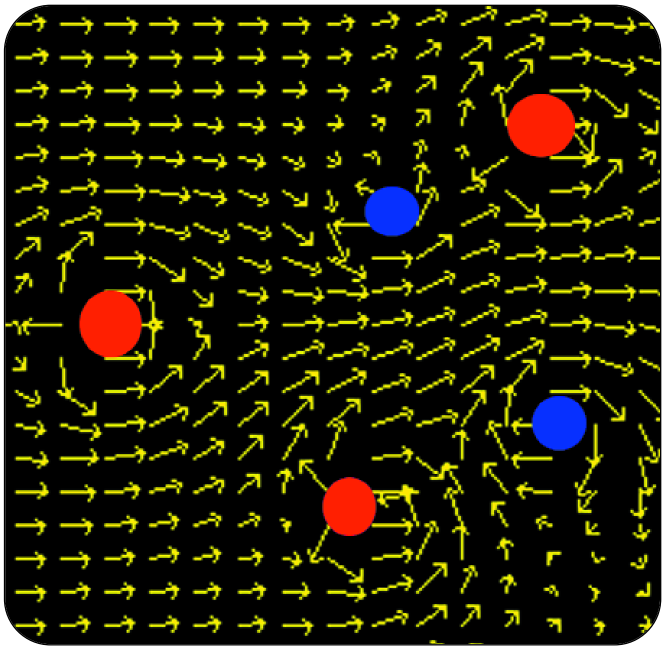






FIGURE 4 (top left). Uniform flow. FIGURE 5 (top middle). Source flow. FIGURE 6 (bottom left). Doublet flow. FIGURE 7 (bottom middle). Vortex flow. FIGURE 8 (far right). A complex flow field.

an interesting approach to rendering the sky. It makes sense for me to pursue this idea a bit further. Instead of just creating a texture, I can think of each individual brush stroke flowing along the river of the heavens. The stars and moon are obstacles in the flow, disrupting and curving the strokes around them.

In order to create an environment with this flow as the sky, I need to define where the sky is drawn. Much like a hemispherical map or a skybox, the brush strokes can be drawn on the surface of a 2D plane that is wrapped around me. This effectively changes the problem into a 2D fluid flow problem. That will make the problem quite a bit easier.

Fluid simulation is a difficult problem. The equations are complex and the calculations are computationally intensive. The governing equations of fluid mechanics are the Navier-Stokes equations. These equations describe the basic laws of mechanics in terms that describe all of the effects and phenomena that we generally associate with fluid flow. The complexity of the involved mathematics means that fluid flow problems are generally solved with numerical methods.

The two primary properties of fluids are viscosity and inertia. Viscosity is a term you normally hear mentioned in connection with motor oil. However, this property of fluid is the source of internal friction forces within the fluid. This property is most apparent in the way that a highly viscous fluid flows, such as honey off of a spoon, when compared with a fluid of relatively low viscosity, such as water. Inertial force is the property that causes the fluid to keep flowing in its direction of motion.

In general, the viscous forces in liquids are much greater than in gases. In a gas, the actual viscous force is so low that idealized gases often ignore internal frictional forces and are called inviscid, meaning not viscous. As such, the behavior of these idealized gases can be described by dynamic equations that are much simpler than the complete Navier-Stokes equations. Examining the sky in the painting, I believe that the formulas of elementary flow would adequately simulate the sky.

Bryan Marshall's Siggraph presentation last year (see For More Information) described the simulation of elementary flows by dividing the simulation area into a grid of discrete cells, as is often needed when simulating viscous fluids. This is done by simulating the "potential flow" as the summation of a set of elementary flows. Once this flow field is described, the total directional change in velocity at any point in the simulation area can be directly determined. In order to use this potential fluid flow field, I can trace particles through the simulation area, modifying the particles' velocity at each position.

Living up to Its Potential

To make a workable simulation system, I need to describe all of the elements that make up the flow field for my work area. These elements are the building blocks that I can use to describe the flow.

Uniform flow. Uniform flow describes flow in a general direction and of a single strength throughout the flow field. The mathematics of uniform flow is pretty simple. The velocity, V, at any point in the flow field is described as a flow rate, U, and angle of flow, θ (see Figure 4).

$$V_x = U\cos\theta$$
$$V_y = -U\sin\theta$$

Source flow. Source flow is flow generated at a point that radiates out from the source, decreasing with the distance from the source point. The velocity is described as a function of the flow rate, *q*, and the distance from the source point, (*x*,*y*).

$$V_x = \frac{q}{2\pi}\left(\frac{x}{x^2+y^2}\right)$$

$$V_y = \frac{q}{2\pi}\left(\frac{y}{x^2+y^2}\right)$$

The flow at the very center of the source is undefined. As you can see, it would give you a divide-by-zero error. This is easily avoided by not introducing particles into the flow at the exact center of the source (see Figure 5).

Doublet flow. Doublet flow describes a bit of a strange function. Given a circular obstacle in a uniform flow, this function describes the flow as it avoids the obstacle. The velocity, *V*, at any point in the flow field is described as a flow rate, *U*, and a length value, *a*.

$$V_x = Ua^2\left[\frac{x^2-y^2}{\left(x^2+y^2\right)^2}\right]$$

$$V_y = Ua^2\left[\frac{xy}{\left(x^2+y^2\right)^2}\right]$$

Like the source flow, the doublet is undefined at the center. However, particles should never reach the center, as the generated velocity will always push the particle away. It is necessary to ensure against initial creation of a particle at the center (see Figure 6).

Vortex flow. Vortex flow describes flow that is rotated about a point. This is the force that causes the rotating element of dynamic fluid flow. The force is applied tangential to the center of the vortex. The velocity is described as a function of the rotational rate, Γ, and the distance from the source point, (*x*,*y*) (see Figure 7).

$$V_x = \frac{\Gamma}{2\pi}\left(\frac{y}{x^2+y^2}\right)$$

$$V_y = -\frac{\Gamma}{2\pi}\left(\frac{x}{x^2+y^2}\right)$$

The vortex is also undefined at the center. Moving particles away from the exact center is sufficient to avoid the issue.

Building Blocks

Using these basic flow elements, I can create a complex flow pattern. At any point in the fluid field, I can calculate the sum of all the elementary flow elements for that position. The value is added to the particle as it moves through the simulation field. This ability to add flow elements to create a combined velocity field is called the superposition principle. Figure 8 shows the sum of a complex set of flow elements where red circles are barriers that generate doublet flows and blue circles are vortex positions.

Now this flow field is fairly static. Particles that travel through this field will follow the same path if they start at the same point. This is not terribly dynamic or turbulent. When we look at the flow of fluids such as smoke in the real world, we generally see quite a bit of dynamic behavior. The reason for this difference is those vortex flow points we dispersed about the simulation space. In an actual fluid flow, the vortices are generated dynamically based on changes in the pressure within the fluid area. The vortex fields are generated at positions where there is shear in the flow. For instance, the point at which the flow direction has changed around an obstacle will generate vortices that will move in the direction of the flow and then slowly diffuse over time.

I can automate the process of creating these vortex source points and simulate them through the flow space. However, it is possible that I can achieve the effect I desire by simply placing the vortex sources in a nonphysical manner to create the type of fluid flow I want.

Go with the Flow

That's all the time I have for now. Next time, I will use these tools to create an artistically interesting flow pattern for my brush strokes to follow. I will also describe how to use texture methods to make the elements look like actual brush strokes.

We also haven't even discussed how these same ideas can be applied to the creation of objects that animate in a very artistic and fluid manner. Until then, think about how physical simulations can be used to create other interesting art styles.

FOR MORE INFORMATION

PUBLICATIONS

Walther, Ingo, and Rainer Metzger. *Van Gogh: The Complete Paintings*. Cologne, Germany: Benedikt Taschen Verlag, September 1997.

WEB SITES

Fluid Flow formulas and Java Applet
See www.simscience.org/fluid/blue/supbas.html.

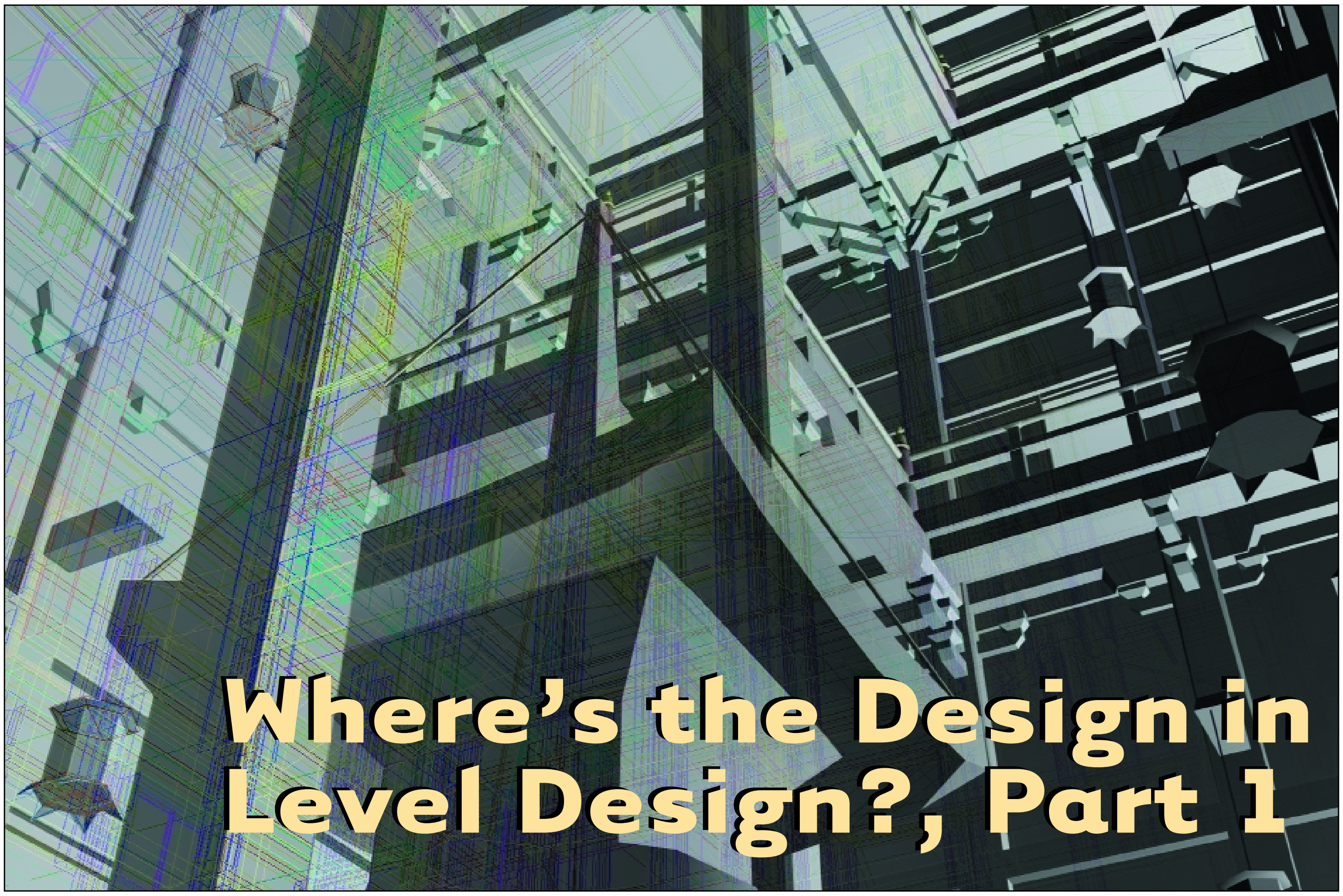
Marshall, Bryan. "Real-Time Interactive Grid Free Fluid Dynamics." SIGGRAPH 2000, Sketches and Applications.
See www.mathengine.com/Files/Marshall_Sig2000_files/index.html.

NPR Quake
See www.cs.wisc.edu/graphics/Courses/cs-838-2000/Students/herrman/nprQuake/index.html.

Quake
id Software
See www.idsoftware.com.

Video Gogh Plug-in for After Effects
See www.revisionfx.com/videogogh.htm.

Return



Where's the Design in Level Design?, Part 1

Azuchi Castle in Japan, modeled by Max Braun.

If you are a game level designer or artist who wants to create 3D interior levels that stand out and get your product noticed, creating a well-designed, believable environment is a sure way to do it. Play-balancing aside, real-time gaming "worlds" of the recent past, made up of planar-surface corridors wallpapered in repeating patterns that show off their pixel components, should be put away bearing a label that reads "For Nostalgic Purposes Only." As hardware capabilities evolve, character development and animation techniques mature, and content development software improves, the process of designing and creating richer levels for players and their 3D counterparts to play in must also evolve.

As a once-practicing interior designer now wearing one of many hats as a game artist, I have a few ideas and some architectural and interior design tips to share which I have learned throughout many years of applying environmental design. Moreover, I'll point out the possibilities which lie at the roots of architectural and interior design and which, in the hands of a creative level designer, can give the art of creating, texturing, and lighting a game level a more human countenance.

In my experience, game developers create games for other developers to appreciate just as a reputable architect would design a public building for other architects to admire and respect. Whether designing a futuristic environment or a children's virtual playroom, a poorly planned 3D environment sporting unskillfully crafted textures is not going to have the same broad audience appeal as one that is well designed and thought out. Consider a great public building that many people love to visit and always feel good in because its designer has taken into account all potential audiences who will visit and interact with it. The designer of this popular structure did not address only a particular or specialized group of people.

In creating levels with mass-market appeal, you should give thought to design that extends beyond the basics to which players of that genre are accustomed. Similarly, level designers need to reach beyond the principles and conventions established for those very specific game audiences. Like the seasoned architect or interior designer, the experienced level designer takes into account who the user

TITO PAGÁN | *Tito Pagán is a seasoned 3D artist working at WildTangent and teaching at Digipen in Seattle. He has worked for several game companies and has created art on 14 PC games. Contact him at tpagan@w-link.net or visit his web site at www.titopagan.com.*

or occupant of their new 3D world is, how they will use it, how they will interact with it, how they will move through it, and how they will approach and depart from it. This interaction, which takes place on a human scale, calls for an attention to detail down to the smallest level for most game environments. In many first-person games, that amount of design detail is not an option, given the close proximity of the game camera to surfaces in the environment. How your in-game textures are applied can be just as critical. Finally, using good design principles generally will also help you "sell" your game world more easily to your internal development team as well as your buyers.

The Price of Bad Design

When starting a new level design, a good understanding of basic design principles and guidelines can help any artist or level designer avoid making costly mistakes. This may be stating the obvious, but it does go on too often in our industry. In game development, mistakes are what we fear most when entering any new project. Good design principles, like a good game design document, can't be overlooked if you wish to avoid basic design mistakes that will cost you and your team lots of time later when you have to redo the level or its contents. The proper layout of a level adds complexity not often considered by the novice level designer who simply wants to jump in and bang out a cool-looking deathmatch level.

