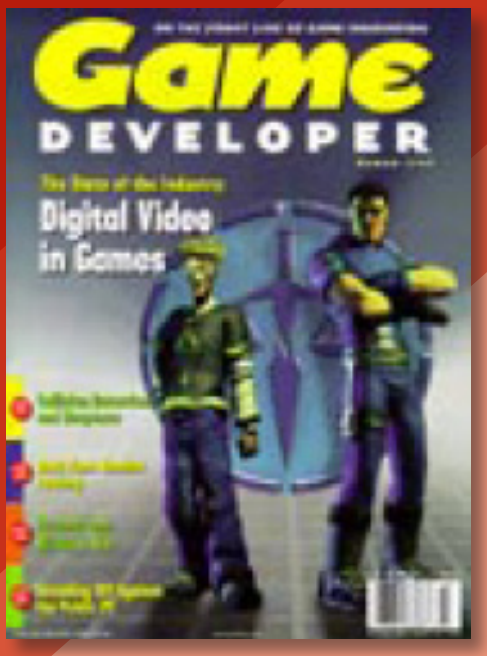
gd

GAME DEVELOPER MAGAZINE

MARCH 1999



UBM
Tech

"Stick to Shorts"

In a young entertainment medium that has seen such tremendous growth as ours has, I sometimes think there's a tendency for everyone to expect radical leaps forward every year. For instance, this magazine's sister web site, Gamasutra, posed a question in December which simply asked, "What is your vision for 1999?" Being a typical, year-end utopian type of question, I figured that there would be a wide range of responses. I was fairly shocked when about half of them lamented current game designs and pined for more innovation from the industry. This sentiment seems to be shared widely among consumer magazines, consumers, and to some extent, even within developer circles, so it deserves to be addressed in this magazine.

What can be done to combat creative stagnation — or even the perception of such stagnation? That's a difficult question. Though it's easy to point fingers at various causes, one can't look at the current state of gaming and say that where we are today is the result of any one particular force, be it ourselves, game publishers, consumers, game reviewers, or the limits of technology. Each year, thousands of games are released by thousands of game development teams, and it's oversimplifying things to say that one force is at work shaping what comes out of our industry.

All that we can do is to strive to create better games. It's that simple. This industry has lost some extremely talented developers who got burned, depressed, angry, or bitter that the medium of interactive games wasn't living up to their expectations. I don't think we can afford to lose more creative talent, and if you find yourself smarting from a bad review of your game, a snide comment from a player online, poor sales of your game, or some other form of negativity, it's your responsibility to recharge and reinspire yourself.

To this end I recommend the book *The Illusion of Life: Disney Animation* (Hyperion, 1995) by Frank Thomas and Olie Johnson. Chris Hecker told me about this book early last year, and after reading it, I can see what spurred his praise. After plowing through dozens of messages complaining about stagnant game designs which thoroughly depressed me, this book was a welcome dose of inspiration.

In particular, one anecdote made me realize that it is the responsibility of creative professionals to fight negativity. The year was 1937, and after producing a number of successful animated shorts, Disney studios was embarking upon its first full-length animated feature, *Snow White*. After an early screening of Snow White for some of the Disney staff, someone wrote "Stick to Shorts" on an unsigned questionnaire and submitted it. The comment really stuck in Walt Disney's craw, and after the success of the film, he used the phrase to illustrate his disdain for poor judgement. According to Thomas and Johnson, if you raised a bad idea in a meeting, "suddenly there would be this loud, 'Ah haaa!' and Walt's finger would come shooting out toward you; in a triumphant voice he would explain, 'You must be the guy who said "Stick to Shorts!"' And for that day you were the guy, and everyone else would keep looking at you and wondering." Disney was a visionary, and he didn't need some nay-sayer in the ranks of his own organization decrying the admittedly risky venture.

I do not think that the current state of game development is as dismal as some paint it, despite those that decry the marginal rate of innovation in game design. For goodness sake, we have a lot to be proud of. A game earned more revenue in the U.S. over the last six weeks of 1998 than any movie during the same period of time! Quite an accomplishment, indeed, and a benchmark I'm sure that we'll get used to seeing — especially if we don't "Stick to Shorts." ■

Game Developer
ON THE FRONT LINE OF GAME INNOVATION

600 Harrison Street, San Francisco, CA 94107
t: 415.905.2200 f: 415.905.2228 w: www.gdmag.com

Publisher
Cynthia A. Blair *cblair@mfi.com*

EDITORIAL

Editorial Director
Alex Dunne *adunne@sirius.com*
Managing Editor
Tor D. Berg *tdberg@sirius.com*
Departments Editor
Wesley Hall *whall@sirius.com*
Art Director
Laura Pool *lpool@mfi.com*
Editor-At-Large
Chris Hecker *checker@d6.com*
Contributing Editors
Jeff Lander *jeffl@darwin3d.com*
Mel Guymon *mel@surreal.com*
Omid Rahmat *omid@compuserve.com*
Advisory Board
Hal Barwood LucasArts
Noah Falstein The Inspiracy
Brian Hook id Software
Susan Lee-Merrow Lucas Learning
Mark Miller Harmonix
Paul Steed id Software
Dan Teven Teven Consulting
Rob Wyatt DreamWorks Interactive

ADVERTISING SALES

Western Regional Sales Manager
Alicia Langer *alanger@mfi.com* t: 415.905.2156
Eastern Regional Sales Manager/Recruitment
Ayrien Houchin *ahouchin@mfi.com* t: 415.905.2788

ADVERTISING PRODUCTION

Vice President Production Andrew A. Mickus
Advertising Production Coordinator Dave Perrotti
Reprints Stella Valdez t: 916.983.6971

MILLER FREEMAN GAME GROUP MARKETING

Group Marketing Manager Gabe Zichermann
MarComm Manager Susan McDonald
Marketing Coordinator Izora Garcia de Lillard

CIRCULATION

Vice President Cirulation Jerry M. Okabe
Assistant Circulation Director Mike Poplardo
Circulation Manager Stephanie Blake
Circulation Assistant Kausha Jackson-Crain
Newsstand Analyst Joyce Gorsuch

INTERNATIONAL LICENSING INFORMATION

Robert J. Abramson and Associates Inc.
President Libby Abramson
720 Post Road, Scarsdale, New York 10583
t: 914.723.4700 f: 914.723.4722
e: *abramson@prodigy.com*

Miller Freeman
A United News & Media publication

Chairman-Miller Freeman Inc. Marshall W. Freeman
President/COO Donald A. Pazour
Senior Vice President/CFO Warren "Andy" Ambrose
Senior Vice Presidents H. Ted Bahr, Darrell Denny Galen A. Poss, Wini D. Ragus, Regina Starr Ridley, Andrew A. Mickus, Jerry M. Okabe
Vice President/SD Show Group KoAnn Vikören
Senior Vice President/Systems and Software Division Regina Ridley

BPA International Membership Applied for March 1998

www.gdmag.com

BIT Blasts

News from the World of Game Development

New Products

by Wesley Hall

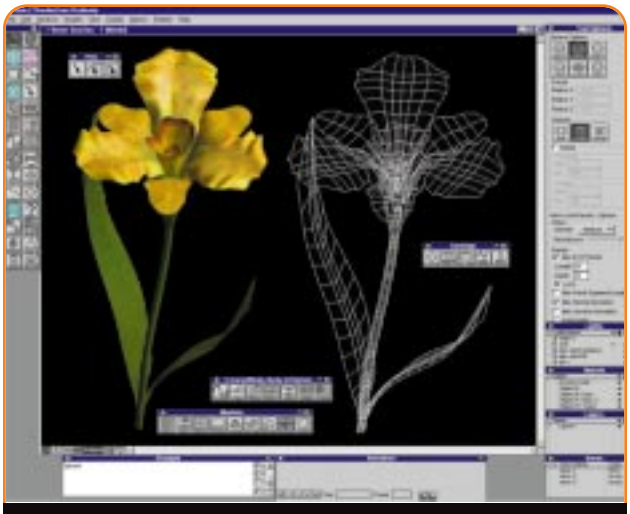
form-Z 3.0 Adds Animation

AUTO*DES*SYS INC. announced at MacWorld the release of form-Z 3.0, the latest version of their 3D modeling program that includes drafting and rendering.

The biggest leaps forward for form-Z 3.0 include objects with multiple parametric personalities, a fully customizable interface, and animation. form-Z now allows you to perform walk-throughs and fly-by animations, and to replay from within form-Z. You can export animations to .AVI and Quicktime. Additionally, there are many other new features including new spline drawing procedures, stair generation from any path, patch modeling, improved skinning, and centroids, local origins, centers of gravity, and coordinate axes that can act as bases for geometric operations.

The new 3.0 release is available for the MacOS, and Windows 95/98/NT/NT Alpha platforms. form-Z 3.0 can be purchased alone, or with the addition of Renderzone (render) and/or Radiozity (for advanced lighting effects). Alone, form-Z retails for $1,495; with Renderzone the price is $1,995; and form-Z Renderzone with Radiozity is $2,390.

- **auto*des*sys Inc.**
  Columbus, Ohio.
  (614)488-8838
  http://www.formz.com



The form-Z interface.

Sorenson Speeds Up Quicktime

SORENSON VISION INC. recently released a major upgrade to their video codec, Sorenson Video 2.0, and announced it at MacWorld SF in January 1999.

The 2.0 upgrade enhances the performance of the encoding and decoding process for Quicktime compatible CD-ROM, DVD, and Web-based applications. 2.0 encodes up to four times faster than the Sorenson's original codec. It also offers more flexibility for managing the quality/bandwidth tradeoffs of compressed video. This release also supports multiple-processor workstations and divides the workload among the available processors in order to speed up jobs. Improved motion handling further increases the quality and clarity of compressed video clips, and reduces the distortion and artifacts. Decoders for playback are built into all releases of Quicktime 3 or later.

Sorenson Video 2.0 is a Quicktime system extension that operates on any platform that supports Quicktime 3 or later. You can purchase the software directly from Sorenson Vision. Upgrades are $99. No developer royalties are required for applications created by Sorenson Video.

- **Sorenson Vision Inc.**
  Logan, Utah
  (408) 970-0696
  http://www.s-vision.com

Graphical User Interface SDK

CHILKAT SOFTWARE INC. released the downloadable Chlikat SDK 1.0 in January 1999.

This new, downloadable SDK will allow you to develop Graphical User Interfaces (GUIs) for your game that incorporate custom artwork and animations. The Chilkat SDK also provides visual components (C++/DirectX) which you can use to compose an application's interfaces. These include managers (row-columns, bulletin boards, clip windows, and so on) and primitives (labels and line edits). Animation classes exist so that visual components can use an animation anywhere a static bitmap might be used. Bitmap classes automatically handle pixel-format conversion, so games can run in any pixel format. Other features include a library that works with palette-sets, image tables to organize bitmaps and animations, and a wide variety of other classes.

The Chilkat SDK supports Windows 95/98/NT, DirectX 5.2 and 6.0, Borland and Microsoft compilers, and Microsoft DirectShow. It also supports MPEG and .AVI. It's licensed on a per-product, royalty-free basis. Pricing is set at $495 for a commercial single-product license, $95 for a shareware developer single-product license, and $45 for a non-commercial license.

- **Chilkat Software Inc.**
  Lisle, Ill.
  (312) 953-3949
  http://www.chilkatsoft.com

Industry Watch

by Alex Dunne

MATTEL MERGER. Mattel announced that it was merging with The Learning Company, a deal valued at a whopping $3.8 billion. The deal fueled speculation on Wall Street that more toy and game company deals in the industry would follow. Mattel CEO Jill Barad said that the merger achieved the company's goal of becoming a $1 billion interactive software business. The transaction is expected to close in March or April 1999 and will be considered a pooling of interests.

TAKE-TWO INTERACTIVE SOFTWARE has been busy in the news lately. First, the company announced that it completed the acquisition of TalonSoft in a stock swap transaction. TalonSoft founders John Davidson and Jim Rose will stay on with the Take-Two Interactive. Take-Two already owns and operates Mission Studios, GearHead Entertainment, Tarantula, and Alternative Reality Technologies.

Second, Take Two and 3000AD (a.k.a. the company Derek Smart built) settled their long-standing feud. Both sides issued a statement saying "...since we have been able to remain friends of sorts through all the occurrences of the past years, we concluded that it was just time to stop bashing each other and start supporting one another." Third, Take-Two issued its financial results for its fiscal year, and things look pretty rosy. The company more than doubled its net sales over the previous year, to $191 million, and saw $6 million in net income — better than their $3.6 million loss in the previous year.

STRATA AQUIRED. Strata, maker of the StudioPro 3D animation environment, was acquired by C-3D Digital. The technology and principal assets of Strata will be added to C-3D Digital's line of 3D technology for broadcast video and the Internet. C-3D Digital (http://www.3d.com) creates stereoscopic media for the Web and television (requiring traditional shutter glasses), and will air sporting events, live broadcasts, feature films and other original content in stereoscopic 3D. C-3D Digital launched the world's first television network to exclusively offer 3D programming to satellite dish owners, cable TV subscribers, and pay-per-view lodging guests.

SEEMINGLY NEVER OUT OF THE NEWS, GT Interactive acquired Legend Entertainment, the developer of MISSION CRITICAL and DEATH GATE, in a cash transaction. Legend becomes GT Interactive's seventh fully integrated software development studio. Legend, headed up by president Bob Bates, is working on the UNREAL LEVEL PAK and UNREAL II for GT Interactive, as well as WHEEL OF TIME, which uses the Unreal engine.

GT Interactive also entered into a publishing agreement with Infinite Machine, a new development company founded by former JEDI KNIGHT developers Justin Chin and Che-Yuan Wang. GT Interactive will publish Infinite Machine's first title, a 3D action game for the PC, which will ship next year. Under the agreement, GT Interactive obtains rights to console versions of the title as well as sequels, add-on, and level packs.

THE BALLYHOOED LAWSUIT by a group of five ULTIMA ONLINE players against Origin and its parent company, EA, was finally settled. The lawsuit, which alleged breach of contract, intentional misrepresentation, negligent misrepresentation, negligence, breach of warranties, and other charges stemming from problems with the game, was put to bed after EA agreed to make a $15,000 charitable donation to The Tech Museum of Innovation in San Jose, California. Both parties agreed to bear their respective legal costs, and agreed that the settlement was not an admission of liability on the part of EA/Origin or Schultz (attorney for the plaintiffs) and his clients. In other words, a Mexican standoff. In a statement after the settlement, EA said "...such frivolous lawsuits stifle innovation and threaten the creative community's efforts to bring new technologies to the consumer." Looks like the only winner was the museum.

The Tech Museum of Innovation in San Jose was the surprising beneficiary of the ULTIMA ONLINE lawsuit.

UPCOMING EVENTS CALENDAR

March 5-6, 1999

ASIFA-Hollywood's Animation Expo
Glendale Civic Auditorium
Glendale, Calif.
Cost: starts at $10
http://www.asifa-hollywood.org/expo.html

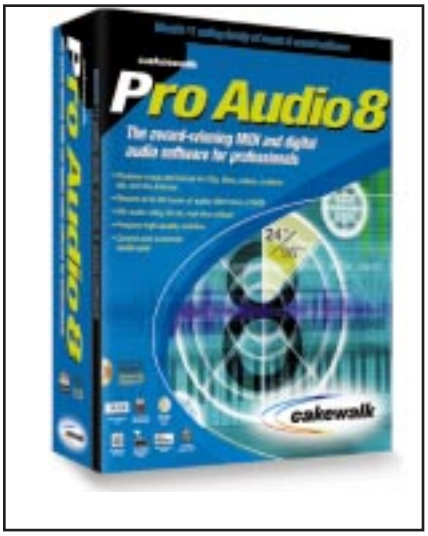
March 15-19, 1999

Game Developer's Conference
San Jose Convention Center
San Jose, Calif.
Cost: starts at $130
http://www.gdconf.com

Corrections

On page 10 of our 1999 Buyers Guide we mistakenly identified 4DPaint 1.5 as being manufactured by 4DVision. 4D Paint is manufactured by Right Hemisphere Ltd. of Auckland, New Zealand. Right Hemisphere can be reached at +64 (9) 309-3204 and the company's web address is http://www.righthemisphere.com.

In "Surveying the Digital Video Landscape" in the January 1999 issue, we incorrectly credited TrueMotion as the video technology used by Interplay in its Star Trek titles. Paul Edelstein of Interplay says that most company titles that include video use "proprietary movie technology owned by Interplay and referred to internally as Interplay's MVE format, of which I am the author." Apologies to Mr. Edelstein and the developers at Interplay.



Cakewalk Pro Audio 8

by Andrew Boyd

If any piece of music software deserves the descriptor venerable, it's Cakewalk's MIDI sequencer product. This thing has been around in one form or another since 1988. Its current form began taking shape in 1992 when it was released for Windows. It turned another corner in 1995 when digital audio features were added to its extensive set of MIDI tools. Cakewalk has done an impressive job of keeping up with or leading its competition in adding new features, refining functionality, and improving the interface, all while keeping prices very reasonable. I took a look at Cakewalk's latest top-of-the-line offering, Pro Audio 8 Deluxe.

I installed the product onto my primary Windows audio machine: a Pentium 200MHz MMX with 64MB of RAM, a Turtle Beach Pinnacle sound card, and an Adaptec Ultra-SCSI card running Windows 95. This machine had Cakewalk Pro Audio 6.1 already installed, and Pro Audio 8 found it and effortlessly updated the application. My other machine, a Pentium Pro 200MHz with 96MB of RAM, a Sound Blaster 16 card, with an Ultra-Wide SCSI card running Windows 98, held no previous versions of Cakewalk, so Pro Audio 8 stepped me through the setup process with little hassle. On both machines, I used a Mark of the Unicorn MIDI Timepiece AV for MIDI interface and synchronization duties, and Pro Audio 8 recognized it flawlessly.

INTERFACE. After you've launched the application, Pro Audio 8 presents you with a new, empty project file (using the default Normal template), or optionally, a very simple but handy window called Quick Start. Quick Start gives one-button access to New File, Open File, Open Recent (with a drop down list of choices), and Learn More…. At first, I turned off Quick Start, thinking I didn't need that sort of thing. Eventually though, I came to like it as, well, a quick way to start. If you choose to create a new project, you're presented with an impressive list of prebuilt (and user-created, if you've made any) project templates. Templates can store all kinds of settings for a new document, so once you figure out how you like to use the program, make up your own default startup template and save the time of making basic settings for each new project. The included templates set up General MIDI-compatible tracks for typical sessions of a given music type. "Classical Brass Quintet," for instance, creates five tracks pre-assigned to General MIDI patches: two for trumpet, one each for French horn, trombone, and tuba, each with their correct key offsets, channels, pan settings, and so on.

The interface is organized as a collection of dockable toolbars around a workspace where you maneuver the various display windows. The primary window is the Track view, which presents a very standard sequencer interface grid of tracks against time. Its adjustable vertical split places track numbers, names, and basic controls on the left, and displays the contents of the tracks — bits of MIDI and/or audio data here called clips — on the right. This arrangement makes immediate sense if you've ever used another multitrack piece of software (MIDI or audio), and should be easy to figure out even if you've never seen anything like it before. The clips display a mini visual representation of their contents, and are by default color-coded by type (and can be further differentiated by color on a clip by clip basis). Clicking on a clip selects it, dragging moves it, and [control]+dragging copies it. This view is comfortable and it's easy to work very quickly here. Interestingly, Pro Audio 8 doesn't provide any kind of bin for clips, so they must always be accessed here, from within a track.

In Pro Audio 8, form follows function when it comes to the interface. And while functionality is extremely important, the interface feels a littler underpolished. When I'm working in Digital Performer (my main ax), I feel as though I'm enveloped in a comfortable, creative space. Somehow, Pro Audio still feels a little clunky and boxy, more like a "tool" than an "instrument," if you will. Every button, dialogue box, and slider in Digital Performer is custom art, creating a very consistent and pretty space in which to create. Pro Audio uses standard Windows dialogues and sliders, drop-down boxes, and text-labeled buttons. This approach is probably better in terms of RAM usage and system performance, but not quite as comfortable and inspirational. I give PA8's interface the highest marks for speed and efficiency when working, and it takes a small hit when it comes to aesthetics.

AUDIO. The first project for which I used Pro Audio 8 involved stacking up 16 voice tracks to create a virtual choir from a single person's voice (a buddy was making a Christmas caroling CD for friends and family). I wanted to see how Pro Audio 8 performed as a straightforward digital audio workstation, because the product's audio features get most of the advertising and promotional attention. While I'm not scrapping my Pro Tools rig just yet, PA8 has sure come a long way since its days as a DOS-based MIDI sequencer, and it packs a lot of power.

Pro Audio 8 offers a rich complement of audio recording and editing features, and integrates well with stand-alone audio editors such as Sound Forge and Cool Edit (both of which, already installed on my machine, were automatically added to the Tools menu when I installed Pro Audio 8). Pro Audio 8 provides tools to record audio directly into a sequence (including manual or automatic punch-in and punch-out), to import

Andrew Boyd is Sound Design Manager at Stormfront Studios in San Rafael, Calif. Drop him a line at aboyd@stormfront.com.

previously recorded audio files, to change gain and apply fades, to split audio files, and to render effects. It can automatically configure itself to your particular setup, but it also gives you a very high degree of fine-tuning to eke the most performance from your system. This tuning lets you adjust the inherent tradeoffs between, for instance, latency and polyphony to suit your taste and working style.

Pro Audio 8 supports DirectX-compatible plug-ins, which can either be used in real time in a mix or while you render to an audio file. Pro Audio 8 ships with a collection of standard effects, including reverb, chorus, flange, delay, pitch shifter, modulate time, and parametric equalizer. Each effect is fully editable, has several presets available from a drop-down menu in the effects window, and lets you save your settings as new presets.

The equalizer and delay effects work well and sound fine, although their interfaces aren't the most elegant I've seen. Unfortunately, the other effects (chorus and flange in particular) fall a bit short. The reverb is actually quite nice at short decay settings (the Club preset is very realistic and usable), but at longer, more dramatic settings, it starts to sound harsh and artificial. In Pro Audio 8's defense, it is quite processor efficient, exacting a surprisingly small performance toll at mix time. And in a mix with primary reverb being provided by a high-quality outboard box (or a good third-party plug-in), it's quite satisfactory. But I feel that the included effects aren't good enough for professional game developers to rely upon to polish an important mix; they'll disappoint you in the end.

MIDI. Cakewalk products have long included a comprehensive set of MIDI recording and editing tools, and Pro Audio 8 is no exception. In fact, I dare say that it is a complete set of MIDI tools; you should be able to do anything you want to do, no matter how unique or esoteric you might think it is. The reason is that in addition to just about every tool you might need in the normal course of a project, Pro Audio 8 provides its own programming language. The Cakewalk Application Language (CAL) is beyond the scope of this review, but suffice it to say, if you like to tweak settings, you'll have plenty to keep you busy. The fact is that there's just not a lot to say about Pro Audio 8's MIDI features — it is the very definition of a full-featured sequencer.

One new feature that deserves mention is the addition of real-time MIDI effects to the mixing palette. Such effects have been around for a generation or two in other products, and they're very useful. For instance, when working on quantize filter settings, there is just no substitute to doing it *in situ*, while all the tracks are playing together, so the effect on the feel of the piece is clear. The effect can then be written to the track, or just used at playback like an audio effect. Other MIDI effects include arppegiator, event filter, and transpose. These effects work great, are easy to use, and are fine additions to the product.

SCORING VIDEO. The ability to open a video file into a project, work on audio and MIDI locked directly to the picture, and then write out the results to a new video file with no generational loss to the video is key to post-production work, especially in games. And it's a feature that's been present in Macintosh-based digital audio sequencers for a while. In PA8, the feature works as advertised, much as it does in other products. I also appreciate the fact that it can work with .AVI, Quicktime, and MPEG video formats. I had some trouble randomly accessing segments within .AVI files that used certain codecs (probably because of the limited keyframing that the codecs provide), but this is certainly not the fault of Pro Audio 8, rather an unfortunate side effect of certain codecs. In any case, this video functionality alone makes Pro Audio 8 a serious contender for anyone scoring game cutscenes.

WORKING MUSICIAN SEEKS MATURE AND STABLE SOFTWARE... Pro Audio 8's feature set is clearly indicative of a product that's been around the block a few times, culled and refined to create a versatile and comprehensive toolkit for the recording musician. And certainly one would expect that a product in version 8 would be as solid and stable as its features are varied and powerful. However, I'm not quite sure this is the case with Pro Audio 8. While I didn't find any bugs per se, twice during fairly routine operations the program crashed on me, and during general use I noticed a number of small glitches

Cakewalk Pro Audio 8 can effortlessly combine MIDI, digital audio, and video in a project.

and strange behavior that made me a little less than confident in the application's overall stability. While I'm not nearly ready to indict the program for being unreliable (to be fair, I was really pounding on it), I do want to share my experience and raise a cautionary flag.

The first crash occurred while I mixed down the caroling piece. As I added effects and mix automation, the large project became too much for my admittedly under-powered system to play back in real time. This deficiency wasn't a huge problem, as I was almost finished with the mix anyway, and Pro Audio 8 has a great Mixdown Audio feature that can mix a large project off line, even if the program can't play it all back at once. I adjusted some final mix settings, selected an empty track pair, and mixed the project down. Up to this point, PA8 worked like a charm. Next, I soloed the newly created stereo mix and listened to it — it was pretty good, so I invoked the Export Audio command to save the project as a stereo wave file so that I could play with it in Sound Forge. My plan was to export the wave, save my project, and exit Pro Audio 8. At that point however, PA8 crashed. According to the error message box, PA8 "performed an illegal operation." Afterward, I had to reboot my system. All my mix settings were gone. Luckily, this was just a fun project for a friend, because if I'd had a client standing behind me, I would have looked pretty bad. The second crash happened while I applied an audio effect to a track. I wasn't able able to reproduce the bugs, so I didn't have a case to present to Twelve Tone's technical support staff. Nonetheless, the episode did nothing to improve my confidence in the product's overall stability.

Because I was unable to reproduce

either crash, I'm not going to put the blame squarely on Pro Audio 8's shoulders. It is possible that other instabilities had crept into the system. Rebooting seemed to solve the problem each time. We all know that Windows is notorious for this sort of thing, and I was working quickly on a project that clearly strained the limits of my hardware. But my system is otherwise stable, a crash is a crash, reboots are very time consuming, and at the very least, my confidence waned. And it was not buoyed by the other little glitches I found, such as windows, toolbars, and buttons not redrawing completely when I switched between views.

DOCUMENTATION. As has been the case with previous editions of this product, Pro Audio 8's documentation is outstanding. The hefty printed User's Guide is absolutely thorough, easy to read, well illustrated, and starts with three informative tutorials that should get even the most bewildered beginner up and running with little trouble. In addition, the online help is simply fantastic — well indexed and cross-referenced and comprehensive beyond belief. Would that all software was so elegantly and effectively documented.