A well-designed level takes into consideration a whole set of requirements, such as user interaction and navigation, that are inherent to the purpose they serve. How will the spaces control and direct the player throughout the explorative and interactive experience? What sort of directional and responsive feedback mechanisms will be provided to assist the process? How will all of the elements tie together to form a cohesive environment that is well understood without compromising aesthetic appeal? The level designer must also consider the impact of particulars such as sound, space, lighting, pace, and scale.

Learning from Others

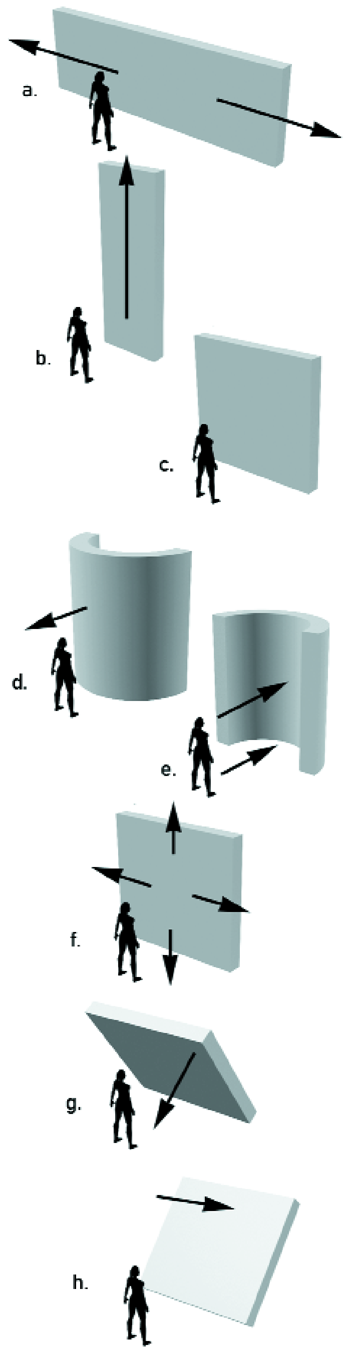
Our friends in architecture have been addressing these same design-related questions for generations. They have many of the answers to our common problems if we would only take the time to explore their proven methods of design. You can find applied methods in much of the architecture around us today if you know what to look for. These are design-oriented tools and principles that can assist us in our process as level creators.

I'm not suggesting that you take the same step-by-step approach utilized in designing real-world structures, which are bounded by gravity, physics, construction methods, and materials that are subjected to natural weather conditions. A virtual existence within a computer game is only limited by imagination and, of course, the CPU, GPU, the capabilities of the level-editing tools, the game engine, and the production budget. Nor am I suggesting that one necessarily fall back on and copy motifs directly from the past. I am advocating, however, that we learn from proven methods how to control the masses and evoke emotion with solid design principles.

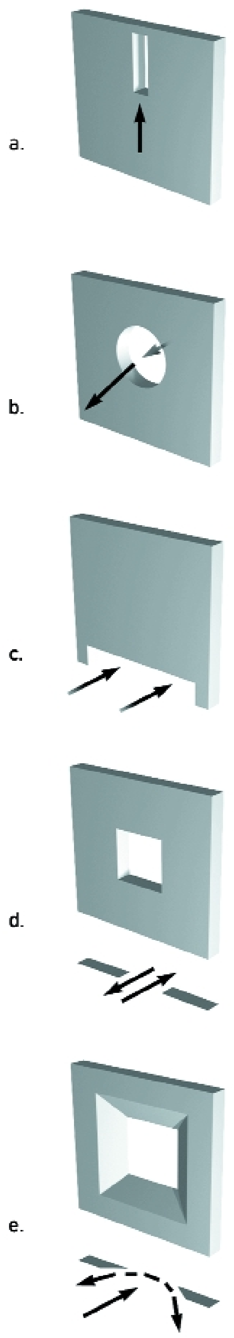
Forms That Express and Serve

There are many similar architectural structures in existence today that are patterned or modeled from the same original idea or form. Architects do this deliberately for practical reasons when constructing or designing such structures. They do it because they understand that specific forms can establish certain moods.

These basic forms are like the grammar of architecture and have been used from antiquity up to the present day as a means of addressing important goals in architectural design. Level designers can borrow much from the expressive potential of form in the theory of architecture. When designing levels, they can use this as a way to establish a common language of form, which audiences can immediately understand regardless of the individual or their culture. These are well-established principles that have immediate application in the design of our virtual worlds. They are not recipes for right



FIGURES 1A–1H. Eight different wall forms that can express weight and direction in a game level.



FIGURES 2A–2E. The window's location in the wall affects the wall's weight.

and wrong; however, they do have a design-oriented goal. I will present a few of them and explain their purpose in hopes that they provoke your interest.

Walls. When laying out a level, the first inclination a level designer tends to have is to go in and plop down a bunch of walls in an attempt to define and separate spaces before ever laying out a floor plan or a concept drawing on paper. This is often done in a 3D program using primitive shapes resembling slabs of generic walls and floors. The resulting interior spaces and exterior spaces created by close placement of separate buildings are then commonly arranged based on functionality, importance, line-of-sight, and progression through the game. At this point it would be a good time to go back and revisit the walls themselves. In game levels such as first-person 3D shooters, little thought is often given to the importance of the wall's form, scale, and angle. The wall motif is an expressive form.

A wall area, in principle, may be formed within eight different motifs (see Figures 1a–1h). The first two (1a and 1b) are concerned with the relationship between width and height, in that the wall's main form is either horizontal or vertical. The next three motifs (1c, 1d, and 1e) deal with the relation to depth, which are the flat, the convex, and the concave main forms. The final three motifs (1f, 1g, and 1h) deal with the slant of the wall. The wall may be upright, leaning toward us, or leaning away from us.

All eight motifs are actual representations of fundamental motion situations, which we may characterize by using words specifying directions. Figures 1a and 1b describe a "follow along" and "upward" motion, respectively. Figures 1c, 1d, and 1e convey a "halting," "advancing," and "retreating" motion. Figures 1f, 1g, and 1h depict a "neutral" motion, a "leaning towards" and "downward" motion, and a "tilting away" and "over" motion. Assuming one stands in approximately the same relative position in front of each wall, that wall will arouse certain motion impulses that create different impressions of the inside-outside relation in depth for that wall.

Comparing Figure 1a and 1b further, we find that the horizontal wall expresses a weight against the ground. Its horizontal nature gives a compressed and compact first impression. It stirs a force that starts the body into motion to follow along beside it in either direction to either side or end, as if seeking an entrance "around the corner" where something interesting or dangerous awaits the player. The vertical wall, on the other hand, is communicative for several reasons. One reason is that the weight expression of the vertical wall will always seem lighter because it is rising toward the sky. Think of churches and their columns and crosses, and look again at the image of the interior of the Japanese Azuchi Castle at the beginning of the article. Another reason is the motion expressed. Whereas the horizontal wall spreads movements, the vertical rising wall collects them. The final reason for this wall's communicative content is that, like a tower or obelisk, such a wall is the image of the erect standing figure that naturally attracts our attention. Throughout architectural history we find many examples of the characteristic differences in vertically and horizontally oriented walls.

Windows and openings. Another expressive form is the window. The window's location in the wall also affects the wall's expression of weight (see Figures 2a–2e). A horizontal window placed low in the wall increases the sinking effect, and a vertical window high up increases the rising effect, while a centralized window seems more ambiguous.

Examine the openings in Figures 2d and 2e. The impression that the motion is from the outside inwards can be heightened or lessened by the opening's profiles. A cut at right angles to the wall emphasizes motion from the outside. The strength of the wall is weakened and it offers no resistance. With a straight profile it is as if the wall's own substance deflects the incision. The entire wall takes on a thin character, seeming to be a stiff plane with no strength.

On the other hand, a diagonally-cut opening will resist motion from outside. The narrowing of the hole itself shows that the wall is about to close. It is given added weight and substance, because the diagonal bevel conveys an impression of greater thickness than the wall actually has. In the diagonally-cut opening the hole itself appears to lie deeper in the wall than does the right-angle-cut open-

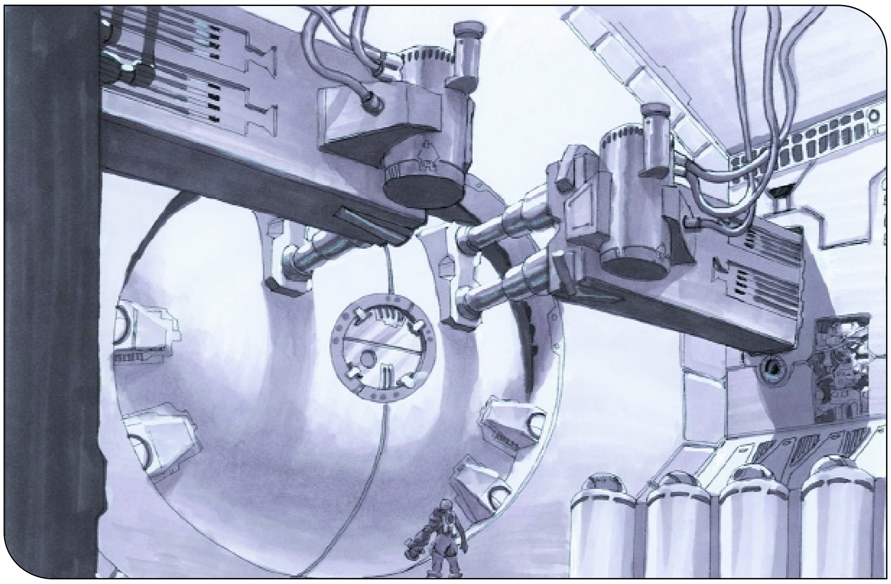


FIGURE 3. Designers can avoid costly mistakes by carefully planning all 3D assets on paper before model construction begins. Concept drawing by Richard Hescox.

ing. It is less accessible and is protected within the wall itself.

These are only a few simple examples of the many architectural forms that are at your disposal should you consider employing them in your own creations.

Achieving Realism

Realism in 3D games is often mistaken for having a photorealistic quality instead of good design. Good design principles will do more in achieving a believable environment that your players will relate to and feel comfortable in. Photorealism is a surface quality usually achieved by photographing images, objects, or natural surfaces and then cladding a 3D environment with a processed and optimized version of these images. Believable levels, on the other hand, call for the effective use of established design principles to address things such as proper lighting, transition between surfaces and textures, and how architectural elements and the surfaces and masses they are integrated with are handled. The proper placement of furniture and architectural detailing, as well as the transition between split levels should also be properly resolved. These are all common concerns in the design of interior and exterior spaces.

Do you need to achieve a photorealistic quality in your art? Before planning your road trip with your digital camera in hand, find out if the game or level design calls for this degree of realism. Is it appropriate for your genre? A photorealistic quality in a level is often well received by most audiences because of its sincere attempt to simulate the known environment around us in the real world. If done right, good photographic images that have correct lighting direction, appropriate scale, and proper surface treatment can enhance a level greatly. Skillfully crafted images can also depict construction methods used, establish a sense of good interior design, and help prolong suspension of disbelief for your players. If done wrong, the offending texture or detail stands out like a sore thumb to artist and nonartist alike.

Photorealistic images can be a double-edged sword — they're easy to achieve with a little bit of effort, a decent camera, and a good paint program, and they require no traditional art skills. The problem is that the images will detract from the scene if not applied appropriately. If this degree of realism is only reserved for having decent physics or accurate targeting in your game, then narrow your focus to achieving good design. Even a cel-shaded game would benefit from having a well-designed and balanced environment.

Preplanning Before Building

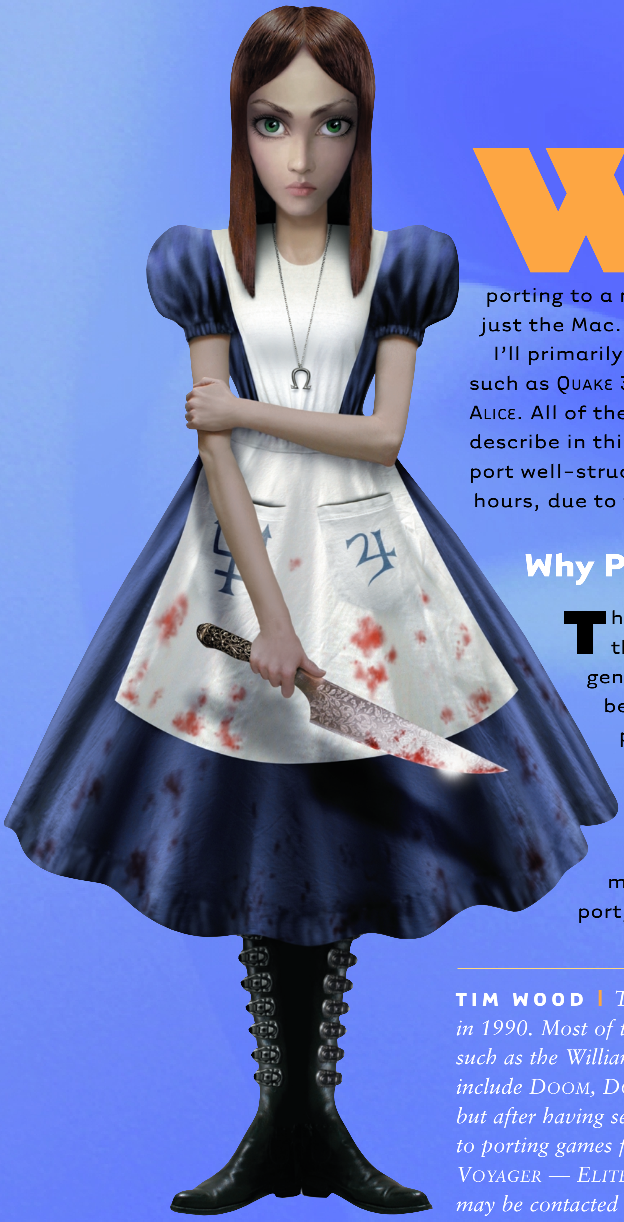
Earlier I mentioned the cost of bad mistakes. Another money-burning mistake developers often make is not preplanning art asset requirements. In addition to having good visual interest, a good design approach and some initial preplanning will pay for itself in no time. Through a lack of knowledge or preparation, we often make our choices about how to design and outfit our environments with art assets we don't need, assets we don't want, or assets that just don't work for some reason. These misfit items then get tossed aside and replaced by others, costing us more time and money to produce. This trial-and-error approach is enough to make any project manager chew through his fingernails. For professional interior designers and architects, the most cost-effective approach to creating a custom structure or interior space is to prequalify all construction needs through careful planning and evaluation, long before raising a hammer. Production artists, modelers, and level designers should prequalify the creation of all 3D assets before clicking a mouse (see Figure 3). Level design needs should also be analyzed to determine the art assets required.

If you want your levels to rise above the accepted game level look and feel, start really taking notice of the successful works of architecture and designed interior spaces around you. I challenge you to seek and borrow what you can that may address specific design problems you are encountering in your levels. Don't just reinvent blindly. Try to apply some of the approaches I've discussed here and enjoy the difference it will make as we continue to raise the bar together. Help define good design in level design. I look forward to your work.

Next Month

In next month's continuation of this topic, I will take you step by step through a design process for creating level assets that will help you avoid making costly mistakes and save you lots of time. I will also cover designing and detailing textures for your levels using more design principles and elements of design you can borrow from architecture and interior design.

Porting Games to Mac OS X



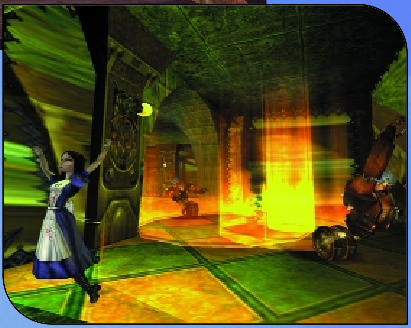
With the release of Mac OS X, the Macintosh platform gains a new path to easy-to-use and high-performance gaming. This article will address how you can easily port your current game over to the Mac and the APIs in Mac OS X that you can use to do so. Many of the issues involved with porting to a new operating system are common to porting to any new OS, not just the Mac.

I'll primarily be addressing this subject from my experiences in porting games such as QUAKE 3: ARENA, STAR TREK: VOYAGER — ELITE FORCE, ONI, and AMERICAN MCGEE'S ALICE. All of these titles obviously belong to a similar style, but the techniques I'll describe in this article are applicable to any game. We at Omni have been able to port well-structured games successfully in a matter of days, and sometimes hours, due to the productivity and ease-of-use advancements in OS X.

Why Port to the Mac?

The first reason is obviously economics. Sure, writing games is one of the most exciting and most challenging jobs around, but if you aren't generating income, you are either independently wealthy, or you won't be doing it for very long. Every developer should strive to write portable and modular code as a matter of course. The benefits of doing this are many and diverse. One of the benefits is being able to move your code easily to a new platform and attract an audience that you wouldn't have attracted otherwise. If your game is written correctly from the beginning, a port to the Mac will generate much more money than the cost of porting it. If you don't do the port, you might as well toss money in the trash.

TIM WOOD | *Tim has been programming the progenitors of Mac OS X since NeXTStep 0.9 in 1990. Most of that time was spent on mission-critical custom applications for companies such as the William Morris Agency and AT&T Wireless Services. Early "spare time" ports include DOOM, DOOM 2, QUAKE 2, QUAKE 3, and 3dfx Glide/MiniGL for Mac OS X Server, but after having seen one database/Corba server/transaction manager too many, Tim switched to porting games full-time. His current projects include OS X Cocoa versions of STAR TREK: VOYAGER — ELITE FORCE, HEAVY METAL F.A.K.K. 2, AMERICAN MCGEE'S ALICE, and ONI. Tim may be contacted at tjw@omnigroup.com.*



Revenue in the Mac market will certainly not be as high as in the PC or console markets, but neither are costs. Advertising does not cost millions of dollars in the Mac market. Due to the high level of community, the word-of-mouth advertising, and possibly piggy-backing on your PC marketing, if you have a simultaneous release date it can yield very good market penetration. The Mac market also doesn't demand that you produce a game with a $3 million budget. If you are looking at original development on the Mac, you can build games very cheaply that will be well received (and perhaps focus more on gameplay rather than having to spend time on all the latest graphics effects just to get on the shelf). Simple and well-designed titles are possible on the Mac.

The Mac market has a longer shelf life for titles than the PC market, and there is less overall competition. Thus, while it is very easy to lose money on a PC or console title, it takes much more effort to do so on a Mac title.

Also, if you license your PC publication rights, you can often hold back the Mac rights. Many PC publishers will not see the economy of scale on the Mac that they need in order for them to turn a profit. By doing so, you can either publish on the Mac yourself or find a publisher that specializes in the Mac market and knows how to make money there. This will give your development house additional income beyond the advances and royalties you get from your PC publisher.

There are other, less obviously money-grubbing reasons to port to the Mac. If you plan on licensing or reusing the engine that you are using for your current title, the work in making your engine run on the Mac can be amortized over multiple titles, and it can generate more licensing interest.

Finally, moving your code to another platform can help uncover many latent bugs in your code. In this case, the extra effort involved in supporting multiple platforms can actually reduce the amount of work at the tail end of a project by ensuring that the base you are building your game on is as stable as possible.

ABOVE (from left to right). Screenshots from some of the games ported to OS X. STAR TREK: VOYAGER — ELITE FORCE, ONI, QUAKE 3: ARENA, and AMERICAN McGEE'S ALICE .

Planning Your Game

As with any complicated task, large gains in productivity can be had if you plan your effort before embarking on it. The first step in planning a project is deciding where you want to end up after all your effort has been expended. This applies to both the features you want in your game and the platforms on which it will run. The earlier you decide on your supported platforms, the easier it will be to achieve your goals.

The decision of which features your game will include is related to the platforms you will support. For example, if you are planning on supporting wireless gaming, you probably won't be using OpenGL for at least a couple of years (beyond that, who knows?). Likewise, if you are going to write games that are going to run on the Mac, you need to pick foundation technologies that are available there. This excludes proprietary technologies like DirectThis and DirectThat (and Mac proprietary technologies such as CoreGraphics, Cocoa, Carbon, and so on if you are going to run on Windows).

Screenshot from STAR TREK: VOYAGER — ELITE FORCE.

There are many well-known software techniques for multi-platform development. I'll assume you are familiar with these for the purpose of this article, and I'll focus exclusively on Mac OS X–specific techniques. Some of the arrows in your quiver for multi-platform development should include separating code using `ifdefs` based on the platform, using custom data types to steer clear of standard type system dependencies, avoiding depending on bitfield order, steering clear of compiler- and linker-specific behaviors, and of course using a good source code control system and open APIs.

Mac OS X Technology

Mac OS X has two primary high-level toolboxes, Cocoa and Carbon. Mac OS X also has two object file formats, Mach-O and CFM. There are some choices to be considered when choosing between these. In this article, I'll be talking about the Cocoa/Mach-O approach. This choice is primarily due to the fact that fewer lines of code are necessary to accomplish similar functionality in Cocoa, and Cocoa requires Mach-O.

Cocoa is Apple's advanced object-oriented application toolkit, which is based on the technology it acquired from NeXT. Carbon is a distillation of the classic Mac OS toolbox APIs that removes a bunch of the less commonly used functions which were not easily implemented in the new world. This includes removing things such as direct access to the hardware, completely obsolete APIs, and so on. The remaining APIs have been modified to work in terms of the new underlying OS. So, a Carbon application can run on both OS 9 and OS X (as long as it doesn't make use of any new OS X services).

The foundation of Mac OS X is the Mach kernel. Mach provides the hardware abstraction and lowest-level OS services. This includes interprocess communication, protected virtual memory, threads, symmetric multi-processing, and driver services. The BSD/POSIX layer sits on top of Mach (so a BSD process is really a Mach task with a little extra goo, and a POSIX thread is really a Mach thread with some of its own goo). All of the API sets are accessible from a user program written using Carbon or Cocoa, but the vast majority of users will be able simply to use the high-level APIs or maybe occasionally use the intermediate BSD APIs. Only in special cases is it necessary to access the Mach API directly, so most of the time Mach sits in the background, providing a rock-solid OS infrastructure. As an example, Mach provides an API for pausing a thread, getting or setting its registers, and then allowing it to continue. Programs don't need to typically do this, but if you were writing a debugger, this would be very important.

Platform-Specific APIs

There are a few platform-specific APIs that most games depend on. These are APIs for performing the following tasks:

- System functions (file and network access, memory management, threading, code loading and unloading)
- Display management
- 3D graphics rendering
- Music and sound effect playback
- Reading input devices.

I'll address each of these functional groups in turn.

System functions. Mac OS X has a BSD 4.4 API layer as part of its Mach-based kernel. Apple has stated that their goal is to be POSIX-compliant. This means that a large portion of the platform-specific APIs can be addressed via both BSD and POSIX APIs.

The Windows stdio interface is very similar to the BSD functionality from which it was copied (except for Windows' strange notion of binary versus text files). The stdio API can be used for all file I/O, with the option of accessing the Mach API for memory-mapping files.

Likewise, the BSD sockets API served as the template for the Windows version. There are some minor differences here (`select()` versus `WaitForSingleEvent()` and the list of supported socket options), but nothing terribly surprising.

Unlike on many systems, the standard memory allocation package on Mac OS X performs very well. Still, for portability with other platforms you may choose simply to use `malloc` to allocate large chunks of memory for your internal memory allocator.

For threading, Mac OS X uses the POSIX threading library (pthreads). The implementation of this library isn't 100 percent complete, but the items which aren't implemented are more esoteric. If your game uses threads at all, you likely only want to create

LISTING 1. Pinning the mouse in the center of the screen in full-screen mode.

```
CGEventErr err;
err = CGAssociateMouseAndMouseCursorPosition(boolValue);

CGDisplayErr err;
err = CGDisplayHideCursor(display);
err = CGDisplayShowCursor(display);

CGPoint point = (CGPoint){x, y};
err = CGDisplayMoveCursorToPoint(display, point);
```

LISTING 2. Setting and restoring the gamma curve.

```
CGGammaValue gamma;
CGDisplayErr err;

gamma = 1.0 / someValue;
err = CGSetDisplayTransferByFormula(
            display,
            min, max, gamma,  // red
            min, max, gamma,  // green
            min, max, gamma); // blue

// Restores ALL displays
CGDisplayRestoreColorSyncSettings();
```

threads, mutexes, and conditions — this portion of the API works fine. If you do want to do anything more interesting, there is the option to use the underlying Mach thread APIs (each pthread corresponds to a Mach thread).

Mac OS X uses a dynamic linker called "dyld," which handles both launch-time linking of shared libraries and run-time linking of code modules. While it is possible to call dyld directly for your code-loading needs, it is probably easier to use the "dl" API defined in Linux, Solaris, and other Unix platforms. A wrapper for dyld that provides the dl interface can be found as part of the open source Darwin kernel that resides underneath Mac OS X (see For More Information).

The input to the dl wrapper API should be a Mach-O "bundle" file (as opposed to a dynamic library, or "dylib"). Using ProjectBuilder, the IDE that ships with Mac OS X, or whichever IDE you prefer (Codewarrior, for example), you can easily build a bundle file. Bundles are typically file wrappers, which simply means that they are directories that contain a variety of resources, one of which is code to be loaded. The path to this code is what should be passed to the dl API functions.

It is also possible to load CFM libraries into Mach-O processes using the Carbon Code Fragment Manager APIs. You might choose to do this if you want to use a toolkit that is only available in CFM format for the Mac (for example, the Bink video library).

Display management: CoreGraphics. At the heart of Apple's new Quartz rendering system is the CoreGraphics framework. CoreGraphics implements a powerful PDF-based imaging model and also supplies primitives for accessing and configuring the display hardware.

CoreGraphics can easily support multiple displays, so the first thing to do is choose the display or displays and your preferred display mode(s). Each mode is a dictionary of key/value pairs which can be queried easily. The `kCGDisplayIOFlags` key returns a mask with various interesting bits. By far the most useful is the `kDisplayModeStretchedFlag`. On a Cinema Display (or other wide-aspect-ratio monitor), there may be multiple versions of the same mode, one with a square pixel aspect ratio and one that is non-square, taking advantage of the full width of the screen. Typically, you will want to pick the unstretched mode, but if your graphics technology allows for it, you could pick the stretched mode and apply a viewport transformation that accounts for the nonsquare pixel aspect ratio (and thus you get to use the entire viewable area of the monitor).

The CoreGraphics framework also allows control over the cursor. In addition to being able to hide/show and move the cursor, CoreGraphics allows you to disassociate the mouse and cursor. This means that when the user moves the mouse, you will receive mouse events, but the cursor on the screen will not change position. This is useful for automatic demonstrations of an application, but it is also useful in full-screen OpenGL applications. If you do not pin the mouse down while in full-screen mode, even though there is a window shielding the entire display, if the user moves the mouse high enough to hit the menu, the menu will start grabbing the mouse events. The easiest way to avoid this is to pin the mouse in the center of the screen while you are in full-screen mode (see Listing 1).

You will also want to allow the user to control the gamma setting inside the game. CoreGraphics provides several functions for setting the gamma curve. Some of these functions take tables of data, and some of them take function descriptions (see Listing 2). When your game is about to exit (or if you have a "reset to defaults" option in your configuration screen), you will want to restore the gamma curve to that specified in the user's ColorSync settings.

3D graphics rendering: OpenGL. The clear choice for 3D on Mac OS X is OpenGL. As with every other platform, the Mac OS X version of OpenGL adds its own API for creating a GL context and binding it to a drawing surface. Mac OS X actually provides three such APIs. Two of these correspond directly to the two high-level UI toolkits. For Cocoa, there is NSOpenGL, while Carbon has AGL. Both of these are very thin layers on top of CoreGL (CGL).

The first stage to creating an OpenGL context is to decide whether you are going to be in full-screen mode or not. If you are, then you need to capture the display and change its mode setting. Capturing the display prevents any other applications from accessing the display and, very importantly, prevents other running applications from being notified of a screen geometry change. If you don't lock the display, other applications will find

LISTING 3. Creating the OpenGL context.

```
#define ADD_ATTR(attr) \
do { \
  attrCount++; \
  attrs = realloc(attrs,sizeof(*attrs)*attributeCount);\
  attrs[attrCount-1] = attr; \
} while (0)

NSOpenGLPixelFormat *pixelFormat;
NSOpenGLPixelFormatAttribute *attrs;
unsigned int attrCount = 0;

attrs = malloc(sizeof(*attrs));

if (fullscreen)
  ADD_ATTR(NSOpenGLPFAFullScreen);
ADD_ATTR(NSOpenGLPFAColorSize);
ADD_ATTR(colorBits);
ADD_ATTR(NSOpenGLPFADepthSize);
ADD_ATTR(depthBits);
ADD_ATTR(NSOpenGLPFADoubleBuffer);
ADD_ATTR(NSOpenGLPFAAccelerated);
ADD_ATTR(NSOpenGLPFAScreenMask);
ADD_ATTR(CGDisplayIDToOpenGLDisplayMask(display));
ADD_ATTR(0);

pixelFormat = [[NSOpenGLPixelFormat alloc]
               initWithAttributes: attrs];

free(attrs);

context = [[NSOpenGLContext alloc]
           initWithFormat: pixelFormat
           shareContext: nil];
[pixelFormat release];

// Set context to draw into
[context makeCurrentContext];
```

out about the geometry change and move their windows around, annoying the player.