All in all, this is an impressive piece of software. It's powerful, flexible, easy to use, and its price is surprisingly low. I can unconditionally recommend it to the hobbyist/amateur/semi-pro. It has every feature you're ever likely to need, and you'll be up to speed with it in no time. For a professional environment, I'd have to raise a cautionary flag about stability, and point out that you will definitely have to spend some additional money on a third-party effects package for mixing. With those caveats, though, the program certainly is powerful, flexible, and efficient. ■

Acknowledgements

Thanks to Phil King for the kind use of his voice.

Cakewalk Pro Audio 8: ★★★½

Company: Twelve Tone Systems Inc.
Cambridge, Mass.
(617) 441-7870
http://www.cakewalk.com

Price: Standard Edition $319; Deluxe Edition, $429; Upgrades $19 - 409

System Requirements: Pentium 100 or higher (Pentium 120 recommended for NT implementation and real-time effects); 16MB RAM (32MB for NT); Windows 95/98/NT-compatible sound card and/or MIDI interface (sound card required for digital audio record/playback).

Pros:

1. Just about every feature you could want in a digital audio sequencer, well implemented
2. Incredible price/performance ratio
3. Outstanding documentation

Cons:

1. Stability might be questionable in a professional environment
2. Some included audio effects leave a bit to be desired
3. The look and feel of the interface is somewhat unpolished

Collision Response: Bouncy, Trouncy, Fun

I was all set this month to start talking about how to handle collision response. It seemed to be the next logical step from the discussion last month on methods for detecting collisions between 3D objects ("When Two Hearts Collide," February 1997). I thought I could just have these objects that you could move around, make collide, and then watch their responses. Yeah, collision response, that will be great! Then I thought, "How am I going to get these objects flying around in the first place?" Well, I could give each object an initial velocity and they would collide. But, I would need world boundaries for those objects to bounce off of so they would stay in play. To direct the objects, I need to be able to apply force. Suddenly, instead of a nice collision demo, I had designed ASTEROIDS. All I wanted was a little demonstration of a fairly simple concept and instead I ended up applying forces and acceleration to particles. I had stumbled on the big "D" word: Dynamics.

That's alright. I will not be afraid. I always say, "Turn a problem into an opportunity." For months, I've been considering picking up where Chris Hecker left off his "Behind the Screen" columns back in June 1997. Chris had created a very interesting rigid body dynamics simulation and spearheaded the use of hardcore physics in the game development community. However, physics is a huge field full of fertile topics that can be distilled into nice column-sized pieces. So once more good friends, into the breach.

What's So Dynamic About It?

When I was writing about inverse kinematics back in September, I was only really interested in kinematics: that is, the study of motion without regard to the forces that cause it. Dynamics, I said, concerns how forces are used to create motion, and I didn't want to open up that can of worms. Well, the can is now open and the worms are climbing all over.

I'm going to have to recap a bit, but I suggest you go back and reread Chris's column from the January 1997 *Game Developer*, "Physics, the Next Frontier." If you don't have the magazine handy, the article is available on the Definition Six web site at http://www.d6.com/users/checker.

This month, I'm going to focus on particle dynamics. What is particularly important about particle dynamics is the relationship between force, f, the mass of a particle, m, and the acceleration of that particle, a. This can be stated in the familiar Newtonian notation as $f = ma$. You may recall from Chris's column that the acceleration of a particle is the derivative with respect to time of the velocity of that particle, v. Likewise, the velocity of the particle is the derivative with respect to time of the position of the particle, x. You can see how this relationship works in Eq. 1.

$$f = ma$$
$$a = \frac{dv}{dt} = \dot{v} = \ddot{x} = \frac{f}{m}$$
$$v = \frac{dx}{dt} = \dot{x}$$

(Eq. 1)

So, let me state the problem I'm trying to solve. Given a set of forces acting on a particle at time t, where will that particle be after a small amount of time has passed? It's clear that with the value of the force and the mass of the particle, I can obtain the acceleration of the particle. If I integrate that acceleration with respect to t, I'll end up with the new velocity of the particle. If I integrate again, I get the new position. Easy, right?

The structure for a particle is in Listing 1. It's easier to store $1/m$ for the particle because this is what I need in the equations. The forces that act on the particle accumulate in the f term. With this information, I can integrate the dynamic system forward in time to establish a new position for the particle. This process involves solving ordinary differential equations. Fortunately,

LISTING 1. *The particle type.*

```
// TYPE FOR A PHYSICAL PARTICLE IN THE SYSTEM
struct tParticle
{
        tVector pos;            // Position of Particle
        tVector v;              // Velocity of Particle
        tVector f;              // Total Force Acting on Particle
        float   oneOverM;       // 1 / Mass of Particle
};
```

Many have told Jeff that his top is made of the rubber and bottom of the spring. Bounce him and Darwin 3D a note at jeffl@darwin3d.com

LISTING 2. *My simple Euler intergrator.*

```
////////////////////////////////////////////////////////////////////////////
// Function:  Integrate
// Purpose: Calculate new Positions and Velocities given a deltatime
// Arguments: DeltaTime that has passed since last iteration
// NoteS:   This integrator uses Euler´s method
////////////////////////////////////////////////////////////////////////////
void CPhysEnv::Integrate( float DeltaTime)
{
/// Local Variables ///////////////////////////////////////////////////////
  int loop;
  tParticle *source,*target;
////////////////////////////////////////////////////////////////////////////
  source = m_CurrentSys; // CURRENT STATE OF PARTICLE
  target = m_TargetSys;  // WHERE I AM GOING TO STORE THE NEW STATE
  for (loop = 0; loop < m_ParticleCnt; loop++)
  {
    // DETERMINE THE NEW VELOCITY FOR THE PARTICLE
    target->v.x = source->v.x + (DeltaTime * source->f.x * source->oneOverM);
    target->v.y = source->v.y + (DeltaTime * source->f.y * source->oneOverM);
    target->v.z = source->v.z + (DeltaTime * source->f.z * source->oneOverM);

    // SET THE NEW POSITION
    target->pos.x = source->pos.x + (DeltaTime * source->v.x);
    target->pos.y = source->pos.y + (DeltaTime * source->v.y);
    target->pos.z = source->pos.z + (DeltaTime * source->v.z);

    source++;
    target++;
  }
}
```


Chris's column described a numerical method of solving these problems. Listing 2 contains code that uses the simplest numerical integrator, known as Euler's method, to compute the new state of the system. The great thing about this integrator is that it's simple to implement and understand. However, because it's a simple approximation, it's subject to numerical instability, as we will see later.

You Can't Force Me to Move, Can You?

I now have a method for dynamically moving particles around in a realistic fashion. However, to get anything interesting to happen, I need to get things moving. This requires the application of some brute force, or several forces. But what kinds of forces do I want to apply to my little particles?

Well, the obvious force that has been applied to objects in games since the beginning of computer simulations is gravity. When I wrote the article on particle systems back in July 1998 ("The Ocean Spray in Your Face"), I had a very simple system for applying a force such as gravity. This time, however, I want to be a bit more physically realistic. Gravity is a constant force that is being applied to all particles. In order to realistically simulate gravity, force must be added into the particle's force accumulator every system update. In general, this force is a vector pointing down along the y axis. However, there's nothing to stop a simulator from having a gravity vector that points in a different direction. In fact, one of the very cool things about having a good physical simulation is that gravity can change and things will still "look" correct. This realistic look may not occur if you are trying to hand animate an object.

Putting the Bounce Back in my Bungie

Now, gravity was a pretty obvious force to apply to particles. But what else can I do? A loose connection of points isn't really all that interesting to watch even if it is simulated with accurate physics. It would be much more entertaining if I could connect those particles to form structures.

What about stretching a spring between two particles? This procedure is actually easy to implement. Hook's spring law (Eq. 2) is a pretty good way of representing the forces that a spring exerts on two points.

$$f_a = -k_s\left(|L| - R\right) + k_d \frac{\dot{L} \bullet L}{|L|}\frac{L}{|L|}$$
$$f_b = -f_a$$
$$L = a - b$$
$$\dot{L} = v_a - v_b$$

(Eq. 2)

This formula represents the force applied to particles *a* and *b*; the distance between these particles, *L*; the rest length of the spring, *r*; the spring constant or "stiffness", k_s; the damping constant, k_d; and the velocity of the particles, *v*. The damping term in the equation is needed in order to sim-

LISTING 3. *A damped spring force.*

```
p1 = &system[spring->p1];
p2 = &system[spring->p2];
VectorDifference(&p1->pos,&p2->pos,&deltaP);          // Vector distance
dist = VectorLength(&deltaP);
// Magnitude of deltaP

Hterm = (dist - spring->restLen) * spring->Ks;        // Ks * (dist - rest)

VectorDifference(&p1->v,&p2->v,&deltaV);              // Delta Velocity Vector
Dterm = (DotProduct(&deltaV,&deltaP) * spring->Kd) / dist;  // Damping Term

ScaleVector(&deltaP,1.0f / dist, &springForce);       // Normalize Distance Vector
ScaleVector(&springForce,-(Hterm + Dterm),&springForce);  // Calc Force
VectorSum(&p1->f,&springForce,&p1->f);                    // Apply to
Particle 1
```

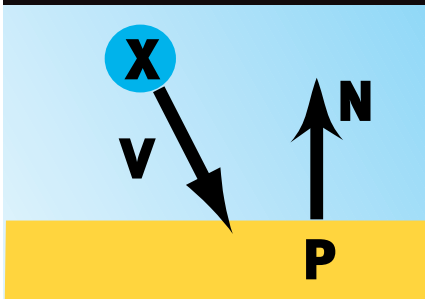


FIGURE 1. *A particle colliding with a plane.*

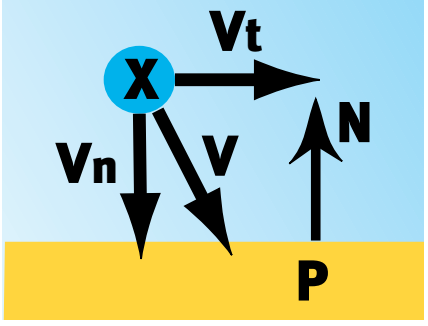


FIGURE 2. *Components of a collision.*

ulate the natural damping that would occur due to the forces of friction. This force, called viscous damping, is the friction force exerted on a system that is directly proportional and opposite to the velocity of the moving mass. In practice, the damping term lends stability to the action of the spring. The code applying the spring force on two particles is in Listing 3.

Other Forces

Viscous drag should be applied to the entire system. A drag is a great way of making the particles look as though they are floating around in oil. It also adds numerical stability to the system, meaning that the particles won't bounce around too much. A viscous drag force is applied by multiplying a damping constant, K_d, with the velocity of the particle and subtracting that force from the accumulator.

Momentary forces are also very useful for interacting with the simulation. I've used a spring tied to a particle and attached the mouse to drag the object around. A force applied to a particle can be used to create a motor or other source of motion.

You can also make some interesting effects by locking a particle. That is, by turning off the simulation for a particular particle, it becomes fixed and can act as an anchor point. (You can achieve the same effect by causing the particle to have an infinite mass. In the simulator, simply set the particle's mass to zero.) Immobilizing one particle like this creates many possibilities for creating complex simulations.

Finally, Back to Collision

Whew, now that I have a nice dynamic particle simulator, I can start talking about collision detection and response again. The simplest form of collision detection that I can add to this simulation is point-to-plane collision. With particles, it will be easy. Last month, I discussed the use of the dot product to determine whether a point has collided with a plane. Take a look at Figure 1.

Particle X with a velocity vector V is moving towards plane P with a normal N. I know that a collision of some sort occurred if $(X\text{-}P) \bullet N < \varepsilon$, where ε is some small threshold near zero. If that value is $< -\varepsilon$, then the particle has passed through the wall, penetrating it. That won't make my simulator happy, so if a particle is penetrating any boundary, it's necessary to back up the simulator a little and try again. If the dot product is just very near zero, then I have what is called a contact and I need to check further.

A particle in contact with a boundary may not be colliding with that boundary if the particle is moving away from the boundary. The relative velocity of the two bodies is checked by calculating $N \bullet V$. If that value is less than zero, the two bodies are in colliding contact and I need to resolve the collision.

To resolve the collision, I need to calculate two more vectors. They represent the motion parallel and tangential to the normal of collision. Take a look at Figure 2.

The normal of collision is simply the normal to the plane. I calculate the velocity after the collision with Eq. 3.

$$V_n = (N \bullet V)N$$
$$V_t = V - V_n$$
$$V' = V_t - K_r V_n \qquad \text{(Eq.3)}$$

In this equation, K_r is the coefficient of restitution. This is the amount of the normal force, V_n, that is applied to the resulting force. If K_r is 1, I have a totally elastic collision. If it is 0, the particle sticks to the plane.

Building with Sticks

Now that I have this nifty particle simulator where I can attach particles with springs and apply forces to them, it's time to build something. Let me start with a simple block such as the one in Figure 3.

Each of the edges of the object is a spring connecting the vertices. Unfortunately, if I run this object through the simulator, I end up with a big heaping mess. The mess occurs

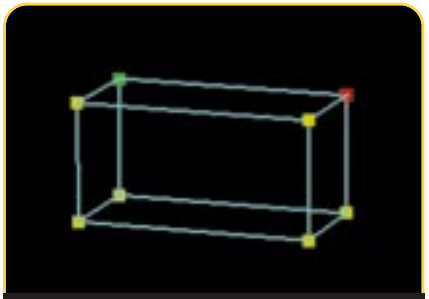


FIGURE 3. *A simple dynamic cube.*

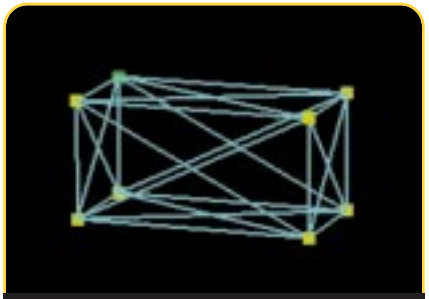


FIGURE 4. *A stable cube.*

FIGURE 5. *A cube out of control.*

because the springs connecting the vertices aren't enough to provide stability for the cube. In order to create a cube that won't collapse, it's necessary to put crossbeam supports on each face of the cube (Figure 4).

Creating objects this way feels more like constructing a bridge than 3D modeling. You find yourself adding struts and crossbeams all over the place.

FOR FURTHER INFO

- Baraff, David, and Andrew Witkin. "Physically Based Modeling," SIGGRAPH Course Notes, July, 1998, pp. B1-C12. I built my first particle dynamics simulator after seeing an article by David Barraff a couple of years ago. For this article, I used one source of his and Andrew Witkin's in particular.
- Hecker, Chris. "Behind the Screen." *Game Developer*, October 1996 – June 1997. Credit for the ideas and some of the methods of simulation go to Chris Hecker. I have tried to base my code on many of his ideas so it will be familiar to readers. His excellent series of articles on rigid body physics got me and many others excited about real-time physics. Hopefully, I can continue to build on this tradition. Also available on Chris's web site at http://www.d6.com.

You will need several good math and physics books if you really want to get into this topic. Here are a few that I used in this article.

- Beer and Johnston. *Vector Mechanics for Engineers: Dynamics*, Sixth Edition, WCB/McGraw-Hill, New York, 1997.
- Mullges and Uhlig. *Numerical Algorithms with C*, Springer-Verlag, New York, 1996
- Acton, Forman S. *Numerical Methods that Work*, Harper and Row, New York, 1970. This last book was a useful little book my father had from his days of working on guidance systems. Now I am using it to make virtua-jello. Go figure.
- Doug DeCarlo at the University of Pennsylvania wrote an application for X-Windows called XSpringies that allows you to simulate 2D particle-spring interactions. You can check this out from his website at http://www.cis.upenn.edu/~dmd/doug.html or get the program at ftp.cis.upenn.edu/pub/dmd/xspringies/xspringies-1.12.tar.Z

Leave a face open and it behaves correctly. The face without the crossbeam supports is more likely to collapse.

Bring Me Stability or Bring My Program Death

I mentioned before that by using a simple Euler integrator, I'm sacrificing numerical stability for ease and speed of calculation. You may wonder, however, what happens when the system becomes unstable. There's a really easy way to find out what will happen. Remember the spring coefficient that was applied to the particles? This coefficient represented the stiffness of the springs used. If I set that value fairly high because I want really stiff springs, the little Euler integrator cannot handle it. If you run that cube I had with stiff springs, you may see something like Figure 4 or something equally interesting. The still frame doesn't do it justice. This is a rigid body way out of control.

There's a solution to combat this instability beyond, "Don't do that" — it's to give my integrator an upgrade. Euler's method is simply not sophisticated enough to handle problems such as this. Next month, I will take a look at how I can improve the integrator with something a little more stable.

Kid in a Gummi Bear Store

I really find in fun to play with this simulator. It's very satisfying to bring in shapes and play with making them stable and tweaking the spring and gravity settings. You then can fling the objects all around and bounce them off the walls. There are many more variables that can be added to the simulator. Other forces such as contact friction can be added. Some interactive features such as pinning vertices would make it more fun. But I think we're on our way to a really fantastic Jello-land simulator. Next month, I also plan on adding support for multiple bodies as well as object-to-object collision. Check out the source code and demo application on the *Game Developer* web site at http://www.gdmag.com. It will allow you to load in your own shapes, connect them with springs, and play around with the simulator. ■

And Now for Something Completely Different...

When was the last time you sat down in front of a game and said to yourself, "Wow, this game really looks cool; I wonder how they did that." You can probably count on one hand the number of games identifiable by just a half-second glance at the screen.

Unfortunately, the number of really good-looking titles seems to have tapered off in the last year or two (With the notable exceptions of the ABE'S ODDYSSEY and RESIDENT EVIL franchises, and the recently released HALF-LIFE). In recent months, all too many hotly anticipated games have turned out to be all hype and no substance — just the same old mediocre stuff.

Far out in the deserts of the great Southwest, a small team of innovators is laboring to bring us a game with an extraordinary look. In an industry bloated with QUAKE-clones and MYST look-alikes, they are quietly working to bring us something refreshing and unique — or, in the words of Monty Python's John Cleese, "something completely different." In this month's issue, we'll be taking a look behind the scenes and discovering exactly how they did it. The title is FLESH & WIRE, and the developer is Running With Scissors (RWS).

FLESH & WIRE: The Challenge

According to Randy Briley, the soft-spoken art lead for the project, the development process for FLESH & WIRE (FW) has always been a little bit different. For starters, the publisher (Ripcord Games) has been very hands-off, letting the development team drive the development. This uncharacteristic display of trust has as much to do with RWS's track record of getting products out the door on time as it does with Ripcord Games' relative newness to the gaming scene. And although the style of game play has some basis in currently released titles (the game is something of a cross between RESIDENT EVIL and THE THUNDERCATS), the look of the game is anything but conventional. From character design and animation to background generation, the unorthodox look derives from equally unorthodox production methods.

FIGURE 1. *Concept Sketches of Some of the characters in FLESH & WIRE.*

Mel has worked in the games industry for several years, with past experience at Eidos and Zombie. Currently, he is working as the art lead on DRAKAN (http://www.surreal.com). Mel can be reached via e-mail at mel@surreal.com.

When RWS finally settled on the game spec, they realized that from a resource production standpoint, they had bitten off more than they could chew. In addition to the standard budget of special effects, GUI art, and several minutes of cut scenes, the spec called for over 200 static screens of game play with in-betweens, and a set of enemy and player characters' 300+ unique animation sequences. With a production cycle of just under 18 months, no budget for outsourcing, and an extremely small art team, the task seemed pretty daunting. It was time to improvise.

After analyzing the production workload, the team determined that the two main liabilities were character animation and static background generation. In both cases, the time and manpower were projected to be too limiting. A faster, efficient workaround was needed for both.

Character Animation

The character designs for the game were anything but conventional. As the game's title implies, the characters seem to be escapees from the Borg infestation of H.R. Giger's zoo. Because the character designs touted a tricky combination of rigid and flexible joint structures, the team devised solid-skin skeletal system using Softimage. The straightforward solution of animating with weighted envelopes enabled the animator to maintain the rigid inflexibility of steel while keeping the organic parts of the body fluid and supple.

To get the characters animated in a timely manner and within budget, however, proved to be a more difficult task. There were over 300 unique animations in the spec, and these had to be completed in a matter of a few months — in time for design and programming to start implementing them.

The team had three options: motion capture, pure hand-animation, or rotoscoping. After looking at the animation lists, the team decided that straight motion capture would be too expensive and too limiting for this specific character set. Although RWS's experience with motion capture was positive (they are currently producing a different title using motion capture services provided through House of Moves), the unique body style and the rigid nature of motion capture data precluded its

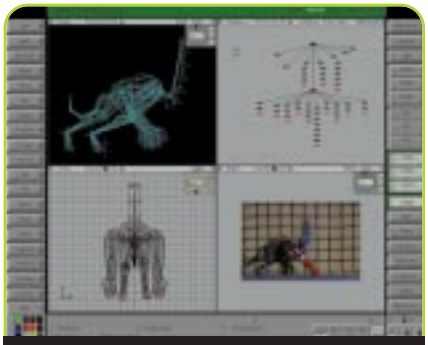


FIGURE 2. *Rotoscoping in Softimage.*

FIGURE 3. *In-game screenshot using miniatures.*

FIGURE 4. *Artist hard at work on the production floor.*

Who the Heck is Running With Scissors?

RWS is a well-established development house located out in Tuscon, Ariz. (that's right, Tuscon, as in cactus and gila monsters). Although the guys at RWS have quietly (but successfully) been making children's games for over a decade, if you recognize the name it's probably because these are the same folks who brought us POSTAL, the controversial and oddly satisfying game that was pulled from the shelves soon after it was released due to the "over-the-top" gratuitous violence.
http://www.gopostal.com

use for this specific task.

Additionally, the amount of time it would take a single animator to provide all 300 animations proved to be too limiting. In contrast, rotoscoping could be done largely in-house with little or no overhead, the production time compared to hand animation was much faster, and although it required the talents of a skilled animator to implement, it provided a cheap, efficient method to complete the animations on schedule.

The team went down to a local gymnasium and interviewed several martial arts students. Then, working closely with the art lead (a martial arts expert himself), the actors were mocked up to look like the characters in the game. Several sets of motion shots were taken, using two synchronized digital cameras set 90 degrees apart (front and side). After digitizing these images and importing them into Softimage, the result was a sequence of images. The animator then animated the characters by hand, using the images as a guide. Figure 2 shows an example of the process being done in Softimage.

The result was a rapidly-created set of fluid, realistic character animations, generated by hand, but with the aid of a human actor. In comparing pure hand animation to rotoscoping, the team surmised the following: 1) rotoscoping took only about a third of the time vs. generating the animations by hand and 2) although the same limitation existed for rotoscoping and motion capture (in that they were required to generate an animation list far in advance of actually producing the animations), the flexible nature of the process allowed the animator a large degree of creative license, so that last minute changes to the actual animation sequences were relatively easy to implement.

Background Generation

Compared to the mammoth task of generating over 200 hundred in-game background scenes, the character animation problem looked simple. With only a handful of 3D artists on staff, the team had to make some tough decisions.

As the project evolved through its initial stages, it became clear that the art direction was evolving towards the techno-grunge look typified by such industry standards as *The Crow* and *City of Lost Children*. The level of detail the team wanted would require hours of tedious texture and modeling work using classical CG methods. Given the size of the team and the allotted time, this simply would not be possible. Rather than cut the design or ask for more time, the team resolved to find a solution that would allow them to maintain the scope of the project while holding true to the artistic vision. They took a gamble, and decided to build the entire game using miniatures.

Near the end of the planning phase of the project, RWS presented the publisher with a proof of concept for the process. For the first test, the team put together a town from a model railroad set and digitized it into the POSTAL engine. In short, the result was a huge success. The models had a photorealistic quality that pure CG couldn't attain, and the artifacts and subtle inconsistencies in the model were filtered out as a result of the digitizing process. As you can see in this later screen shot in Figure 3, the result looks like a set from a Hollywood special effects studio.

The Process

Put simply, the sets for the game were built with "anything we could get our hands on," says Randy Briley. Basically, the team would just bring stuff in: PVC piping, copper tubing, old VCR's, and so on, and the

FIGURE 6. *The digitizing room.*

FIGURE 5. *One of the larger models. This one was used for in-game cinematics.*

pieces were glued together and painted using a hot glue gun and standard modeling paints. Most of the backdrops for the game were created using styrofoam panels, which proved easy to get hold of and standardize. "Once we got an assembly line going with a certain panel (background piece), we could crank each one out in a matter of a few hours."

Early on in the production cycle, it was determined that because the most time-prohibitive factor was the sheer number of pieces required for the background sets, the production would begin concurrently with level design and engine creation (the engine for the game was developed as part of the production process). As with all production lines, any reworking of resources would prove highly detrimental. Once objects were created, they needed to be left intact. Otherwise a realistic production schedule could not be maintained.

To enable the designers to have the ability to interactively tweak the levels, the sets were designed to be assembled from modular pieces. This single key factor is what allowed the art team to begin production on final in-game architecture at a very early point in the production cycle, thus enabling design and programming to have the requisite time for implementation and tweaking.

Figure 6 shows the setup for taking the actual video footage. A standard Cannon XL1 digital video camera, capable of 60+ FPS at NTSC resolution (720×486) was used to capture the images. Spotlights with colored gel filters provided the lighting. Note the smoke machine in the bottom right of the shot. During filming, smoke was turned on to give the captured images depth, and to soften the edges of the materials used in the shots. Also, volumetric lighting effects were used extensively, adding needed realism.

Figures 7 and 8 show the results once the images have been processed and filtered into the engine. Note the different lighting schemes that have been applied. Because the engine provided for real-time lighting on the polygonal characters, it was important to match up the lighting values for the scenes. This turned out to be one of the most difficult aspects of the process because if the scenes were lit incorrectly, they had to be retaken from scratch. However, the different lighting schemes also enabled the team to reuse the same modular pieces over and over again. The abstract nature of the design was such that the same piece looked correct from several different angles, and in fact, this efficient re-use of resources was key to the success of the process.

Another difficult problem was setting up the exact camera angles in the game to match the angle from which the shot was taken. Because, in the engine, the scene was basically created using a set of flat planes and rudimentary polygons, the entire setup could be shifted to give the correct perspective and depth of field to the on-screen actors.

As it turned out, the process went much more smoothly than anticipated. For example, the large piece of geometry in Figure 6 took about two weeks to get completely finished and into the engine. And because it was built with a modular design and an abstract construction, it was usable again and again from different angles and with different lighting schemes. Since then, the team has been able to increase its projected number of screens from 200 to over 300, and still maintain the scheduled production cycle.