Next, you need to create a pixel format object that describes the list of attributes that the OpenGL context must have. This includes the number of color and depth bits, whether the context should support full-screen usage, and so on. In Mac OS X, the OpenGL context must always have the same color depth as the frame buffer. Once we have the pixel format, we simply use it to create the OpenGL context and then discard it (see Listing 3).

We can set and query a wide variety of parameters on the context. One useful example is setting whether buffer flushes are synchronized to the vertical refresh.

LISTING 4. Setting up CoreAudio.

```
propertySize = sizeof(outputDeviceID);
status = AudioHardwareGetProperty(
        kAudioHardwarePropertyDefaultOutputDevice,
        &propertySize, &outputDeviceID);

propertySize = sizeof(bufferByteCount);
bufferByteCount = SAMPLES_PER_BUFFER * sizeof(float);
status = AudioDeviceSetProperty(outputDeviceID,
        NULL, 0, NO,
        kAudioDevicePropertyBufferSize,
        propertySize, &bufferByteCount);

status = AudioDeviceAddIOProc(outputDeviceID,
                              audioDeviceIOProc,
                              userInfoPointer);

status = AudioDeviceStart(outputDeviceID,
                          audioDeviceIOProc);
```

Before we can draw anything into the context, we naturally need to make it the current context. We also need to be able to clear the current context, display the context, and occasionally we need to find the current context. All of these operations are simple one-liners.

Music and sound effect playback: CoreAudio and Sound Manager. Mac OS X includes several APIs for making noise. CoreAudio is the lowest-layer audio API. CoreAudio uses a callback to provide sound samples. This callback is invoked in a different thread, so it must be at least minimally thread-safe. All samples are in floating-point format, making it easier to perform mixing. The callback receives several timestamps, two of which are valid depending upon whether the callback is being invoked to play or record audio samples. One of the timestamps is the current time; in the case of playback, the other timestamp is the time at which the samples currently being requested will actually be heard. This allows for fine-grain synchronization between what you see on the screen and what you hear.

Setting up CoreAudio is very simple. We simply get the device on which we want to play audio, configure the buffer size, provide a callback, and tell the device to start playing (see Listing 4).

The audio API a level above CoreAudio is the Carbon Sound Manager. Since Sound Manager is built on top of CoreAudio, it will have slightly higher overhead than using CoreAudio directly, although if you are using `short` samples internally instead of floating-point samples, you may be better off using Sound Manager. Unlike CoreAudio, you do not need to provide a callback function, but can instead just send play commands whenever appropriate. Sound Manager does provide a command that will call a callback function, so you can issue another play command and request another callback when that buffer is finished.

LISTING 5. Example mouse input-handling code.

```
NSEventType eventType;
CGMouseDelta deltaX, deltaY;

eventType = [event type];
switch (eventType) {
case NSLeftMouseDown:
case NSLeftMouseUp:
case NSRightMouseDown:
case NSRightMouseUp:
case 25: // New undocumented ´other´ mouse down
case 26: // New undocumented ´other´ mouse up
        break;

case NSSystemDefined:
        if ([event subtype] == 7) {
          unsigned int buttons;

          buttons = [event data2];
          // buttons is a bitfield of 32 mouse button states
        }
        break;

case NSMouseMoved:
case NSLeftMouseDragged:
case NSRightMouseDragged:
case 27: // New undocumented ´other´ mouse dragged
        CGGetLastMouseDelta(&deltaX, &deltaY);
        break;

}
```

In addition to the relatively simple CoreAudio and Sound Manager APIs, Mac OS X provides the QuickTime API. QuickTime is extremely powerful and, thus, rather more complicated than either of the two lower-level APIs. QuickTime provides facilities for playing audio, video, Flash, PDF, and other media types. Of particular interest for game developers are the audio decompression capabilities of QuickTime. The code to do this is too long to present here, but is available with the rest of the example programs developed for this article (see For More Information).

Reading input devices. Keyboard and mouse input can be implemented through a combination of the normal Cocoa event mechanism and calls to CoreGraphics. Support for other input device types (such as joysticks) can reportedly be accomplished via HIDManager, but as this API is not documented yet, I won't address it here (although check the Omni Group web site for updates later).

Keyboard up and down events are just normal Cocoa event objects. These objects carry a string of characters in the event, encoded as Unicode characters. Function keys and other special keys such as Help and Home are defined in the vendor-specific Unicode range. Modifier keys such as Shift, Control, and Command do not transmit keyboard up or down events, since there are no Unicode characters for these keys. There is a "flags changed" event that is sent when the state of these keys changes.

Mouse button events are transmitted in two different ways under Mac OS X. Left, right, and "other" buttons have individual up and down events, but if you want to handle a larger number of buttons, it is easiest just to ignore these events. Instead, you can use the "system defined" event which is sent each time a mouse button changes state — part of the data payload for this event type is a 32-bit mask of the current button state.

When the mouse moves, events are sent to your application. These events contain absolute mouse position information, clipped to the bounds of whatever screen the mouse resides on. The data in these events can be ignored, with the event just serving as a notification that you should call CoreGraphics' `CGGetLastMouseDelta()` function. See Listing 5 for an example event-handling loop for mouse events.

There are a few common problems you might run into with input on Mac OS X:

- The keyboard repeat and mouse-scaling settings are not automatically restored when your application terminates.
- The "other" mouse button events have been added since the public beta but are not yet documented.
- It is best to avoid assuming that your engine will be able to poll the state of a device button, since some platforms only have event interfaces.

Also, if you are creating a windowed application (not full-screen), you will need to create your own window object in which to place your game view. If you want to receive mouse movement events in this case, you need to request them explicitly (this is not necessary in full-screen mode).

PowerPC Specifics

The code snippets in the example programs are sufficient to build a game that runs on the Mac, but in order to make a game that runs as well as possible, there are a few PowerPC-specific issues that you may need to address, depending upon the architecture of your game.

Pitfalls. The Macintosh does not have the memory bandwidth that Intel boxes have. This is less true on the newer machines, but if you are targeting older iMacs, you will need to be aware of this. There are things that you can do to help avoid this problem. First, stay away from back-to-back load and store operations. Instead, load several values, operate on them, and then store them. The PowerPC chip has a huge register file compared to Pentiums. You can avoid a lot of memory operations simply by putting more values in registers. The PowerPC also provides a set of cache control instructions that allow you to preload cache lines, flush them, or zero out entire cache lines (much faster than doing it yourself, since you avoid the read to load the cache line).

Converting between floating-point and integer formats is expensive on the PowerPC. There are two reasons for this. First, since the PowerPC is RISC, the floating-point and integer units are only connected via memory through the load and store unit. Additionally, the PowerPC (prior to the G4 Altivec instruction set) does not have architecture-level support for converting from integers to floats. Casting between float and integer is never free on any architecture, but it is definitely more expensive on the PowerPC. You can often get large performance increases by eliminating needless casting back and forth between `int` and `float`.

If your game engine is in C++, you will not be able to mix Objective-C code snippets as listed into the same files as your core C++ code. This isn't a huge problem, since all the platform-specific code should be isolated in its own files anyway. Currently, the simplest way to call between C++ and Objective-C is to use a vanilla C interface. If you design your platform support library interface in pure C, you won't even notice this problem. Apple is devoting engineering resources to this issue, however, and will be talking about their progress at the 2001 Apple Worldwide Developers Conference in May.

Screenshot from American McGee's Alice.

Optimizations. The PowerPC has a few instructions that deserve to be pointed out for possible optimizations.

If your engine uses the square root math library function, you might be able to use the `frsqrte` instruction. This instruction computes an estimate of the inverse of the square root. Depending upon your precision needs, you can use multiple Newton-Raphson refinement steps to extend the precision of the result. The `frsqrte` instruction can in practice be up to 16 times faster than `1.0 / sqrt(x)`. In addition to using this instruction for the reciprocal square root, you can also use it to compute a normal square root by simply multiplying its result by the original value, since `x / sqrt(x) = sqrt(x)`.

The PowerPC provides an `fsel` instruction for performing simple if/else assignments. This can eliminate branches in inner loops, which not only reduces the total number of instructions issued, but also frees up branch prediction slots and eliminates the possibility of an incorrectly predicted branch.

Another interesting group of instructions is the `lwbrx` family (load word byte reversed indexed). This family of instructions allows you to load or store two- and four-byte values and also perform endian swapping. This is much faster than loading the value and then performing bitwise operations in order to swap the value around manually.

Performance-monitoring tools. Mac OS X ships with a full set of developer tools. Included in this are several performance monitoring tools. The Sampler application (and the "sampler" command line tool) will periodically stop all the threads in your application, record their stacks, and then let them continue. This provides you with a tree of wall clock time spent, easily allowing you to find the portions of your program that are using the most time. You can also invert the tree, putting the leaves at the root, allowing you to find small leaf routines that are taking a large amount of time.

Omni also provides the OmniTimer framework (see For More Information). This allows you to insert instrumentation calls into your application at key points (typically determined by running Sampler) in order to get very high precision timings. OmniTimer uses the PowerPC TBR (time base register) in order to minimize the overhead of collecting the timestamps.

Advanced topics. The PowerPC G4 ships with what is considered by many to be the best SIMD instruction set in a consumer CPU. With the right type of task, Altivec can provide huge performance gains. It is also possible to store floating-point (X, Y, Z, W) vector data in single Altivec registers and operate on those registers using macros or inline functions. Care must be taken to keep the values in the vector registers to yield performance gains. The jury is still out on the feasibility of this approach, but it is worth considering.

Mac OS X provides full symmetric multi-processing. If you have the right sort of tasks (that is, very few synchronization points and low data flow), you can break them up into separate threads and Mac OS X will automatically schedule them to different CPUs (if available).

Mac Attack

In the past couple of years, Apple has increased its focus on gaming, and it shows. The Macintosh is now a great gaming platform and only looks to improve in the coming years. By porting to the Mac you can experience increased portability and robustness for all platforms.

Even better, by adding the Mac enthusiasts to your customer base, you can increase your revenue stream while continuing to produce excellent games.

FOR MORE INFORMATION

This page on our web site contains pointers to Mac OS X–specific game development resources. Included in this is a reference to my GDC 2001 talk, including several sample applications that demonstrate all of the systems (and more) addressed above.

www.omnigroup.com/community/developer/game-development/

Apple

http://developer.apple.com/games/index.html

Return

Leveling the Playing Field

Implementing Lag Compensation to Improve the Online Multiplayer Experience

YAHN BERNIER | *Yahn received his undergraduate degree in chemistry from Harvard University and his law degree from the University of Florida School of Law. Yahn spent five years practicing patent law in Atlanta and, in his spare time, he authored the popular QUAKE level editor BSP. Yahn joined Valve in 1998 working on technology for Valve's first title, HALF-LIFE. Currently, Yahn is a senior engineer and development lead for Valve's upcoming title TEAM FORTRESS 2. Contact him at yahn@valvesoftware.com.*

Designing first-person action games for Internet multiplay is a challenging process. Having robust online gameplay in your action title, however, is quickly becoming essential to the success and longevity of the title. In addition, the PC space is notorious for requiring developers to support a wide variety of customer setups. Often, customers are running on less than state-of-the-art hardware or network connections.

While broadband has been held out as a panacea for all of the current woes of online gaming, broadband is not a simple solution allowing developers to ignore the implications of latency and other network factors in game designs. Moreover, it will be some time before broadband truly becomes widely adopted in the United States and much longer before it can be assumed to exist for your clients in the rest of the world. In addition, there are many poorly executed broadband solutions, in which users may occasionally have high bandwidth but more often than not will also have significant latency and packet loss in their connections.

The traditional client/server architecture, when applied to Internet multiplayer action games, presents some problems of its own when faced with network latencies. For instance, one of the most significant challenges is synchronizing instant-hit weapon fire under a system where the players all have different amounts of latency, and the latency itself varies from frame to frame. Synchronization must occur while simultaneously balancing the need for instant feedback with the need for maintaining a fair landscape of play between narrowband and broadband users.

In some first-person shooter titles such as QUAKEWORLD (a later version of QUAKE),

you have to lead your target by some distance related to your latency to the server. Aiming directly at another player and pressing the fire button is almost ensured to miss that player. The trouble with this approach is that leading with an instant-hit weapon is just not realistic, and the player control has nonpredictable responsiveness due to the varying latency.

Underpinning HALF-LIFE's networking architecture was our game design goal to allow each player to have completely responsive interaction with the world and with his or her weapons. This article details the technology we developed and the trade-offs that were made to achieve this goal. Along the way, I'll also give some background on how client/server architectures work in typical online action games.

Basic Architecture of a Client/Server Game

Most action games played on the Internet today use a modified client/server architecture. Games such as HALF-LIFE, including its mods such as COUNTER-STRIKE and TEAM FORTRESS CLASSIC, operate on such a system, as do games based on the QUAKE 3 engine and the UNREAL TOURNAMENT engine. In these games, there is a single, authoritative server which is responsible for running the main game logic. To this are connected one or more

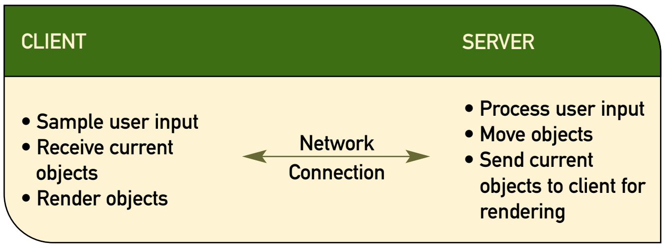


FIGURE 1. General client/server architecture.

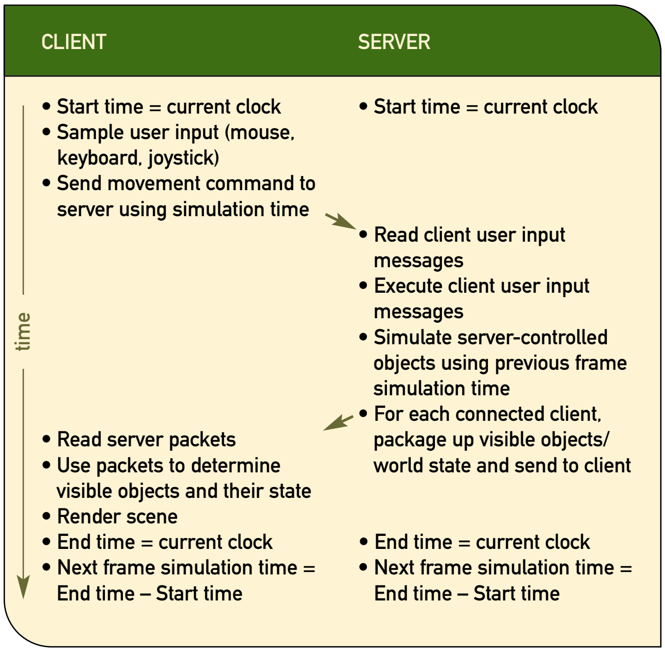

CLIENT
SERVER
time
• Start time = current clock
• Sample user input (mouse, keyboard, joystick)
• Send movement command to server using simulation time
• Start time = current clock
• Read client user input messages
• Execute client user input messages
• Simulate server-controlled objects using previous frame simulation time
• For each connected client, package up visible objects/ world state and send to client
• Read server packets
• Use packets to determine visible objects and their state
• Render scene
• End time = current clock
• Next frame simulation time = End time – Start time
• End time = current clock
• Next frame simulation time = End time – Start time

FIGURE 2. The system frame loop allows for synchronized functionality between the client and the server.

"dumb" clients. These clients initially were nothing more than a way for the user input to be sampled and forwarded to the server for execution. The server would execute the input commands, move other objects, and then send back to the client a list of objects to render (see Figure 1). Of course, the actual system has more components to it, but this simplified breakdown is useful for thinking about prediction and lag compensation.

For this discussion, all of the messaging and coordination necessary to start up the connection between client and server is omitted. The client and server are functionally synchronized on a per-frame basis after initialization is completed. The system's frame loop looks like Figure 2.

Each time the client and server make a pass through this loop, the frame time is used for determining how much simulation will be required on the next frame. If your frame rate is constant, then the frame time will be a correct measure; otherwise, the frame times will be incorrect. But there isn't really a solution to this, unless you have some way to compute deterministically how long it is going to take to run the next frame loop iteration before actually running it.

In this model, non-player objects run purely on the server, while player objects drive their movement based on incoming packets. Of course, this is not the only possible way to accomplish this task, but it does make sense.

In games using the HALF-LIFE engine, the user input message that is sent to the server is quite simple and contains just a few essential fields: frame time, view direction, movement velocity, currently pressed buttons, and a few weapons flags (see the `usercmd_t` data structure in the HALF-LIFE SDK, file common/usercmd.h).

Using this data structure and the client/server process above creates a quite simple simulation, but it doesn't react well under real-world situations with significant latency in client connections. The main problem is that the client truly is "dumb," and all it does is the simple task of sampling movement input and waiting for the server to tell it the results. If the client has 500 milliseconds of latency in its connection to the server, then it will take 500 milliseconds for any client actions to be acknowledged by the server and for the results to be perceptible on the client. While this round-trip delay may be short and thus acceptable on a LAN, it is definitely not acceptable on the Internet.

Client-Side Prediction

One method for ameliorating this problem is to perform the client's movement locally and just assume, temporarily, that the server will accept and acknowledge the client commands directly. This method can be called "client-side prediction."

Client-side prediction requires us to let go of the "dumb" or minimal client principle. That's not to say that the client is fully in control of its simulation, as in a peer-to-peer game with no central server. There is still an authoritative server running the simulation, as I noted. Having an authoritative server means that even if the client simulates different results from the server, the server's results will eventually correct the client's incorrect simulation. Because of the latency in the connection, the correction might not occur until the time for a full round trip has passed. The downside is that this can cause a very perceptible shift in the player's position when the prediction error is resolved.

To implement client-side prediction, the following general procedure is used. As before, client inputs are sampled and a user command is generated. Also as before, this user command is sent off to the server. However, each user command (and the exact time it was generated) is stored on the client. The prediction algorithm then uses these stored commands.

Prediction starts by using the last acknowledged movement from the server. This acknowledgment indicates which user command was last acted upon by the server and also tells us the exact position (and other state data) of the player after that movement command was simulated on the server. The last acknowledged command will be somewhere in the past if there is any lag in the connection. For instance, if the client is running at 50 frames per second (20 milliseconds per frame) and the network connection has 100 milliseconds of latency (round-trip), then the client will have five stored commands since the last one acknowledged by the server. These five commands are simulated on the client as a part of client-side prediction. Assuming full prediction, the client starts with the latest data from the server and then runs the five com-

mands through logic similar to what the server uses for simulation of client movement. Running these commands should produce an accurate final state on the client which can be used to determine from what position to render the scene for the current frame.

The HALF-LIFE engine shares player movement code between the client and server, in order to minimize discrepancies in the prediction logic. These are the routines in the pm_shared/ (which stands for "player movement shared") folder of the HALF-LIFE SDK (see For More Information). The input for these shared routines is encapsulated in the user command and a "from" player state. The output is the new player state after executing the user command. The general algorithm on the client is shown in the following pseudocode.

```
FromState: state after last user command acknowledged by the server
Command: first command after last user command acknowledged by server
while (true)
{
    run Command on FromState to generate ToState;
    if (Command was the most up to date command)
      break;

    FromState = ToState;
    Command = next command;
};
```

The final to state is the prediction result and is used for rendering the scene that frame. The portion where the command is run is simply the portion where all of the player state data is copied into the shared data structure, the command is processed (by executing the common code in the pm_shared routines in HALF-LIFE's case), and the resulting data is copied to the to state.

There are a few important caveats to this system. You'll notice that, depending upon the client's latency and how fast the client is generating commands (the client's frame rate), the client will most often end up running the same commands over and over again until they are acknowledged by the server and dropped from the list of commands (done via a sliding window in HALF-LIFE's case). The first consideration is how to handle any sound effects and visual effects that are created in the shared code. Because commands can be run over and over again, it's important not to create events such as footstep sounds multiple times every time the old commands are rerun. In addition, it's important for the server not to send the client effects that are already being predicted on the client. However, the client still must rerun the old commands, or else there will be no way for the server to correct any erroneous prediction by the client. The solution to this problem is easy: the client just marks those commands which have not been predicted yet on the client and only plays effects if the command is being run for the first time on the client.

The other caveat is with respect to state data that exists solely on the client and is not part of the authoritative update data from the server. If you don't have any of this type of data, then you can simply use the last acknowledged state from the server as a starting point and run the prediction commands in place on that data to arrive at a final state. In this case, you don't need to keep all of the intermediate results along the route for predicting from the last acknowledged state to the current time. However, if you are doing any logic completely on the client side (this could include functionality such as determining where the eye position is when you are in the process of crouching) which affects fields that are not replicated from the server to the client by the networking layer, then you will need to store the intermediate results of prediction. This can be done with a sliding window, where the from state is at the start, and each time you run a command through prediction, you fill in the next state in the window. When the server finally acknowledges receiving one or more commands that had been predicted, it is a simple matter of looking up which state the server is acknowledging and then copying over the data that is completely client-side to the new from state.

So far, the preceding procedure describes how to accomplish client-side prediction of movements in a manner similar to that used in QUAKEWORLD (see For More Information for the link to the QUAKEWORLD source code).

Weapons Fire Prediction

Layering prediction of weapons fire onto the above system is straightforward. Additional state information is required for the local player on the client, of course, including which weapons are being held, which one is active, and how much ammo each of these weapons has remaining. With this information, the firing logic can be layered on top of the movement logic, because once again, the state of the firing buttons is included in the user command data structure that is shared between the client and the server. Of course, this can get complicated if the actual weapon logic is different between client and server. In HALF-LIFE, we chose to avoid this complication by moving the implementation of a weapon's firing logic into shared code, just like the player movement code. All of the variables that contribute to determining weapon state (for example, ammunition, when the next firing of the weapon can occur, which weapon animation is playing, and so forth), are then part of the authoritative server state and are replicated to the client so that they can be used on the client for prediction of weapon state there.

Predicting weapons fire on the client will likely also lead to the decision to predict weapon switching, deployment, and holstering

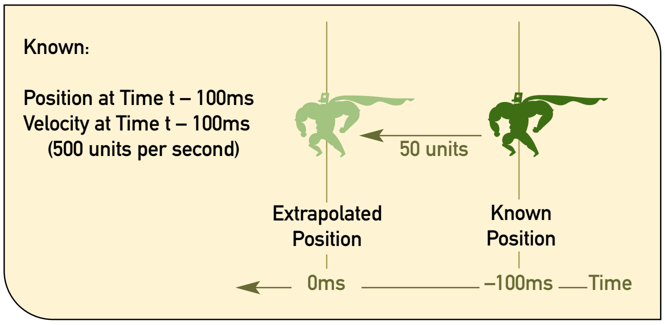


FIGURE 3. Extrapolation allows the client to calculate where the other player will be, so that the local player can play in real time.

so that players will feel that the game is 100 percent responsive to their movement and weapon activities. This goes a long way toward reducing the feeling of latency that many players have come to endure with Internet-based action games.

Replicating the necessary fields to the client and handling all of the intermediate state is a fair amount of work. At this point you may be asking, why not ditch the server stuff and just run the movement and weapons purely on the client side? Then, the client would just send results to the server such as, "I'm now at position X, and, by the way, I just shot Player 2 in the head." This works fine if you can trust the client, and in fact is how a lot of military simulation systems work, since they're a closed system and can trust all of the clients. Generally this is how peer-to-peer games work as well. Unfortunately for HALF-LIFE and most Internet-based action games, this mechanism is unworkable because of realistic concerns about cheating. If we encapsulate absolute state data in this fashion, we'd raise the motivation to hack the client even higher than it already is. (For a discussion of cheating and what developers can do to deter it, see Matt Pritchard's *Game Developer* article in For More Information.) For our games, this risk is too high, and so we have to fall back on requiring an authoritative server.

A system where movements and weapon effects are predicted client-side is very workable. For instance, this is the system that the QUAKE 3 engine supports. A problem with this system, though, is that players still have to combat their latency to determine how far to lead their targets when firing. In other words, although you hear the weapon firing immediately and your position is up-to-date, the results of your shots are still subject to your latency. For example, if you're aiming at a player running perpendicular to your view, and you have 100 milliseconds of latency and the player is running at 500 units per second, then you'll need to aim 50 units ahead of the target to hit it with an instant-hit weapon. The greater the latency, the greater the lead targeting required. Getting a "feel" for your latency is difficult for the player and detracts from the gaming experience. QUAKE 3 attempts to mitigate this by playing a "hit" tone when the client receives confirmation of a hit. This gives players some hint as to how far they need to lead their weapons fire. Obviously, with sufficient latency and an opponent who is actively dodging, it is quite difficult to get enough feedback to focus in on an opponent consistently. If your latency is fluctuating, it can be even more challenging.

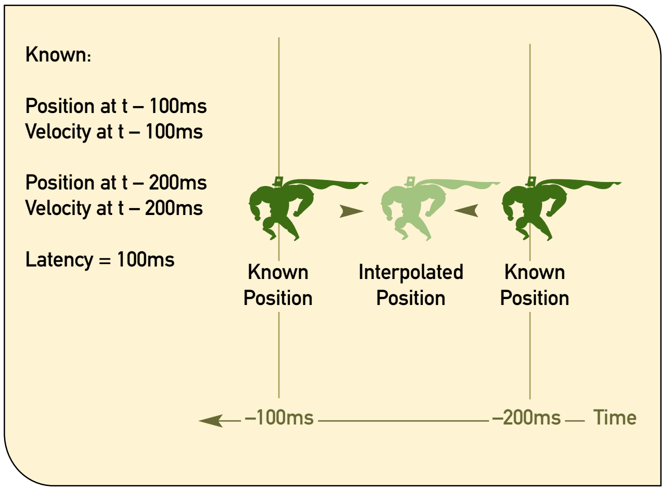
Determining Opponent Position

Another important aspect influencing how a user perceives the responsiveness of the world is the client's mechanism for determining the position of the other players. The two most basic mechanisms for determining where to display the other players are extrapolation and interpolation.

Extrapolation. In extrapolation, the other player is simulated forward in time from the last known position, direction, and velocity in more or less a ballistic manner. That is, if you're lagged 100 milliseconds, and the last update you received indicated that the other player was running 500 units per second perpendicular to your view, then the client can assume that the other player has moved 50 units straight ahead from their previous position. The client can then draw the player at that extrapolated position and the local player can more or less aim right at the other player (see Figure 3).

The biggest drawback of using this method is that player movements are not usually very ballistic. Instead, they're usually very nondeterministic and very jerky. ("Jerk" is a measure of how fast acceleration forces are changing.) Layer on top of this the unrealistic player physics models that most FPS games use, where players can turn instantaneously and apply unrealistic forces to create huge accelerations at arbitrary angles, and you'll see that the extrapolation is quite often incorrect. The developer can mitigate this error by limiting the extrapolation time (QUAKEWORLD, for instance, limited extrapolation to 100 milliseconds). This limitation helps because once the true player position is finally received, there will be a limited amount of corrective warping. Unfortunately, most players still have more than 150 milliseconds of latency, so the player must still lead other players in order to hit them. If those players are "warping" to new positions because of extrapolation errors, the gameplay suffers nonetheless.

Interpolation. The other method for determining where to display objects and players is interpolation. Interpolation uses the two most recent acknowledged player positions and interpolates between them based on the latency. For instance, if the server is sending 10 updates per second of the world state, we could impose 100 milliseconds of interpolation delay into our rendering. As we render frames, we interpolate the position of the



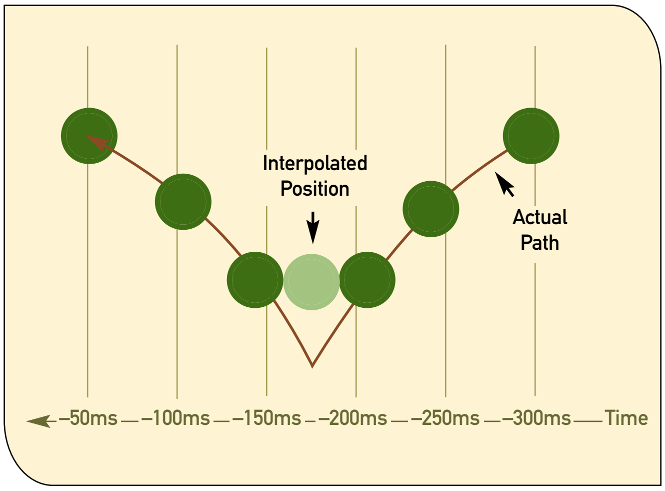


FIGURE 4 (left). Interpolation uses previously acknowledged positions and buffers one additional round-trip latency.
FIGURE 5 (right). Interpolation suffers from visual quality issues.

object between the last updated position and the position directly before that over 100 milliseconds. As the object arrives at the last updated position, we should receive the next update from the server and we can then start moving toward this new position (see Figure 4).

If one of the server update packets fails to arrive, we have two choices: we can extrapolate the player position as described previously, or we can simply have the player rest at the position in the last update until a new update arrives (which causes the player's movement to stutter).