By far however, the biggest advantage of the process is the lack of any requirement for CG expertise on the part of the artists. Consider that with a single trained 3D artist to guide the process, the bulk of the artists can be classically trained with little or no industry expertise. This means that production costs go down for any given piece of work,

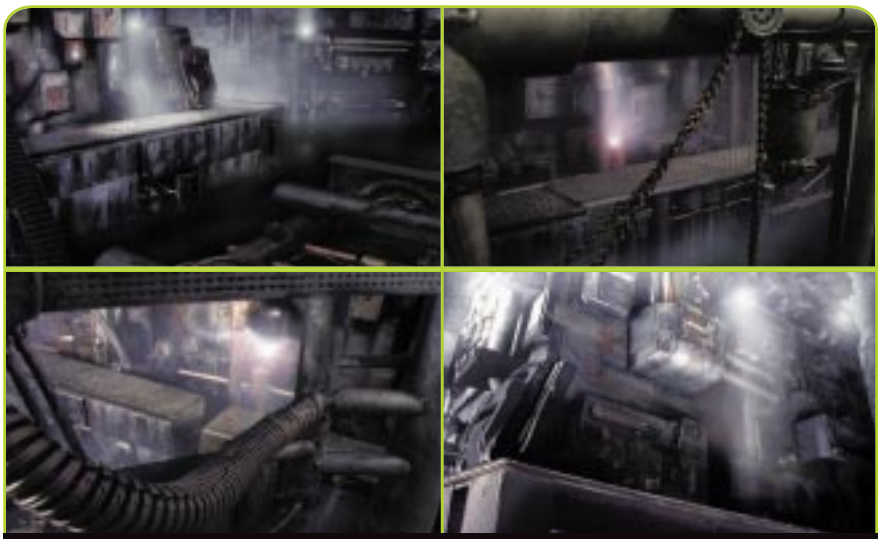
FIGURE 11. *One of the final images in the game.*

or, you get a lot more resource for a lot less money. Again, an efficient, innovative method for solving an otherwise unsolvable problem.

Necessity is the mother of innovation, and it stands to reason that the success of the process is due, at least in some degree to the relative isolation of the RWS team. The fact that the developers are insulated from the mainstream has had mostly positive consequences. The team has been obliged to re-invent the wheel in several instances, and in so doing, they have progressed down a slightly different path than the rest of us. And although the project is far from complete, it's easy to see that if it stays on track, the result will be nothing less than spectacular. ■

SPECIAL THANKS:

Randy Briley, David DeGasperis, Vince Desiderio, Mike Reidel, Amy Searcy, and the rest of the FLESH & WIRE team.



FIGURES 7, 8, 9, AND 10. *Models digitized into the engine.*

DVD — A/V Feast and Famine

DVD, the digital versatile disc, is going to pass the halfway mark in 1999 to becoming the replacement for CD-ROM drives in PCs. While there continues to be some uncertainty over copyright issues and storage formats, and while there is even more uncertainty about how the DVD-ROM market will shape up, it is still safe to say that in the next two years we can kiss 600MB of storage goodbye, and move on to multi-gigabytes of DVD-ROM storage.

Most developers acknowledge grudgingly that this extra storage could force them to provide more video content than they feel comfortable with or find necessary. However, it could also help to increase the general quality of game graphics via better and larger textures. A little less clear is the impact on audio development, and particularly the use of emerging 3D audio features. Nevertheless, DVD drives are going to be out in force in the coming year irrespective of whether game developers need them or not, and undoubtedly, all that extra storage space will get filled up.

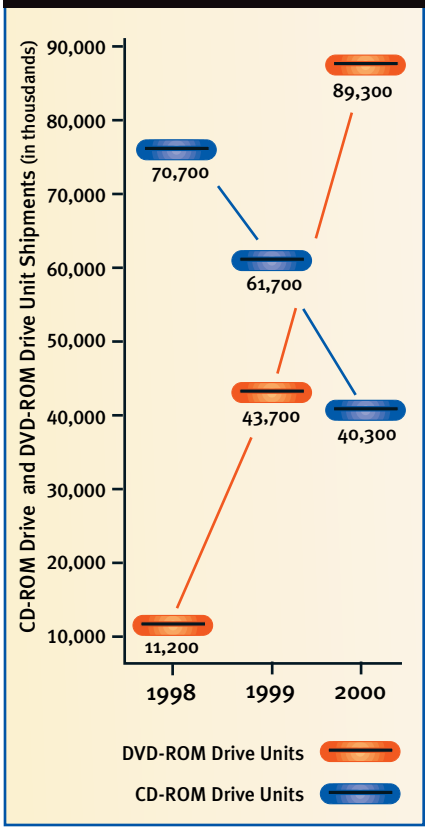
DVD Market Growth

There seems to be little agonizing from PC companies as they transition from CD-ROM drives to DVD. This is mostly due to the fact that CD-ROM is very well established, and DVD drives in PCs provide backward compatible with legacy CD disks. More importantly, the transition is also aided by the fact that consumers are hungry for greater content and more high-bandwidth multimedia. CD-ROMs established themselves on the basis of the oversell of multimedia CD titles in the early 1990s. Most of the multimedia CD-ROM market has disappeared to be replaced by either games or edutainment category products. In the case of DVD, consumer expectations of multimedia content are difficult to gauge. It may be that DVD video in the home will force developers to pay more attention to the quality of audio and video on DVD-ROM on a PC, or even in a console at some point. One thing is for certain: in the North American market at least, it is unlikely that DVD movies will be a driving force in DVD-ROM content on the PC. The computer industry went down that route with VideoCD, basically an MPEG video on CD-ROM. And despite its popularity in some areas of Asia, the VideoCD is primarily used to distribute adult video content in North America and Europe.

And there's more storage to come beyond existing DVD-ROM capacity. As storage needs increase beyond the year 2000, we will start to see media that go beyond DVD-ROM's 4.7GB of storage space to somewhere in the region of 15GB. The DVD-ROM will grow to be a significant force in the next five years, and software is going to be a big part of the success of DVD. According to Cambridge Associates, the total number of DVDs forecasted to be replicated in 2002 will be 1.1 billion units, of which 350 million will be video and the remaining 760 million will be DVD-ROMs.

More storage is going to put pressure on game developers to use that additional capacity. I still remember how some publishers were taken to task for not using all the storage capabilities of CD-ROMs in the early days. Consumers expected that when they bought a CD, it needed to be full of stuff, irrespective of whether 50MB of content fitted nicely on one CD platter rather than a box full of floppy disks. There was 600MB more left unexploited. I would still like to think that more video content will be anathema to the development community, and that developers place some bigger expectations on audio.

FIGURE 3. *Worldwide PC CD-ROM and DVD-ROM drive unit shipments (in thousands). Source: IDC.*



Omid Rahmat works for Doodah Marketing as a copywriter, consultant, tea boy, and sole employee. He also writes regularly on the computer graphics and entertainment markets for online and print publications. Contact him at omid@compuserve.com.

Making Noise with DVD

I may be alone in that sentiment. Jeremy Schwartz, senior analyst at Forrester Research, has this to say about DVD's impact on the audio portion of the game industry: "A lot of people are not thinking about branching video, so there's an opportunity for audio. However, I think most game companies will keep the percentage of the budget they spend on audio the same. Unfortunately, audio is still the orphan child compared to graphics, and usually the cool 3D audio stuff is the first thing to get cut out of a game's budget when the belt is tightened." The key question is then, apart from the quality of 3D graphics, what will DVD bring to the interactive audio experience? No one wants to make branching video games, but no one seems too interested in upping their audio budgets either.

Arguably, one of the of the biggest advancements in the audio market has been positional audio, and Dolby's AC3 specification transcends and bridges both DVD and PC markets. (AC3 is the standard 3D audio format on DVD and is also widely used on the PC.) Standards remain a tricky issue for DVD audio. In fact, the original incarnation of the DVD audio format was slated to be set way back in 1996. Intel and 38 other consumer electronics manufacturers are members of the DVD WG-4 Audio Working Group and support its proposed DVD audio specification. However, the Audio Working Group, like all committees of its kind, suffers from the problem of having to bring together the conflicting agendas of its members to ratify a universal standard. Furthermore, Intel was the first computer industry representative invited to join the forum, and that was only this year. So, DVD audio remains a fledgling idea on the PC.

The DVD Forum finally released the Version 1.0 specifications in October, 1998. It is a very flexible format. It's possible to store more than 74 minutes of audio content for two channels at the highest quality, which translates into a sampling frequency of 192KHz-with 24-bit quantization, or six channels for Dolby Digital 5.1 Surround Sound at a 92KHz sampling frequency with 24-bit quantization. Quantization is at 16, 20, and 24 bits. Currently, CD audio technology is limited to a sampling rate of 44.1KHz and 16 bits of data. The format supports all DVD formats, from 12cm diameter 4.7GB single-sided disks to 17GB dual-layer double-sided 8cm units. In short, the audio quality on DVD surpasses anything you have ever experienced on CD. Tom White of the MIDI Manufacturers Association summarizes the state of DVD audio this way, "There are a lot of flavors of DVD audio, so it is not clear what standards game developers would develop to, and what kind of hardware playback installed base there would be in the PC market to support it." That includes WG-4.

The key question is then, apart from the quality of 3D graphics, what will DVD bring to the interactive audio experience?

Content is a Fat King

While CD-ROM is confined to audio and computer markets for the main part, DVD promises to be on everything from game consoles to PCs to digital television set-top boxes. DVD will nudge game developers even further into the mass-market entertainment industry. This may ultimately, be the true strategic value of moving to DVD.

Game developers have started to slowly test the waters of this new medium. Psygnosis's LANDER is the company's first DVD game and probably the game industry's most widely publicized DVD title. It incorporates MPEG-2 video sequences with Dolby Digital 5.1 channel music as well as in-game audio and interleaving sound effects. LANDER is also optimized to allow game players to play the DVD movies and trailers in the game on a DVD movie player or to just listen to the soundtrack. So, the role of DVD is ultimately coming down to how it plays in the general entertainment market. LANDER is marketed in the same manner that DVD video is marketed to a traditionally VCR-loving public: buy the new format and get quality and new features at no extra expense. Despite the ramifications, game developers are a conservative lot in some areas, and DVD development is not taking the industry by storm. Sonic Solutions of Novato, Calif., has been involved in DVD authoring and production for three years now. The company worked with Hollywood studios to encode MPEG-2 content even before the standards for DVD were ratified. Mark Ely, director of product marketing at Sonic Solutions, has noticed the reluctance of game developers to embrace DVD." We have seen a tremendous lag in support for DVD on the game developers' side," he said.

"One of the reasons may have been the lack of an installed base on the PC. It has only really been in the last couple of months that we have seen DVD game titles, but the tools have been around awhile. Initially, what we are seeing is game developers merging content from multiple CDs onto one DVD or recoding video and audio content for DVD, and quality is the single greatest advantage of DVD over CD-ROM to them."

For companies such as Sonic Solutions, the Hollywood production market is relatively small. Growth will have to come from corporate developers, multimedia developers, and game developers. Perhaps the higher cost of producing a DVD is a factor in its sluggish growth in the game industry. Presently, the cost of encoding and producing DVD audio and video is roughly 50 percent higher than for other media, but that cost is coming down dramatically. Furthermore, the cost of mastering a DVD disk is substantially higher than for CD-ROM. Fortunately, the pattern of development is following the trends in the CD-ROM market, and by 2000 developers can expect DVD production and mastering costs to be the equivalent of CD-ROM. In the meantime, to set up a small DVD audio and video authoring suite can set developers back anywhere from $30,000-$50,000. The big publishers can do it, and high-profile content will benefit.

DVD is as good a convergence technology as anything that has been touted. It can bounce from PC to set-top to home multimedia player, but how it will do that is still a mystery. For now, game developers look at the medium as better and bigger storage, and maybe not much else. It's your call. Good strategy, or bad strategy? ■


Back
25
24A
24
26A
23A
23
27
27A
28
16
16A
17
17A
18

VIDEO IN GAMES: THE STATE OF THE INDUSTRY

BY BEN WAGGONER AND HALSTEAD YORK

Video is one of the most commonly used and least understood elements of modern computer games. Ever since the CD-ROM offered a medium able to carry significant amounts of audio and video, game developers have worked to incorporate video into their titles. Although there have been many missteps along the way, video is still a critical element of the game designer's palette. When used well, it can create mood, set up game play, introduce characters, and forward narrative. When used poorly, it can rip you out of the game faster than a direct rocket-launcher hit during a deathmatch.

Ben Waggoner is the chief technologist of Journeyman Digital, a company specializing in pushing the quality envelope for interactive media assets. As the World's Greatest Compressionist and avid game player, he has an unquenchable need to improve the quality of digital video across the gaming industry.

Halstead York is the head of digital media production for Journeyman Digital. When not producing and consulting, Halstead spreads his digital media gospel through teaching and lectures.

Throughout the brief history of our industry, games have striven to be more immersive, more involving, and more dramatic. The term often used is "cinematic." There was even a time when the term "interactive movie" was very much in vogue. Now of course, we look back at this period knowing that most filmmakers don't know much about games and that game designers don't necessarily make good films. The half-baked efforts of this period stigmatized the term "full-motion video" (FMV). Indeed, we've reached a point where one recent FMV adventure game, A FORK IN THE TAIL, proudly proclaimed that it contained "FMV that doesn't suck." For the record, the FMV didn't suck. However, the game did, and for a very simple reason: game players don't like being passive. Game players like to play, and choosing which sophomoric one liner to use on the scantily clad, direct-to-video harlot in front of you *ain't* playing.