In essence, in one basic form, interpolation buffers one additional round-trip latency on the client. The other players, therefore, are drawn where they were at a point in the past that is your latency plus the amount of time over which you are interpolating. To deal with the occasional dropped packet, we could set the interpolation time to 200 milliseconds instead of 100 milliseconds, for example. This would allow us to miss one update and still have the player interpolating toward a valid position without a hitch. Of course, increasing the interpolating time is a trade-off, because it trades additional latency (making the interpolated player even harder to hit) for visual smoothness.

Unfortunately, interpolation also suffers from visual quality issues that are difficult to resolve. Imagine that the object being interpolated is a bouncing ball (which accurately describes some of our players, actually). At the extremes, the ball is either high in the air or hitting the ground. However, on average, the ball is somewhere in between. The bounciness of the ball appears flattened out due to the ball being interpolated between just before and just after hitting the ground or reaching its high point. This is a classic sampling problem and can be alleviated by sampling the world state more frequently. However, we are still quite likely never actually to have an interpolation target state be exactly at the ground or high point (see Figure 5).

One interesting consideration we have to layer on top of the interpolation scheme is some way to determine that an object has been teleported. Otherwise we might "smoothly" move the object over great distances. We can either set a flag in the update that says "don't interpolate" or "clear out the position history," or we can determine if the distance traveled between updates is too large, and thereby presume it to be a teleport.

Using Lag Compensation

Understanding interpolation is important in designing for lag compensation, because interpolation is another type of latency in the player's experience. Since the player is looking at other objects that have been interpolated, the amount of interpolation must be taken into consideration in computing, on the server, whether the player's aim was true.

Lag compensation normalizes server-side the state of the world for each player as that player's commands are executed. You can think of lag compensation as taking a step back in time, on the server, and looking at the state of the world at the exact instant that the player performed some action. The algorithm works as follows:

1. Compute a fairly accurate latency for the player.
2. Search the server history for the update that was received by the player just before they issued the movement command.
3. From that update and the following one, move the other players backward in time to exactly where they were when the player's command was created. This moving backward must account for both connection latency and the interpolation amount that the client was using that frame (which is encoded in the player command).
4. Execute the command using these "old" positions.
5. Move all of the players back to their current positions.

Note that when we move the player backward in time, it might also require forcing additional state information backward (for instance, whether the player was alive or dead, or whether the player was ducking). The end result of lag compensation is that each player is able to aim directly at other players without having to worry about leading their target to score a hit. Of course, this behavior creates its own issues and is a game design trade-off.

Game Design Implications

The introduction of lag compensation allows each player to run and interact with the world and most other players in real time, with no apparent latency. In this respect, it is important to understand that some inconsistencies can occur. Of course, the old system with the authoritative server and "dumb" or simple clients had its own inconsistencies. In the end, making this trade-off is a game design decision. For HALF-LIFE, we believe deciding in favor of lag compensation was justified.

The first problem of the old system was that you had to lead your target by some amount that was related to your latency. Aiming directly at another player and pressing the fire button was almost sure to miss that player. The inconsistency is that aiming is just not realistic and that the player controls have non-predictable responsiveness.

With lag compensation, the inconsistencies are different. For most players, all they have to do is acquire some aiming skill and they can become proficient. Lag compensation allows the player to aim directly at their target and press the fire button (for instant-hit weapons). The inconsistencies that can sometimes occur are from the point of view of the player being fired upon.

For instance, if a highly lagged player shoots at a less lagged player and scores a hit, it can appear to the less lagged player that they've somehow been "shot around a corner." The lower-lagged player may have just darted around a corner, but the higher-lagged player is seeing everything in the past, so they may have a direct line of sight to the other player. If the higher-lagged player is sufficiently lagged, say 500 milliseconds or so, this scenario is quite possible. When the higher-lagged player's command arrives at the server, the hiding player is transported backward in time and calculated as being hit. This is an extreme case, but in this case the lower-lagged player thinks that they were somehow shot from around the corner. But from the higher-lagged player's perspective, they simply lined up the crosshairs and fired a direct hit. From a game design point of view, this was an easy decision: let each individual player have completely responsive interaction with the world and with their weapons.

In practice, this inconsistency is much less pronounced in normal combat situations. For first-person shooters, there are two more typical cases. First, consider two players running straight at each other, pressing the fire button. In this case, it's quite likely that lag compensation will just move the other player backward along the same line as their movement. The person being shot will be looking straight at the attacker and no "the bullet bent around the corner" feeling will be present.

The second example is one player aiming at another while the other dashes perpendicularly across the first player's field of vision. In this case, the inconsistency is minimized for a wholly different reason. The player who is dashing across the view of the shooter probably has (in first-person shooters, at least) a field of view of 90 degrees or less. In essence, the dashing player can't see where the aiming player is aiming. Therefore, getting shot isn't going to be surprising or feel wrong (you get what you deserve for running around in the open like a maniac). Of course, if you're playing a tank game, or a game where the player can run in one direction and look in another, then this scenario is less clear-cut. You might actually see the other player aiming in a slightly incorrect direction.

Additional Notes

For weapons that fire projectiles, lag compensation is more problematic. For instance, if the projectile lives autonomously on the server, then what time space should the projectile live in? Does every other player need to be "moved backward" every time the projectile is ready to be simulated and moved by the server? If so, how far backward in time should the other players be moved? These are interesting questions to consider. In HALF-LIFE, we avoided them; we simply don't lag-compensate projectile objects (that's not to say that we don't predict the sound of players firing the projectile on the client, just that the actual projectile is not lag-compensated in any way).

In general, lag compensation is an effective tool to ameliorate the effects of latency on today's Internet-enabled action games. The decision of whether to implement such a system rests with the game designer, since the decision directly changes the feel of the game. For HALF-LIFE, TEAM FORTRESS, and COUNTER-STRIKE, the benefits of lag compensation easily outweighed the inconsistencies.

FOR MORE INFORMATION

WEB SITES

QUAKEWORLD Source Code
ftp://ftp.idsoftware.com/idstuff/source/q1source.zip

HALF-LIFE SDK
http://download.cnet.com/downloads/0-10045-100-3422497.html

ARTICLES

Internet Cheating in Games
Pritchard, Matt. "How to Hurt the Hackers: The Scoop on Internet Cheating and How You Can Combat It" (June 2000).
Also at www.gamasutra.com/features/20000724/pritchard_01.htm

Return

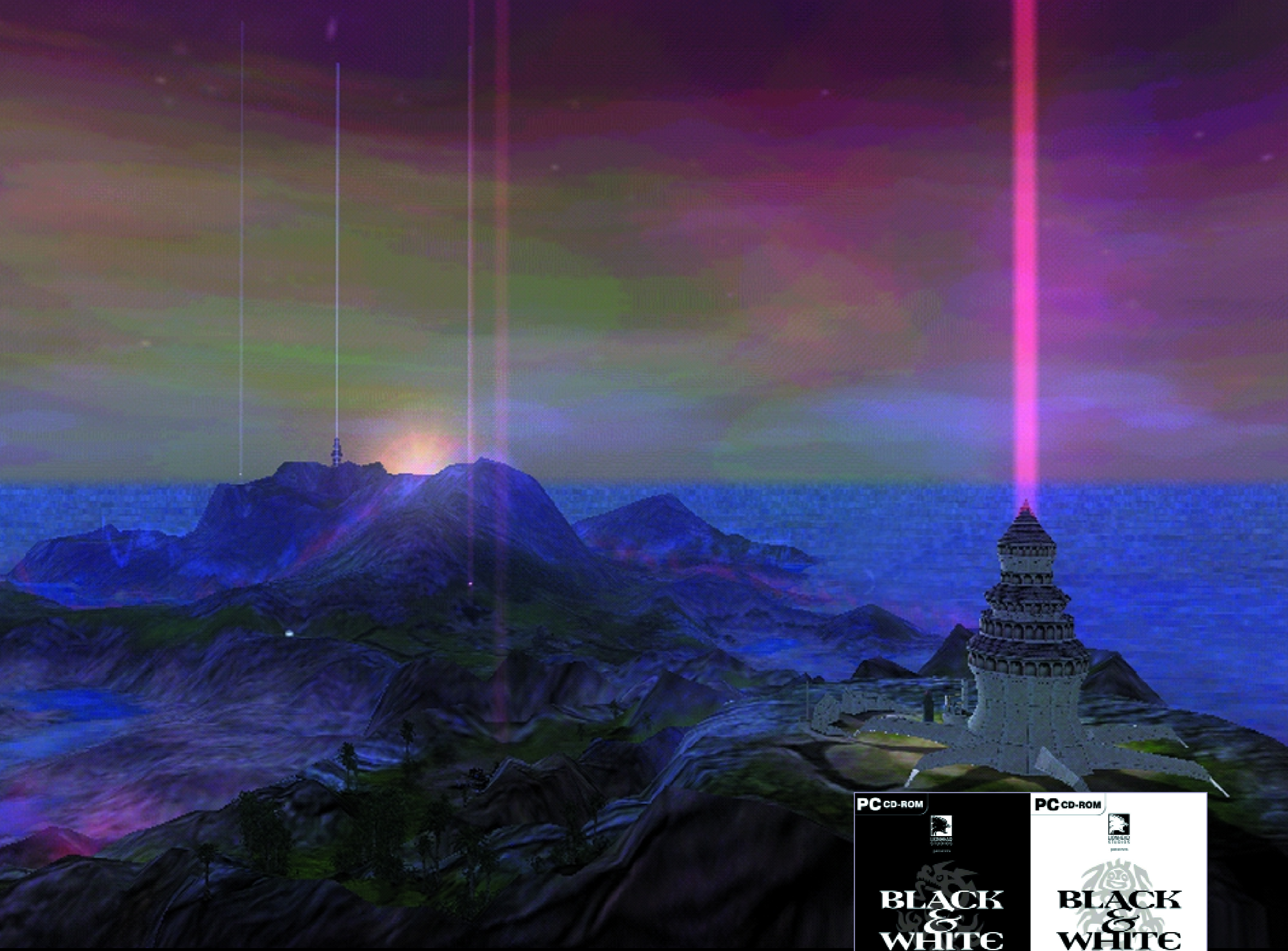
Lionhead Studios' Black & White

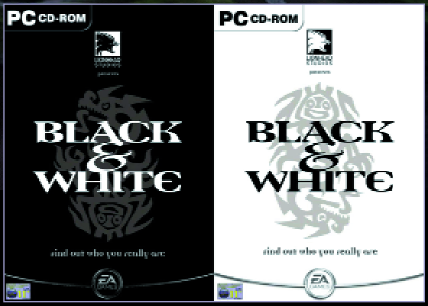
PETER MOLYNEUX | *Peter co-founded Bullfrog Productions in 1987 and created a new genre of computer games, the "god game," with the release of* POPULOUS*. Since then, Peter has been responsible for a string of massive-selling games, including* POWERMONGER*,* THEME PARK*,* MAGIC CARPET*, and* DUNGEON KEEPER*. He founded Lionhead Studios in 1997, whose first game,* BLACK & WHITE*, was released in March of this year.*

BLACK & WHITE is the game I always wanted to make. From the days of POPULOUS I had been fascinated by the idea of controlling and influencing people in an entire world. I was also interested in the concepts of good and evil as tools the player can use to rule or change the world. These themes crop up regularly in my games, but I realize now that with every game I was heading toward my ultimate goal — the god game BLACK & WHITE.

I wanted the game to be more flexible, more open, and more attractive than anything I'd ever played. I was determined that the player could do almost anything he or she wanted. Instead of leading players deeper into a world of levels and testing them with tougher and tougher monsters, I wanted players to be engaged by the story but to take it at their own pace and decide which bits to tackle and when to tackle them.

More technically, I didn't want a panel of icons or a set of on-screen options. With





DUNGEON KEEPER I felt we overdid the control panel, and, while it worked, it didn't add to the immersive sense of being this evil overlord deep underground. Frankly, it simply reminded you that you were playing a videogame.

Finally, I wanted to place into BLACK & WHITE the ability to select a creature (originally any creature from the landscape) and turn it into a huge, intelligent being which could learn, operate independently, and do your bidding when you wanted. I knew that this would require an artificial intelligence structure unlike any ever written. It had to be the best.

Of course, I needed a team for all this, but I wanted the right sort of team and so had to build it slowly. A core team of about six was formed, and at the start of Lionhead we worked at my house. Our first task was to create a library of tools, so we spent our time there doing boring foundation tool-building.

GAME DATA

PUBLISHER: Electronic Arts

FULL-TIME DEVELOPERS: 25

CONTRACTORS: 3

BUDGET: Approx. £4 million (approx. $5.7 million)

LENGTH OF DEVELOPMENT: 3 years, 1 month, 10 days

RELEASE DATE: March 30, 2001

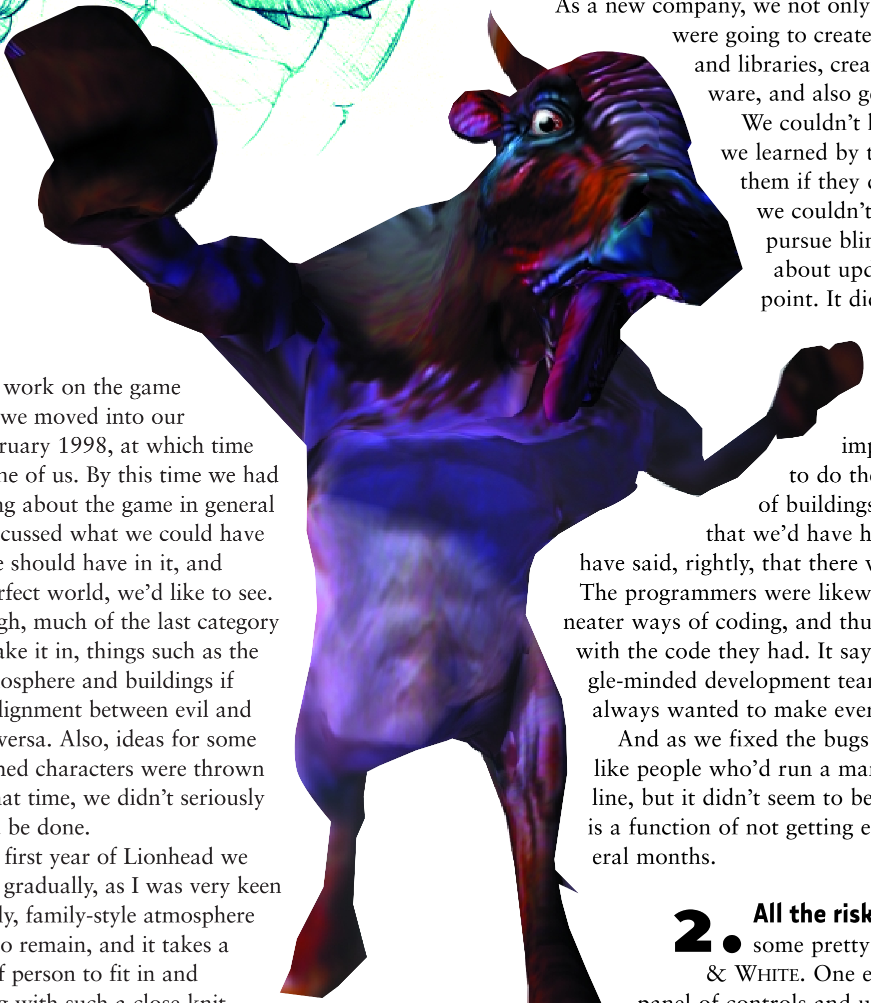
PLATFORMS: Windows 95/98/2000/ME

HARDWARE USED: 800MHz Pentium IIIs with 256MB RAM, 30GB hard drives, and Nvidia GeForce graphics cards

SOFTWARE USED: Microsoft Dev Studio, 3ds Max

NOTABLE TECHNOLOGIES: Bink for video playback, Immersion touch sense for force-feedback mouse

PROJECT SIZE: Approx. 2 million lines of code



ABOVE. Concept drawing for the tortoise Creature. RIGHT. Render of the evil cow Creature.