While many developers have been licking the wounds they received after mistaking cutscenes for game play, others have forwarded the state of the art. GRIM FANDANGO's FMV adds camera movement and the pacing of professional editing, while keeping the fabulous feel of its engine intact. In fact, many can't tell where the video ends and the engine begins. Jane's has done a wonderful job with FMV, repurposing training films and using a news broadcast format at the beginning and end of campaigns in the LONGBOW series. Here, video is used to present information in a concise and informative manner while simultaneously creating a technothriller feel in the simulations.

A Brief History of Video in Games

The first game using video was the arcade classic DRAGON'S LAIR. However, this game was based around an analog LaserDisc, not digital video. The first game using digital video in any significant way was SHERLOCK HOLMES: CONSULTING DETECTIVE, released back in 1992. It had a budget of over $1 million and was shot by experienced filmmakers with professional actors on real sets. Given the limited quality of the playback technologies of the day, the quality of the presentation was poor on most platforms, but the game used video well and it was a pretty fun beer-and-pretzels game.

The first big commercial hit in the burgeoning interactive movie genre was Trilobyte's THE 7TH GUEST. In retrospect, the success of this title was probably more due to the novelty of seeing video on a computer, rather than the game's design. Functionally, the game was a series of barely related logic puzzles that used cut scenes as a reward for solving those puzzles. The video was poorly produced; actors were filmed in front of a blue sheet with a home video camera. It was impossible to make a clean key from this, and so the actors were left with ugly blue halos. The game's signature effect of transparent ghosts with blue halos walking through rendered environments wasn't originally intended — rather, it was a last-ditch attempt to make the best use of this artifact. THE 7TH GUEST's success set up some basic fallacies for the genre: quality doesn't matter, video is enough of a reward for users, and game play doesn't have to be tied to the video narrative. Needless to say, this model isn't seen in games anymore, at least in games in which the actors wear clothing.

The standout success of the early digital video era was MYST. While reviled by some game purists, MYST remains the most popular game of all time, and immeasurably contributed to expanding the mass market for interactive entertainment and CD-ROM drives in general. MYST used video in an appropriate manner — in short snippets, inside the overall game environment, advancing plot in a plot-oriented game.

The expectation grew in some quarters that "Interactive Movies" would prove to become the dominant form of computer game, and prove to be another outlet for the talents of second-tier stars. Some of them were even quite interesting, such as PHANTASMAGORIA.

WING COMMANDER III was another turning point in the genre. It melded THE 7TH GUEST's model of video sequences between game play elements with the plot-integration of MYST, and added big name stars such as Mark Hamill and Malcolm McDowell. And it was moderately successful, both creatively and financially. However, the costs for producing what was effectively a TV miniseries in parallel with game development were very high, and the games had to be hugely successful to break even.

The way video was used in WING COMMANDER III remains dominant today, even though our video budgets are a small fraction. We see video in short, plot-advancing sequences that alternate with game play and give context to what happens in game play sequences. Most of the budget and energy goes to an impressive opening sequence that sets up the plot and is endlessly repeated at game retailers as an attract loop.

Son, Put Down That Camera Before Somebody Gets Hurt

A lot has changed since the Hi8-video-and-blue-sheet days of THE 7TH GUEST. Game players today demand a much higher level of creative and technical competence, and developers need to recognize that the production values of their video need to match (and often exceed) the quality of the rest of the game.

Why are the production values of cutscenes and other video elements so critical? Because you're competing against the consumer's instinctive understanding of production values, an education furthered every time they go to the movies or turn on a TV.

Too often, a game's development process starts without much thought for the video other than, "We'll have a cool video intro" or "We'll use cutscenes to introduce each level." All too often, this attitude leads to cinematics that seem incongruous and inappropriate for the game. I remember watching the MECHWARRIOR 2 opening animation, and thinking how great it looked. When I started the first mission, the game's then state-of-the-art 3D engine seemed cheap and flat. Now, I've been playing and working on games for some time, so intellectually, I knew that the MECHWARRIOR engine was damned fine. However, I couldn't escape wondering what it would be like to play that game I saw

in the intro, with large rocky hills to hide behind and cool textures on the Mechs. In short, Activision presented a bill of goods in the intro that they couldn't deliver once the player took control of the action.

Cinematics should either fit in with the look and feel of the game engine, or use the high visual definition afforded by FMV to reveal details that can't be seen within the game. GRIM FANDANGO, BLADE RUNNER, and INTERSTATE '76 offer excellent examples of FMV that looks like the rest of the game, but offers details and subtleties of motion that are still out of reach of a modern real-time 3D engine. They add depth and character to the games without making the player feel that the game play is a weak and underdeveloped version of the cinematics. The COMMAND & CONQUER series and QUAKE II offer excellent examples of games that use FMV to step completely outside of the game mechanics and offer views of the game universe outside of what's

The Black Art of Chroma Key Compositing

If you're combining live action and animated elements within a scene, compositing will play a large role in your final product. You'll most likely begin by shooting video on a blue or green screen. The background color is then removed using a chroma keying process. Blue and green are the most popular colors for a removable background because they are easy to remove and rarely appear in foreground subjects. Chroma key production and post-production is complicated, and it's strongly recommended that you allow professionals to handle it. However, if time, budget, or ego considerations get in the way, here are a few pointers:

The original design spec for JOURNEYMAN PROJECT 3 called for a blue time-travel suit. When developers realized that wouldn't work in a blue-screen shoot, the suit was quickly repainted green.

1. **USE A BIG SPACE.** You'll want to keep your talent as far a way from the screen as you can and light as flatly and broadly as you possible (more on this later). For these reasons, you want your space to offer as much room as possible.

2. **GIVE YOUR BLUE SURFACE A SLIGHT CURVE.** A very slight curve on the vertical axis will help soften the light and lessen visible highlights on the screen.

3. **USE FLAT LIGHTING ON THE SCREEN.** The screen should be lit softly from above and possibly the sides. Never point lights at the screen from behind (or too near) the camera. The goal is to introduce no contrast variations. A video tool called a waveform monitor is used to measure the brightness and color of the screen to make sure it is consistent throughout. Any good production group will use one. If you use any glossy materials on your screen, you're sunk.

4. **WHEN YOU LIGHT THE TALENT, DON'T RELIGHT THE SCREEN.** Dramatic and flattering lighting is often antithetical to good bluescreen lighting. Keep the talent far enough from the screen so that you can light them without their lights hitting the screen. Never point talent lights at the screen — you won't be able to key later. And remember to recheck your waveform monitor after you've lit everything.

5. **CHOOSE A PROFESSIONAL, HIGH-QUALITY TAPE FORMAT.** You'll need a format that can handle highly saturated, complicated images. This requirement precludes the use of consumer formats such as VHS and Hi-8. The new DV format is a better choice, although it deals with color space in a somewhat limited fashion, and can create pixelated artifacts around the edges of foreground objects, making the objects difficult to key. Betacam is a viable low end, with Digital Betacam, Digital-S, and film being the preferred format choices.

6. **NARROW YOUR DEPTH OF FIELD.** You may want the talent in focus, but keep the screen as blurry as possible in order to hide the inevitable imperfections in the screen and make keying much simpler.

7. **KICK THE TALENT (NO, DON'T HURT THEM).** A warm-colored light (try an orange gel) placed behind the talent pointing at his or her back will help create a strong, very unblue edge on the talent, which will keep fingers and hair form disappearing in the key.

8. **AVOID THE KEY COLOR IN ANY FOREGROUND OBJECTS.** Yes this sounds simple, but trust me — it isn't. Blue shirts, socks, even eyes can disappear in the key, as will anything that reflects blue (particularly white and metallics). If you have to use blue elements (such as the WING COMMANDER uniforms), shoot on a green screen.

9. **FIX PROBLEMS IN PREPRODUCTION.** Generally, what you're paying for with high-end post tools is the ability to screw up in production. *Terminator 2* might have cost one third as much if it hadn't been rushed through production so quickly. If you don't have access to high-end tools, following these tips (and others like them) are the only chance you'll have to get a good key. You have time and intelligence, which can offer you almost all of the advantages of money as long as you use them well. Also, I don't recommend putting too much faith in the equipment to hide your mistakes.

10. **IT'S IMPORTANT THAT THE LIVE-ACTION VIDEO MATCHES THE LIGHTING PRESENT IN THE RENDERED BACKGROUND ELEMENTS.** Otherwise, even the cleanest composite will stand out, and the video will look fake and gimmicky. Giving the animator a diagram of your lighting setup and a tape of the shoot are a requisite.

shown during game play. The cinematics in these games let players look at the world from other perspectives, offering images and information that these players can hold in their minds as they play the game. As we watch a unit's health go into the red in C&C, we remember the opening scenes of missiles and bullets ripping through flesh and steel. Wandering through the Strogg's homeworld in QUAKE II, we think from time to time of that dramatic flyover in the game's intro and keep wondering just what that big gun we saw is meant for.

On the other hand, LucasArts' otherwise excellent JEDI KNIGHT: DARK FORCES 2 made poor use of live action and prerendered animation in its cutscenes. Far too often, a cutscene would transition into game play, leaving the player completely disoriented by how different the surroundings and characters looked. Either the cutscenes made me feel that the much lower-quality in-game graphics were a gyp, or poor acting made important, dramatic villains laughable. Anybody remember Jerec shooting his tongue out like a frog in the opening video? Goofy.

Which is not to say that live-action video is the downfall of a game. The WING COMMANDER games, particularly WING COMMANDER: PROPHECY, made excellent use of video to further the plot and add depth to characters. The FMV that bookends most missions achieves its goal: it makes the player feel as though he or she is the lead character in a sweeping B-movie space opera. The deeply underrated SPYCRAFT: THE GREAT GAME made excellent use of video, creating a sense of realism and immediacy that couldn't have been matched with CGI. Similarly, GABRIEL KNIGHT: THE BEAST WITHIN used live-action video brilliantly, and showed just how involving real actors can make a game.

Putting Together the Right Team

For many developers, it has become important to bring in a production company or independent producer at the same time the rest of the asset team is being assembled, particularly if the video will involve a live-action production. However, even if the sequences are 100 percent animated, it's preferable to have the team creating the assets involved as early as possible.

Just as it is important for the developer to appreciate the intricacies of FMV, so to must the FMV team understand the game on which they're working. Make sure that they have a real sense of the look and feel of the rest of the game, particularly in-game elements. If the game uses a palette overwhelmed with blues, then the video should match it. FINAL FANTASY VII's cutscenes have wildly different styles of character animation than the rest of the game. Its video uses a more traditional anime style, while game play centers around super-deformed characters. The super-deformed style is often used in Japanese animation to convey humor and comic relief. However, I'd go out on a limb and say that the dichotomy in the look is disruptive and antithetical to the somber adventure that the game seems to work so hard to present. Apparently, Square agrees — FINAL FANTASY VIII won't use the super-deformed design elements.

Ten Digital Video Game Disasters

1. Bad bluescreen. Do things look funny around the edges of your actress's long hair? Is the background either too visible or not visible enough? That's the product of bad bluescreen. These sorts of effects can be expensively fixed in post with frame-by-frame tweaking, but it's much better to get right in the first place. See The 7th Guest.

2. Interlaced video. In analog video, the even lines are captured 1/60th of a second apart from the odd lines. So, when objects are moving rapidly, you see a stair-stepping pattern. It's ugly, increases data rates, and is easy to eliminate. Every digital video program out there can deinterlace, so do it, or we'll make fun of you at E3. See Dune 2000.

3. Uncropped video. In analog video, there's almost always crud around the edges of the image that isn't seen on TV. Computer monitors show every last pixel all the way around. So you need to crop out the crud on the edges. See the Longbow series.

4. Wrong aspect ratio. Pixels are always square, right? Wrong. Many high-end digital video standards use 720×486 pixels per frame, where the pixels are taller than they are wide. So if the video is played like that on a computer, everything looks wide. Scale the video so that the pixels wind up square and circles aren't ovals.

5. Bad actors. It's an old saw in video production that many good stage actors don't work out well in front of a camera, because their performances are aimed at people thirty feet away in the audience, not two feet away in a close-up. See almost every FMV game made.

6. Poor voice acting. In many ways, poor voice acting is much more of a problem than poor video. There are two main reasons for this: audio is everywhere in games, and voice acting looks easy. Voice acting is not easy. If it was, it wouldn't be obvious which computer games had the engineers do voice-overs. Voice acting requires skill and experience, and it's a lot more expensive to spend eight hours in a recording studio with a volunteer than two hours with a professional.

Ask this question before using someone as a voice actor: "Are you a member of AFTRA?" If they respond yes, then they're almost certainly a skilled professional. If they give you a speech about how they don't like unions, at least they know what business they're in, and might be okay. If they say "What's AFTRA?" smile and go somewhere else.(AFTRA is the American Federation of Television and Radio Artists.) See the Resident Evil series.

7. Wrong frame rate. Video runs at 30 frames a second, and film at 24 frames a second. The final frame rate of digital video should be an integer fraction of the original frame rate in order to preserve smooth motion. For example, video should play back at 30 FPS or 15 FPS, and film at 24 or 12. Note that if you transfer film to video, you still should restore the frame rate to 24 FPS. Media Cleaner Pro does a great job with this conversion. See virtually every game with content shot on film, such as Close Combat.