At Lionhead Studios, we all knew that BLACK & WHITE was going to be something special. This belief became self-fulfilling as we were inspired by each new feature and every neat, innovative section of code. Naturally, this meant that everyone worked exceptionally hard. Over the course of the project the team did the work of a group twice their number. We regularly went home as dawn broke, and weekends became something other people did.

We started work on the game proper when we moved into our offices in February 1998, at which time there were nine of us. By this time we had begun thinking about the game in general terms. We discussed what we could have in it, what we should have in it, and what, in a perfect world, we'd like to see. Funnily enough, much of the last category did in fact make it in, things such as the changing atmosphere and buildings if you change alignment between evil and good or vice versa. Also, ideas for some fully lip-synched characters were thrown around. At that time, we didn't seriously think it could be done.

During the first year of Lionhead we added people gradually, as I was very keen for the friendly, family-style atmosphere of Lionhead to remain, and it takes a certain sort of person to fit in and enjoy working with such a close-knit team. This policy of only recruiting people whom we felt had the talent and a way of working which fit in with Lionhead's existing members meant that our team had evolved their own way of working. They didn't just carry out their tasks but questioned, tested, and pushed both themselves and each other. It's labor-intensive, but you often end up with more than you expected. For example, the art team divided up the tribal styles for the villages and tried to outdo each other in terms of design and effort put in. The result was better design work than we thought we'd get.

What Went Right

1. It got finished. This sounds stupid, but we encountered some big problems, and there were times when we doubted that the game (as it ultimately ended up) would get released. As a new company, we not only had to work out the game we were going to create, but we had to write the tools and libraries, create everything from scratch in software, and also gel together as a team.

We couldn't have a dress rehearsal for this, so we learned by trying things and then changing them if they didn't work. As time rolled on, we couldn't afford to make any mistakes or pursue blind alleys. For example, we talked about updating some of the graphics at one point. It didn't seem a big job, but once we'd changed some of the buildings in the tribal villages, they showed up any unchanged ones and made them look less impressive, so we had to assign time to do them all. We got a much better set of buildings out of this, but if we'd known that we'd have had to do all of them, we would have said, rightly, that there wasn't time.

The programmers were likewise coming up with neater and neater ways of coding, and thus trying to do more and more with the code they had. It says a lot about the talented and single-minded development team at Lionhead that everybody always wanted to make every element that little bit better.

And as we fixed the bugs and sent the game to QA, we felt like people who'd run a marathon and could see the finish line, but it didn't seem to be getting any closer. Perhaps this is a function of not getting enough sleep over a period of several months.

2. All the risks paid off. We wanted to do some pretty groundbreaking things in BLACK & WHITE. One example was doing away with the panel of controls and using the Gesture system for casting Miracles. We tried and tried to get this feeling just right, and if we'd had to dump it, I'd have been so disappointed. But after research, testing, and simple trial and error we got it working beautifully, and we now have a feature no one else does.

Also, integrating the story line into such a free-flowing strategy game was a risk. We thought it would sit quietly behind the game, popping up to direct you if you hadn't moved on, but the story came alive and started to draw the player through the game in a way none of us, apart from perhaps

LEFT (top and bottom). Concept art featuring Horny from DUNGEON KEEPER — a great deal to live up to. RIGHT. The final product.

scriptwriter James Leach, had envisaged. It also gave us characters such as Sable, the Creature trainer, and those advisors whom we hear people now quoting lines from, and who exist outside the game as recognizable characters.

The huge, learning, intelligent Creature was also more of a gamble than he now seems. To go into AI in such an in-depth way required Richard Evans, our AI programmer, to consider what learning was, how practice works, and how the reinforcement of ideas comes about. Then he built all this into a character which appeared to live and learn like, say, a clever puppy. AI is always a minefield, and I'm always disappointed by great strategy games which appear to have the most simple, easy-to-predict AI running your enemies. We just wanted to advance the technology to its extreme.

We also wanted to do more with graphics and animation blending. The world changes depending on whether you're playing as a good or an evil god, and things take on subtle new looks. The Creature, the player's hand, and many of the buildings change, and we used more animation blending to achieve smooth movement and changes than anyone else has ever done, I believe.

We're also the first game (apart from Microsoft's FLIGHT SIMULATOR) which enables you to import real weather in real time into the world. We are also the first to enable unified messaging, whereby you can send messages to the web from the game, or receive them, using e-mail and mobile phones. This integrated two-way messaging as well as the ability to take your Creature out of BLACK & WHITE and onto the web is brand-new. Also, the game can import names from your e-mail package and assign them to unique villagers in your tribe in the game. I expect lots of games to do similar things in the future, but we took massive risks and devoted huge amounts of effort to being the first and to making it work properly.

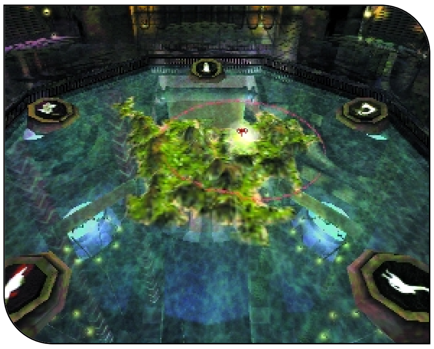
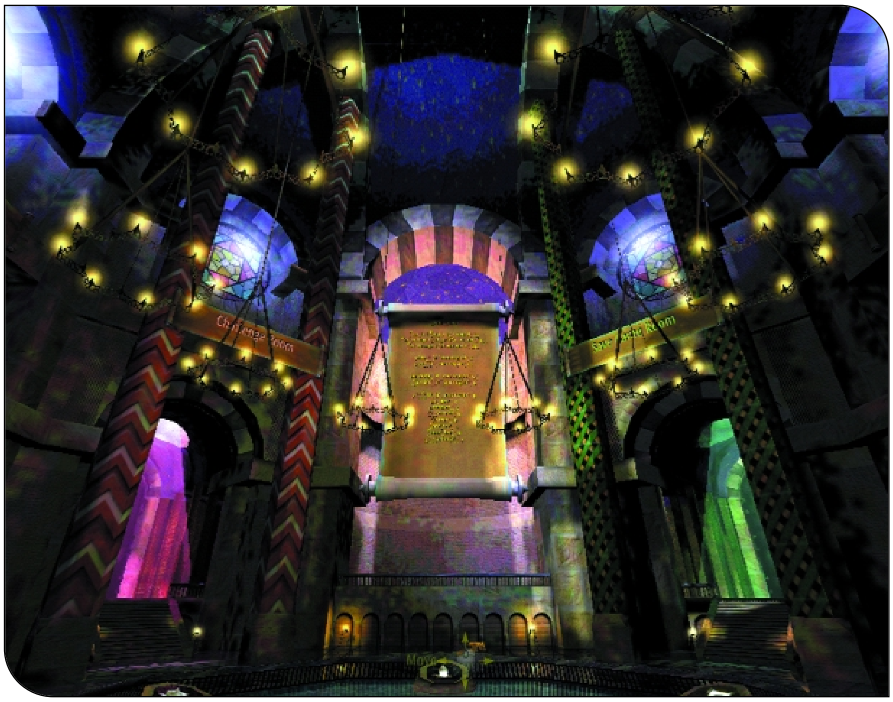
3. The game looks so stunning. When we started, we used a wireframe test bed and a couple of conceptual screenshots to provide some atmosphere. I first showed the test bed and these mocked-up screenshots to the press at E3 in Atlanta in 1998, and I could see on the assembled faces that nobody believed we could accomplish anything like it in the final game. I was complimented on the depth and beauty of preliminary efforts, but the compliments had a slightly hollow ring. I could almost hear people thinking, "Yeah, it looks great, but anyone can draw pretty screens using an art package. What's your game really going to look like?"

Not only did we manage to pull off the look we wanted, but we exceeded it by some margin. The sheer beauty of the lands is something I hope won't be matched for a while, and the fact that you can move, zoom, and rotate to view it from any angle, anywhere in the game, is again something we got spot-on.

Looking back, I don't know whether we were insanely ambitious, because at the time we started, you couldn't have done what we did. We needed so much custom-written software, and we also needed the minimum specification of the PC community at large to get better before this would be viable. When we started BLACK & WHITE, most people had 32MB of RAM in their PCs. The game requires 64MB, but that's commonplace now. So, if you like, we aimed beyond the horizon, and the world rotated and caught up with us so we still hit our target.

I still have those original screenshots, and I still like looking at them. We wrote a book called *The Making of Black & White*, and from reading that, it's clear that we went from a bunch of bizarre ideas linked by the concept of supreme control to the best game I have ever seen.

4. The artificial intelligence. The Creature AI, as I have mentioned, is absolutely spot-on. Richard Evans worked

ABOVE. The citadel, inside (right and bottom left) and out (top left), turned out better than we dared hope.

tirelessly on this, and it became something that surprised even him with its flexibility and power. The AI isn't just restricted to the Creature. Every villager in the game has it as well, and they are all different in their wishes, desires, motivation, and personality. Because there is no upper limit to the number of villagers you can have, we had to cap the AI slightly by giving some of the villager control to the Village Center, which acts like a hive and farms out some of the cooperative elements to the people. We couldn't have them interrogating each other, so this central control means that they do work as a unit but can retain their individual characteristics. This makes the game much faster and still gives them minds of their own.

The Creature himself is an astonishing piece of work. Once he starts learning, he forms his own personality as he goes, and no two players will ever have the same Creature. The complexity is kept to a minimum to keep him fast, but we managed to steer completely clear of using random elements to make him seem like he has a mind of his own. And there is nothing in the game that you can do which you can't teach your Creature to do. It's true to say that the Creature mirrors you and your actions, so in BLACK & WHITE we've got a game in which part of the game itself learns from everything you do and tailors itself to you.

5. The way the team came together to make BLACK & WHITE happen. This is Lionhead Studios' first project, and everything started from scratch. The people, the software, and the working environment were all new. Although this was exactly what we needed to do a game so fresh and diverse, it also created problems which I was delighted to overcome. The lack of any precedent meant that things took a lot longer than they should have, and the open-ended nature of the game throughout much of its development meant that team members were limited only by their own imagination.

But the nice thing is, every member of the Lionhead team gelled brilliantly, and although I know we picked the very best people, there is an element of luck in whether they can all work together so well. We certainly lucked out with the team, and every one of them contributed massively to making the game what it is.

The last few months of the project were the hardest any of us has ever had to work, but thanks to the people, they were also some of the most fun months we ever had. If nothing else, we'll always remember the time we spent closeted together making BLACK & WHITE.

And I'll never forget that without the right team, this game never would have happened. It's as simple as that.

What Went Wrong

1. Planning the story. We underestimated how long it would take to construct and write the story element of BLACK & WHITE. The free-form nature of the game required an unfolding tale to give it some structure and lead it to a conclusion, and in October 1999 we began to work on the story. We thought it would take no more than two months, but after a while we realized that we didn't have the skill set needed to take care of this vital aspect of the game. I contacted James Leach, who'd been the in-house games scriptwriter at Bullfrog and had worked on SYNDICATE WARS, DUNGEON KEEPER, THEME HOSPITAL, and many others. He was working as a freelance ad copywriter but gladly came on board, again in a freelance capacity, and turned our ideas into a fully plotted story line, wrote hundreds of challenges and quests, and wrote all the dialogue in the game. It ended up being more than 60,000 words, the size of a novel.

Hiring James meant that we got a sense of continuity, consis-

LEFT. Creature comforts. RIGHT (top and bottom). We tried to make the micromanagement of the villagers as user-friendly as possible.

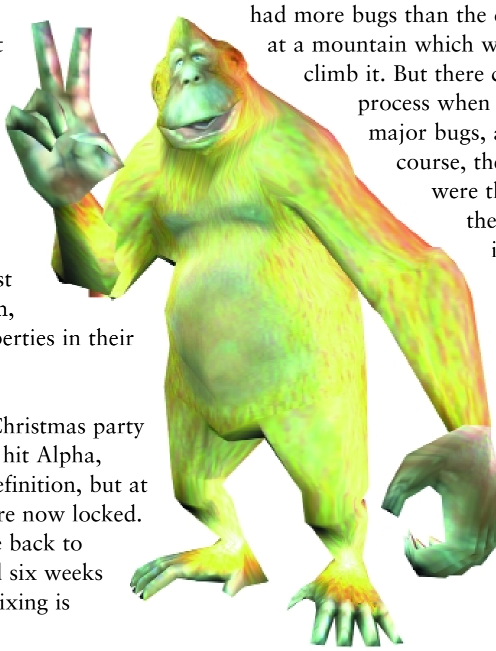
tency, and style throughout the game. It also meant that we could describe what we wanted, or even write placeholder text, and he would rapidly turn it into finished work. Sections of the game that were still at an early stage seemed more easy to understand, get a feel for, and work on when we used dialogue and text which seemed, to us, finished. Of course, another pass was usually needed to make it accurate and sometimes to polish it, but having a dedicated scriptwriter made this a simple task.

Storytelling in games, as elsewhere, is an art. If a story line flows easily and naturally, that's because someone has worked incredibly hard at it. I'm a great believer in the emotion and immersion that can be added to a game through good story and dialogue. It can't make a bad game good, but it can make any game better. And when the script was looked at by Hollywood scriptwriters and film directors from the BBC, we knew we were on to a winner.

Another by-product of using a professional scriptwriter was that we morphed the in-game advisors, the good and evil guys, from being just sources of information and guidance into stylish, popular characters who are now bankable properties in their own right.