8. Eight-bit audio. Never, ever, ever use 8-bit audio for anything. Ever. Really. With modern audio codecs, 8-bit always produces bigger files and sounds worse than 16-bit.

9. Mismatched foreground and background elements. So, you want to shoot actors on a bluescreen, then incorporate computer graphic elements in the background? The video looks great, and so do the computer graphics, but why don't the spaceships look like they're in the same universe as the talent? Lots of reasons, probably. In order to make elements match, they need to have similar frame rate, grain, motion blur, lighting highlights, and so on. A good animator can make a great looking spaceship. It takes a great animator to make it look like part of the set.

10. Video technical requirements out of sync with the rest of the game. While I like codecs as much as the next guy (well, a lot more than the next guy), it's a mistake to have the processor requirements for your cutscenes be higher than for the rest of the game. Yet, it happens. Sometimes, testers get so used to skipping past the video that they never check cutscenes on low-end computers. Great looking video doesn't count if users can't see it. See Civilization II and Close Combat: A Bridge Too Far.

Ensuring Video Production Success

Creating quality FMV. Far too often, I'll see video that looks *almost* professional. For example, Dune 2000 (along with just about every other Westwood title using live action), had crystal-clear interlacing artifacts caused by not removing one of the two fields that makes up a video frame. Just about any digital video professional could have corrected this problem with a mouse click. These artifacts are distracting, and make digital video compression much more difficult.

Problems such as this can crop up in every stage of in the production of video, from lighting to direction to compositing to editing. The video in Black Dahlia, a game completely dependent on the success of its cinematic elements is, quite simply, professional. The lighting on actors matches backgrounds. They are well placed within their environment. 3D animation is married sensibly and elegantly to stock footage, as well as to newly shot elements. The bottom line is make sure that you have video professionals working on your FMV segments, otherwise your quality could suffer.

Actors need good direction. Game developers have often shown the ability to pull wooden performances from very talented actors. Many a time, I've seen some of my favorite character actors deliver lines with all the conviction of a Creationist giving the keynote address at a Darwinists' convention. These problems often stem from a fairly simple issue: the talent has been inadequately prepared for the task at hand. If the developer is in charge of the production of the video assets, then the fault lies there.

When making the transition from the stage to film or television, actors often complain about the lack of linearity. Movies are rarely shot in sequence, and an actor used to being able to carry their character from scene to scene can get lost in the technical considerations of production. What's worse is that the nonlinearity of game plots can make an actor's role even more difficult when filming game video.

In the game developers' world, video sequences often exist around game sequences, and each segment

might come from and go to many branches. Actors can find it difficult to maintain a strong grasp on what their character is doing or, for that matter, who they are.

This fact can result in flat, unappealing performances. Therefore, if you're going to be working with actors, empower them with an understanding of the process they're involved in. Some developers have taken to bringing large flowcharts to the set, so that the actors can see where they are in a given scene, and understand how it relates to the rest of the game. Developers have also learned the value of rehearsals and read-throughs of the script. Many problems with a scene can be caught during this process.

Keep your animation animated. Almost all titles today use 3D animation in their cutscenes, either as background elements for actors (see the sidebar "The Black Art of Chroma Key Compositing") or as the sole element in the video. Most modern game development teams have top notch 3D artists on their teams. They are brilliant at taking complicated objects and characters and building low–polygon-count 3D models with depth and life. However, these skills don't necessarily translate into the ability to produce an animated segment for a videogame.

For example, JEDI KNIGHT: DARK FORCES 2 opens with a long, drawn out flythrough over Nar-Shaddaa. Although this is a wonderful example of whatever modeling program the animator used, it's long and uninvolving. There is nothing wrong with using animation to introduce new settings and hardware. Blizzard has done a very good job of that with STARCRAFT. You just can't assume that a camera spinning around a 3D object is inherently dramatic. Without any sort of context, it's boring.

Post-production preparedness. Digital video post production is often thought of as simply the stage where things that went wrong are corrected — as in, "We'll fix it in post." However, as mentioned earlier, it's a lot cheaper do to something correctly the first time. Keep that in mind.

Post production should be thought of as fine tuning what you've already done right, not correcting earlier flaws. This is the stage where a scene's structure and rhythm are honed via video editing, animation and live action are merged, and graphics and subtitles are added.

Never underestimate the critical importance of good editing. It can be tempting for some computer animators to create an entire scene purely with camera motion instead of using cuts — almost always a terrible idea. Cutting from different camera angles and subjects is core to our video experience. A great way to understand how this works is to watch TV with the volume off. Notice how often the cuts happen during a scene. Note how they set up the scene, emphasize facial expressions, and set up a rhythmic pace. INTERSTATE '76, with its primitive graphics, was able to really capture the feel of a 1970s action film by mimicking their editing style.

Some big differences exist between traditional video post and post for game delivery. First, traditional video is made up of tall, rectangular pixels.

Computers use square ones. Second, each traditional video frame is made of two unique picture fields interlaced into one frame, with all the even lines capturing a moment in time a fraction of a second different from the odd lines. Video in games is always progressive scan, with only one image in the frame (see "Ten Digital Video Disasters").

Because RGB computer monitors are much better than analog televisions, games can use film-like color saturation and contrast. The drawback to the high-quality images produced by computer monitors is that a lot of subtle noise that would go unnoticed on a television is often readily apparent on the monitor. This can be especially obvious in shadow tones, and it requires processing by a tool such as Media Cleaner or After Effects to reduce the noise.

Digital video codecs also have limitations that should be addressed in post production. For example, Smacker and other codecs that use a limited palette of on-screen colors benefit from video processing that limits the color palette used in any given frame. Other codecs have trouble with certain types of content, such as rapidly moving complex textures. Perform some sample tests of your post and edits well before final compression, so you can alleviate trouble spots in the video.

The Future of Games

So, where is all this going? Great places. Over the next few years, DVD-ROM will become ubiquitous, as will MPEG-2 playback. These advances will increase the technical quality of video in games beyond our current goal of "broadcast quality." The larger media size of the DVD-ROM will also allow more video storage capacity per disc. A dual-layer, single-sided disc can hold over four hours of extremely high-quality video, with one gigabyte left over for the rest of the game.

Not all video will go over to MPEG-2, though. While great for standalone cut scenes, MPEG-2 is difficult to use within games themselves, where video is displayed in only part of the screen or at the same time that other things are going on. Fortunately, a number of next-generation software-based codecs are hitting the market, such as Duck's TrueMotion 2X and Rad Game Tools' Bink. Both provide very high-quality content at reasonable data rates and both should be released by the time you read this.

Today's video production tools are much better suited to game production. The whole video world is going digital, which massively improved the quality and costs of game video. Most of the major video manufacturers have announced progressive scan cameras, which will eliminate the interlacing conversion problems (thereby doubling effective resolution) that have limited quality since the beginning of the game industry. In addition, the next wave of high-definition cameras will offer the possibility of film-quality video on the desktop. The price of professional-quality equipment is also dropping very rapidly, to the point where today's $4,000 camera is better for game video than the $40,000 rigs of ten years ago.

As playback quality increases, so will the demands on developers to produce high-quality video. After decades playing the poor cousin to the broadcast world, DVD and next-generation video playback engines will offer us the opportunity to provide the end user will far better that broadcast quality. Game developers have an opportunity to take a leadership position in video quality. ■

FOR FURTHER INFO

- *Filmmaker's Handbook*, by Steven Ascher and Edward Pincus (New American Library Trade, 1984), is still the best primer available, and a new version should be available by the time you read this.
- http://www.codeccentral.com is a Terran-hosted web site that's a great source for codec information
- *Game Developer*'s sister publication *DV* is a great source for information on video production and technologies. More information can be found at http://www.dv.com.

Ca i g Shad V l me

by Hubert Nguyen

As I recall, the very first 3D game with real projected 3D shadows was VIRTUA FIGHTER, a Sega AM2 arcade game based on a Model1 board. That was the beginning of a new era of fighting games. Each character was composed of more than 1,500 flat-shaded triangles. Their shadows were projected onto a perfectly planar ring. Sega AM2's developers had several reasons for choosing a simple planar ring.

1. A planar area was sufficient for a terrific 30 FPS game play.
2. Casting shadows on a relief is CPU-hungry because it's not hardware-accelerated.
3. The main processor (a 680x0 family) was too slow to handle the inverse-kinematics mathematics needed for characters on a nonplanar area (which didn't appear until arcade machines started featuring the Model3 board).

For years, fighting games were the only games to use real-time shadows. These games typically featured only two characters, each casting a shadow on the floor. But these games lacked interobject shadow casting, such as the first player's shadow projected onto the second player's. Nowadays, with the power of arcade systems such as Sega's Model3 family, games such as VIRTUA FIGHTER 3 are capable of more elaborated special effects. Nonplanar shadows are now possible, though interobject shadow casting is still missing.

My love of fighting games inspired my efforts to come up with a way to create shadows that could be projected onto any object, even onto the other fighter (Figure 1). As with many techniques, the one I'll describe in this article has advantages and disadvantages, but I think the idea is worth sharing.

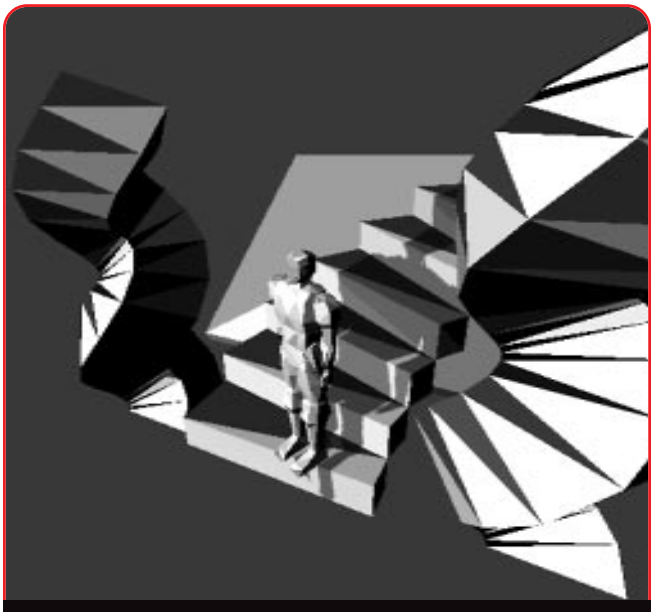


FIGURE 1. *An example of projection shadow mapping, in a screenshot taken from the sample application accompanying this article.*

The Easy Answer?

There are several ways to create 3D shadows. The most popular technique is to project the whole mesh (or at least the part that's visible from the light) onto a single plane (Figure 2). Some developers are using a simplified mesh to cast the shadow (for example, TEKKEN 3 from Namco). This technique dramatically reduces the number of triangles that must be processed, and usually gives acceptable visual quality. Proceeding with polygons does offer benefits such as

Nguyen Hubert Huu lives in Nanterre, France. He built demos for Impact Studios between 1993 and 1995, and is now working for a French game publisher. You can contact him via e-mail at nguyenhub@aol.com

FIGURE 2. *An example of shadow created with triangles projected on a plane.*

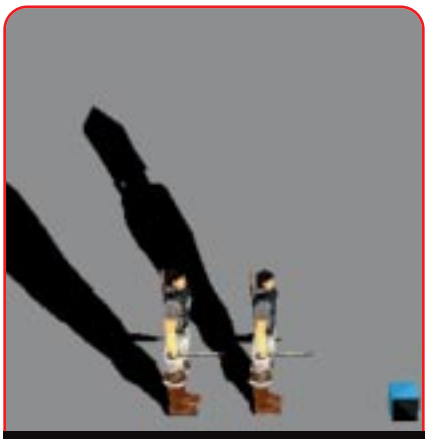


FIGURE 3. *Note that edges are sharp even if the shadow is stretched.*

LISTING 1. *Establishing the relationship between* **SrcObject** *and* **DstObject.**

```
01    loop on SrcObject polygons
02    {
03    loop on DstObject polygons
04       {
05          project SrcObject polygon on DstObject polygon plane
06          if ( projected SrcObject polygon shadow in DstObject polygon)
07          {
08             Newpolygon <- project SrcObject polygon on DstObject polygon plane
09             Shadowpolygon <- clip Newpolygon against DstObject polygon edges
10          }
11       }
12    }
```

razor-sharp edges (Figure 3) and speed (the game can render the shadow in flat-shading, which is usually faster than fully lit texture-mapped rendering). Projecting and clipping polygons on an infinite plane is pretty fast.

Also, on a nonplanar surface, creating polygons for a shadow is more difficult. First, you have to perform the projection onto a general plane, rather than the typical single-axis–aligned plane (vertical or horizontal). We could say that casting a shadow onto something other than a plane is equivalent to projecting one object onto another, both composed of polygons. Let's call the shadow-caster object **SrcObject** and the receiver of the shadow **DstObject**. For each polygon of **SrcObject**, we'll have to determine which polygons of **DstObject** will receive the shadow. One could represent the process with the pseudocode shown in Listing 1.

Of course, this algorithm is a little "brutal." It can be optimized to reduce the number of polygons that must be processed. For example, we could skip those polygons that are light-face–culled. Nevertheless, the overall complexity of this algorithm will always be pretty high. Lines 05 and 06 can be quite CPU-intensive compared to a simple planar shadow casting without clipping. We could further optimize this technique's performances by precalculating and storing certain data, such as the planes equation for each **SrcPolygon**, **DstPolygon**, and so on. But this algorithm is still a pretty complex process.