2. Fixing the bugs. After canceling our Christmas party on December 26, 2000, we managed to hit Alpha, which as any developer knows is a very loose definition, but at least we could say that all the game features were now locked. After a well-deserved Christmas break, we came back to find that we had more than 3,000 bugs. We had six weeks to reduce this to zero, but the thing about bug-fixing is that you can solve one problem but in doing so create three more. So although we worked as hard as we could, the overall figure crept down slowly rather than dropped at the rate at which we were actually sorting out the bugs.

By this stage the team was very tired. The only things that kept them going were the sense that the end was in sight and the fact that they could now play the game and actually experience what we had created. Bugs, of course, could have killed the game, so there was no way around it but to fix each and every one. We had bug lists circulated to every member of the staff, and we put up a chart on the wall which was updated daily. Some days we had more bugs than the day before, and that was like looking at a mountain which was growing quicker than we could climb it. But there came a moment three weeks into this process when we felt we'd broken the back of the major bugs, and the numbers fell steadily. Of course, the irony was that the last 10 bugs were the hardest to fix, and with every one there were four more created. It was as if the game just didn't want to be finished and perfected.



Good ape.

3. The project was too big. BLACK & WHITE got to be so large that we almost felt lost within the code. In fact there are well over a million lines of code within the game. Loading up even the most simple of the smallest tools would take many minutes,

ABOVE. Tortoise morphs from evil to good. BELOW. Concept sketch of the good Celt.

and compiling the entire game took over an hour. This meant that toward the end of the development phase even a tiny change could take a whole day to implement.

Checking in changes and rectifying errors was a nightmare. We eventually decided to limit the checking-in to one machine, and we implemented a buddy system whereby nothing was done without an onlooker checking it at every stage. This put a stop to tired people checking in changes at four in the morning and finding that, instead of fixing something, they'd actually caused further problems.

Another worry about the project's size was that we didn't think the game would fit on one CD, although we were desperate for it to. The audio files are immense. Music, dialogue, and effects are all compressed, but of sufficiently high quality that we refused to reduce them any further.

And with 15 language versions to get translated and recorded, we had to do the biggest localization job I've ever seen. This landed on Lionhead Studios at the very busiest time, and although our publisher did an excellent job of handling it, we were needed to check and answer questions and to provide explanations for some of the more arcane elements of the game.

4. Leaving things out. The idea of the game didn't really change much over the course of its creation. But I do have some regrets that features we thought would be great proved unworkable. I expected this, as it happens with every project, but I thought the problems would be caused by software or even hardware limitations. In fact, it came down more to emotional issues.

For example, the original idea of the Creatures was that a player could choose to make any living thing a Creature. We wanted the player to be able to select an ant and grow that, or a human being from a tribe, and raise him or her. Christian Bravery, one of the artists, spent a long time drawing concept work and sketches depicting what the Creatures could look like at various stages of their development. This of course included humans.

We soon realized that people would have certain expectations from a human. Players wouldn't expect a turtle to learn as quickly as a man, but if we dumbed down the people, they'd seem like a proto-hominid race from eons ago, and we didn't want that. Also, discipline in the game involves slapping your Creature. We certainly couldn't have the player slapping a child or a woman or, really, even a grown man. The emotional feel of raising a human, teaching him or her to eat what you want, and leading him or her around in a speechless environment was all wrong.

Christian's work in visualizing humans as player Creatures was all for nothing in the end, and we dropped the idea. We also dropped the notion of turning any living thing into a trainable Creature, as ants, butterflies, fish, and other nonmammals would have caused big problems. A flying Creature would change BLACK & WHITE into a totally different game.

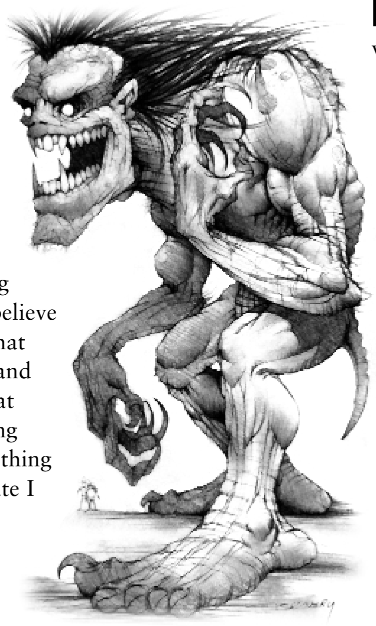
I also regret that we couldn't use color as a dynamic concept a little more. The landscapes in the game are gorgeous, and our sound and music man, Russell Shaw, suggested that various spells could drain the color out of areas, or spread different colors around. We liked this idea for its surrealism, and

ABOVE. The game's underlying detail is immense but never overwhelming. BELOW. Concept art of the evil Celt.

we thought about having color wars with other wizards (at this stage you weren't a god, you were a wizard battling others on a land). The idea lost momentum when we thought about how the land would actually look, and how it would seem like something drawn by a preschooler. I still like the idea of color wars, but I think children's TV has also cottoned on to the idea, which means we won't be going there.

5. Talking about release dates. I have to admit, ruefully, that I have a reputation for being, shall we say, optimistic about when the projects I'm working on will be completed. I opened my big mouth and announced that Lionhead Studios would finish Black & White and get it released at the end of last year.

I just can't resist talking about whatever I'm currently working on. This has been a problem I've experienced with every game I've ever developed. But the thing is, when I think something is going to be finished in December, I really do believe it. People at Lionhead were telling me that we had to build in time for bug-fixing, and I knew this was true, but the truth is that there seems to be no formula for working out how long things will take. The best thing to do, I guess, is to take the finishing date I first think of and move it twice as far away — and then not announce it until we're halfway there.



It's a function of working on products which could literally be endless. Unlike a film, where once the footage is shot, you edit it with an idea of where you'll end up, you can add completely new features to a game and then balance it and change it radically right up until the last minute. I'm sure that there were many people who didn't believe me when I said we'd finished making Black & White and were only convinced when they saw a box with a CD in it.

"Just More"

Black & White is unlike any other game ever written. That was our goal, and we achieved it. We wanted something more beautiful, more complex, more emotive, more innovative, more clever, and more, well, just more.

As you've read, it was beset by problems. We nearly drove ourselves crazy solving them. Nothing worthwhile is ever simple, though, and for every minute spent thinking up wonderful ideas to include in the game, there were probably 20 hours of sheer hard effort trying to get them to work.

People told Lionhead we were perfectionists, but if we were, the game would never have been finished. It's not a perfect game. Our next game won't be, either. But because there's no such thing as a perfect game, we'll just try to do something different, and do it as well as we possibly can. Someone asked me recently what drove us to work so hard on this and to spend so much time thinking outside the box. The simple truth of my answer only struck me afterwards. With Black & White, we made the game that we wanted to play.

Return

The Passing of a Legend

Origin Systems has always been to me like some Ivy League college, an institution beyond my reach or means. Many used to refer to it as "Origin U," a place where young, creative people went to learn if they had the chops to make games. Attending Origin U required starting at the bottom — usually in the QA department or as a technical design assistant — and working outrageous hours for very little pay. A college degree was as helpful in getting into Origin U as it would be at an Army boot camp — more of a handicap than anything else. Making games was the Wild West of computer software, and code was written by mighty heroes supported by groups of wide-eyed apprentices longing for the day they too would become heroes. When they "graduated," they quit and founded dozens of start-up game companies over the years, turning Austin into a hotbed of computer game development.

I wanted to make games, but I wanted a living as well, so instead of attending Origin U, I took my first full-time job across the street at a graphics company doing drivers for AutoCAD and 3D Studio for DOS. It wasn't the most exciting work I could imagine, but it paid decently and was a great learning experience. I just didn't think I was in a good position to break into the game industry. I already had a computer science degree and was starting work on a master's with support from my employer, and I had a lot of things to learn about writing software for the PC. Ironically, it was through an ex-Origin employee I met at that job that I found myself a co-founder at a game company start-up a few years later.

In many ways, Origin Systems defined the game industry in its youth. It was founded by Richard Garriott, who was programmer/designer/artist/sound designer and created games sold in plastic bags. Origin mushroomed over the years with the successes of the ULTIMA series, WING COMMANDER, and a slew of other hits into a behemoth employing 150-plus people. Whenever the Austin journalists wanted to do a story on computer games, it was Origin Systems they turned to. Richard was the classic eccentric millionaire and poster child for young success via technology. The distribution muscle and money that Electronic Arts commanded meant that the Origin brand was everywhere, on consoles and PCs. Without that kind of money, WING COMMANDER 3 would probably have been the *Plan 9 from Outer Space* of computer games, or at least a lot shorter.

With the recent layoffs of the entire ULTIMA ONLINE 2 team, cancellation of all in-development projects at Origin, and the death of the Origin brand, an era has come to an end. Even the employees who once worked at Origin recognized it as a watershed event, holding a "wake" in Austin by the lakeshore and burning UO2 design documents in memory of the place that was. Sure, the OSI offices are still here, and ULTIMA ONLINE continues to cater to an addicted hardcore audience willing to shell out $10 a month for their chance to be fantasy heroes, but a company with no future projects isn't long for this world, and everyone knows it.

It certainly seems a sad ending for one of the founding institutions of our industry. Some will no doubt blame Origin's demise on the "suits" at Electronic Arts, but I suspect the company culture

continued on page 71

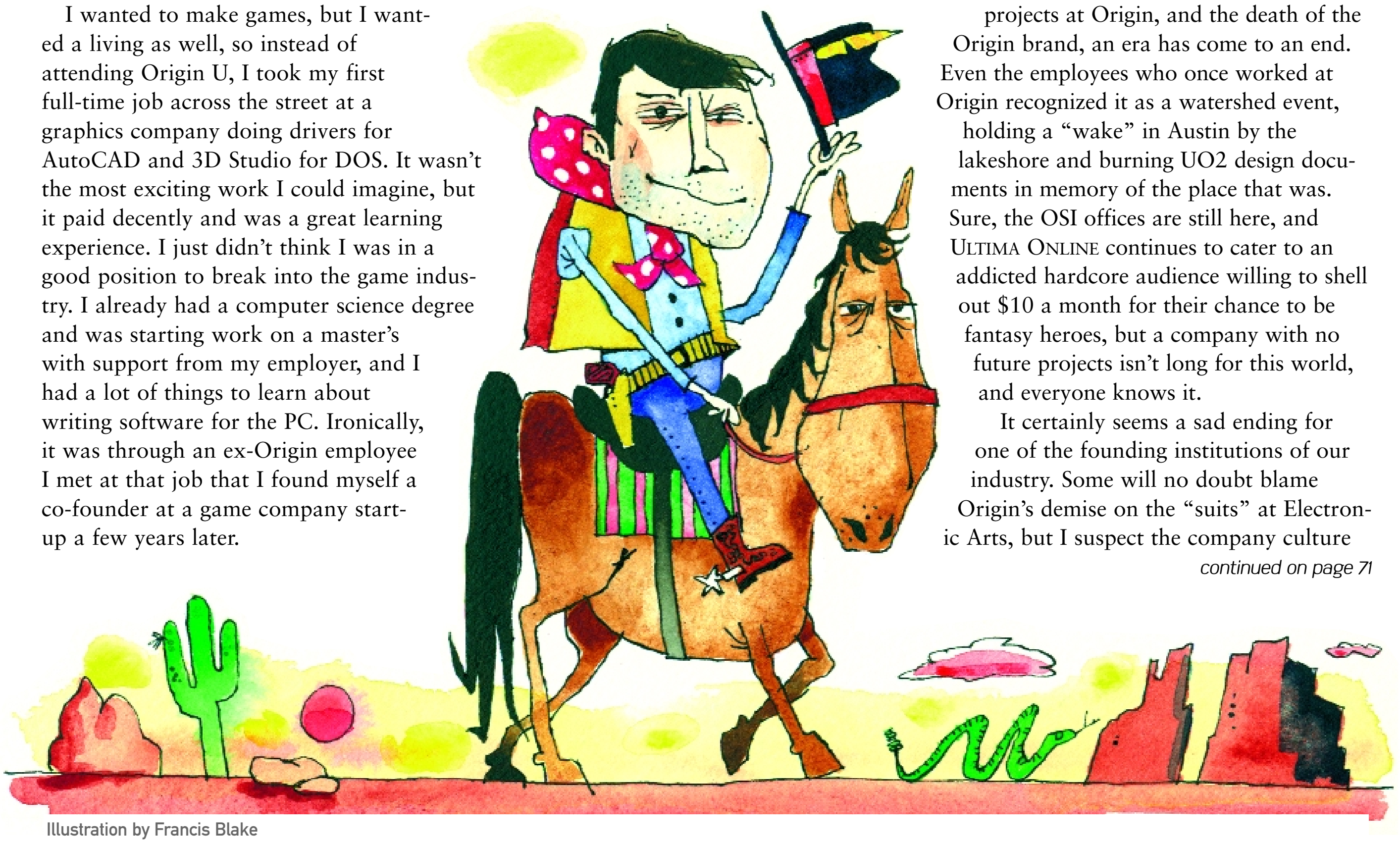
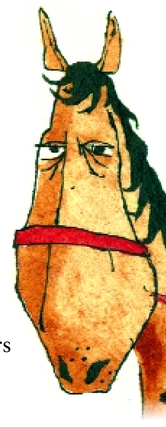


Illustration by Francis Blake



continued from page 72

was also to blame. There is no doubt that Origin's development teams demonstrated creativity and vision aplenty, but according to many ex-Originites I have known over the years, the teams were often plagued by a lack of engineering discipline and productivity, relying heavily on the efforts of a dedicated few. There is probably an ex-Origin employee in your organization who can attest to this.

The computer game industry is full of people who long for the glory days when content was king and developers controlled the project. The industry has changed since those days; projects are larger and more complex, games are written with interdisciplinary teams from a wide range of backgrounds, and the platforms are far more complex and powerful. For now, the publishers and other large companies are the only ones who have the financial resources we need. Creativity and vision are as important today as they were back then, but the industry also needs professionalism, engineering discipline, and responsibility. Budgets and schedules may be a drag, but we are spending other people's money, and it is incumbent on us to spend it wisely.

More importantly, we should not use nostalgia as an excuse for making the same mistakes over and over again. If you are writing games the same way you were in the 1980s, then you are doing your investor and yourself a disservice. The platforms have evolved, the tools have evolved, and the projects have evolved, so certainly developers should as well. The age of heroes is past, and while we remember the buzz of those wild days, we should also remember the wasted human effort, personal lives, and money which could have gone to worthy projects instead of keeping developers in the manner to which they had become accustomed.

CHUCK WALBOURN | *Chuck is currently director of technology at Kinesoft Development, developing tools and software components for use on Kinesoft's PC title CRIMSON ORDER and other projects. He was previously a co-founder of Charybdis Enterprises Inc., a small Austin-based PC developer which created two original PC titles (INTERACTIVE MAGIC and FIGHTER PILOT) for Electronic Arts. He can be reached at chuckw@kinesoft.com.*