The code in Listing 1 will give us a brand new set of polygons with which to build the shadow. In most cases, the number of triangles in the shadow will be greater than or equal to the number of triangles in the **SrcObject** that are visible from the light source because a **SrcObject** polygon can be projected onto more than one **DstObject** polygon. This calculation is probably the costliest part of this algorithm. Fortunately, alternatives do exist.

Projecting Shadows

I first realized that I could use a texture to cast a shadow when I saw a 3Dfx demo featuring a spotlight implemented as a projected texture. I imagined how easy it would be to render a shadow into the projected texture. Projecting textures doesn't require CPU-intensive operations such as projecting onto general planes and clipping polygons. All we have to do is calculate the texture coordinates for applying a rendered texture onto an object. Projecting shadows employs a similar technique. The principle is to generate textures dynamically using a light source as the point of view.

The idea is pretty simple. Let's imagine that we want to project a picture onto an object. we simply transform each vertex of the object into the projector space (the light source), and use its new coordinates (x',y',z') to create the texture coordinates (s,t). This technique is even more intuitive when you can see what's happening. Figure 4 is

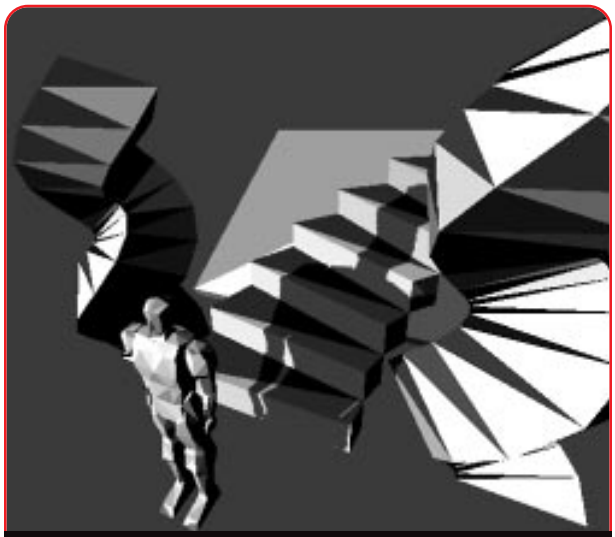


FIGURE 4. *Let's see how to achieve this effect...*

FIGURE 5. *This is the scene from the light's field of view.*

FIGURE 6. *These objects will receive the character's shadow.*

FIGURE 7. *The shadow texture map.*

the final result that we want to obtain. Figure 5 shows the scene and the figure from the light's point of view. You can see in Figure 5 that the shape of the shadow in Figure 4 is exactly hidden by the figure of the man when viewed from the light's point of view. The scene will receive the shadow. Figure 6 shows the scene objects from the light's point of view; we need the coordinates of these objects in order to calculate the (s,t) coordinates for the shadow-mapping. Figure 7 shows just the shadow texture — we've rendered the objects for which we want to create shadows (in this case, the character only). By projecting the texture in Figure 7 onto the objects in Figure 6, we get the final effect (Figure 4). In theory, casting shadows using textures is as simple as that.

Now let's look at the code. A sample application that performs these calculations is available from the *Game Developer* web site. The most important part of the algorithm computes the texture shadow coordinates for all the objects onto which we want to cast shadows. This calculation basically remaps the (x,y,z) coordinates of the object into (s,t) coordinates in the shadow map (Listing 2).

Here, the (x,y,z) coordinates are computed in the light's space coordinates (the object has been previously transformed by the `Rotate()` function). Figure 8 helps to visualize how the mapping is performed. The projection is planar — we only use (x,y). Because the object has been projected with its perspective relative to the light's point of view, the mapping takes care of the perspective. Figure 9 illustrates the result of this operation. Once we've remapped the (x,y) coordinates as (s,t) coordinates, we simply use (s,t) as normal texture coordinates. In our case, remapping was as simple as reusing the (x,y) coordinates as (s,t) coordinates.

Obviously, we need to consider the size of the shadow texture and the size of the viewport used to transform the object into the light's point of view. The sample application uses Glide, which has texture coordinates in the range

LISTING 2. *Remapping the (x,y,z) coordinates of the object into (s,t) coordinates in the shadow map.*

```
void CastOnMesh(Mesh &m)
{
float prod;

for (int i=0; i<m.nTriangles; i++)
{
   int v1= m.TriangleArray[i*3+0];
   int v2= m.TriangleArray[i*3+1];
   int v3= m.TriangleArray[i*3+2];
   m.FlagArray[i] = 0;     // by default, reset the flag

   // Backface Culling
   //((v3.x - v1.x) * (v2.y - v1.y)) - ( (v2.x-v1.x)*(v3.y-v1.y) )
   prod =
   (
   ((m.VertexArray[v3].px-m.VertexArray[v1].px)*(m.VertexArray[v2].py-m.VertexArray[v1].py))
   -((m.VertexArray[v2].px-m.VertexArray[v1].px)*(m.VertexArray[v3].py-m.VertexArray[v1].py))
   );
   if (prod<0.0f) continue;       // reject by BackFace Culling

   if ((m.VertexArray[v1].flags &
      m.VertexArray[v2].flags &
      m.VertexArray[v3].flags) != 0) continue; // reject if the triangle is completely
         // outside of the light POV
   m.VertexArray[ v1 ].u = m.VertexArray[ v1 ].px; // (x,y,z) are in Light FOV
   m.VertexArray[ v1 ].v = m.VertexArray[ v1 ].py;

   m.VertexArray[ v2 ].u = m.VertexArray[ v2 ].px;
   m.VertexArray[ v2 ].v = m.VertexArray[ v2 ].py;

   m.VertexArray[ v3 ].u = m.VertexArray[ v3 ].px;
   m.VertexArray[ v3 ].v = m.VertexArray[ v3 ].py;
   m.FlagArray[i] = 1;        // 1= draw this triangle for the current frame
   }
}
```

FIGURE 8. *Superimposition of the scene's position and the texture from the light's point of view.*

FIGURE 9. *The scene has been displayed using the shadow texture.*

[0..256] instead of the range [0..1.0] used by most of APIs. Because the shadow texture is a square 256×256, we don't have to scale the (x,y) coordinates when using them as (s,t) coordinates.

Crossing Hurdles

We need to be careful of a couple of things in regard to texture coordinates. Some (s,t) couples will have values outside of the range [0.0..1.0]. In order to avoid a tiling effect, which would distort the result by repeating the shadow in an undesired place, the rendering device must be configured into clamp mode.

Every texel can be addressed by a couple of texture coordinates (s,t) in the range [0.0..1.0]. In theory, giving a value outside of this range to s or t would cause an addressing error. Most people who've programmed a software rasterizer have encountered this problem. To avoid it, many hardware manufacturers have methods of keeping texture coordinates inside of the texture. These include tiling and clamping. Both modes modify the texture coordinates after they've been interpolated by the chip for each texel. When configured for tiling, the chip removes the integer part of the (s,t) coordinates (for example, 1.7 becomes 0.7). Clamping, on the other hand, brings the values that exceed the range to the nearest bound (for example, -2.3 becomes 0, and 1.3 becomes 1.0).

Figure 10 shows a clamped square that has texture coordinates greater than 1.0 and less than 0.0. Figure 11 shows the opposite, with tiling enabled. If you look closely at Figure 10, you can see that if the borders of our texture aren't clean, we'll get an unwanted effect on the display of our shadow. Figure 12 is an example the sort of problems that can arise in clamp mode. This bug is caused by the object being clipped by one of the texture's borders (Figure 13). To avoid those effects, we must make sure that our object is completely inside the point of view and isn't being clipped by the texture's border. We must be sure that our textures have clean borders, with a width of at least two pixels. In the sample application that accompanies this article, I've attached the light to the object and adjusted the point of view to avoid any clipping. A piece of code could automatically check whether or not the object is inside the point of view. A good adjustment will maximize the accuracy of the shadow in the scene by increasing the surface taken by your shadow in the texture. We'll talk about that at the end of the article.

Implementation

Now that we understand how texture projection works, let's look at how it fits into the rest of the program. I've divided the process up into five steps:

1. Render the object for which we want to create a shadow from the light's point of view.

FIGURE 10. *Clamping example.*

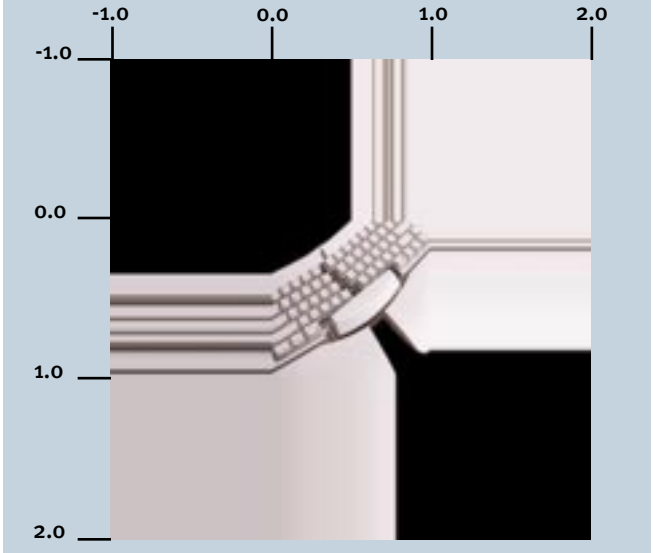


FIGURE 11. *Tiling example.*

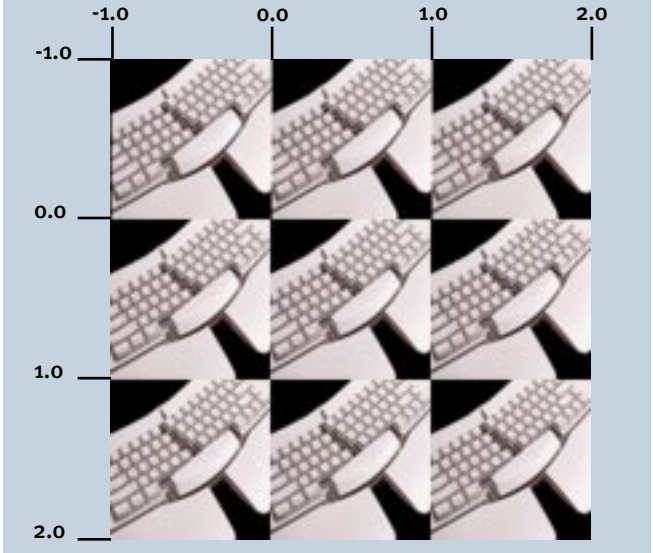


FIGURE 12. *Clamping the texture coordinates can cause strange display effects.*

FIGURE 13. *The bug in Figure 12 is due to this clipping of the character into the shadow texture.*

2. Create a texture using the result of Step 1.
3. Project the texture by calculating (s,t) coordinates for each object onto which we want the shadow to be projected.
4. Render the scene from the camera's point of view.
5. Render the triangles that are visible from the light's point of view using the (s,t) calculated during Step 3 and the texture from Step 2.

This is the basic method behind shadow projection. Now let's look at each step in detail.

STEP 1. We have to render the figure of the object that will cast a shadow in our scene. We'll need to transform, project, and render the object into the light's point of view. (A flat rendering is sufficient to achieve a simple shadow, but you can imagine a lot of special effects that could be implemented with this technique.) We can speed up this operation by using a simplified mesh (a level of detail mesh, for example); the effect on the final result won't be too great, especially if the object is animated (a character, for example). The drawing of the triangles is pretty fast because of the simplicity of a flat rendering (particularly for software rasterizers). The fastest rendering, of course, can be achieved with the help of a 3D accelerator. Because we're only interested in a figure of the object, we can deactivate the Z-buffer or any other sorting technique — the result will be the same, and the overall performance of this step will improve. As usual, any optimization aimed at reducing the number of triangles to be rendered would help speed things up (backface culling and such).

STEP 2. Copying the rendered figure from Step 1 into a texture is currently the slowest part of the sample application, due to the slowness of the LFB (linear frame buffer) access on the Voodoo chipset. Normally, this operation should be a lot faster. Some chips allow hardware blitting between the frame buffer and texture memory. Better still, some boards — such as the 3Dfx Voodoo Banshee — let you render directly into texture memory. In the latter two cases, you can consider this step as almost free, and expect a gain of 50 percent in performance for the sample application provided with this article.

STEP 3. Now, we transform all of the objects upon which we want to cast the shadow into the light's point of view. This is a good time to flag triangles that aren't visible in the light's point of view, so that we can reject them during Step 5. We then calculate the new (s,t) coordinates to perform the projection of the shadow. The speed of this step depends on the number of vertices in the mesh. In general, the complexity of the mesh isn't detrimental to the final frame rate. This step should typically account for about 15 to 20 percent of the overall frame. So, although it's not a critical issue, one could easily speed up this process. For example, we could use occlusion techniques (potentially visible sets, portals, and so on) that will decrease the number of vertices to process in the scene. Backface-culling can also divide the number of vertices by two for most objects.

STEP 4. We render the whole scene as we normally would. We don't need to worry about shadows for now.

STEP 5. Now we're going to render all the triangles that we determined were visible in the light's point of view during Step 3 (Figure 6). We'll texture these triangles with the texture map produced in Step 2 and we'll use the texture coordinates computed in Step 3. All of these triangles should have been rendered in Step 4. We are therefore using multipass rendering to render them a second time for the final result. Figure 14 shows the triangles that have been rendered twice to perform a multipass rendering. To setup multipass rendering, we have to configure the Z-buffer or W-buffer in Less or Equal comparison mode or the second pass (Step 5) won't be visible because the pixels will be rejected by the depth-sorting.

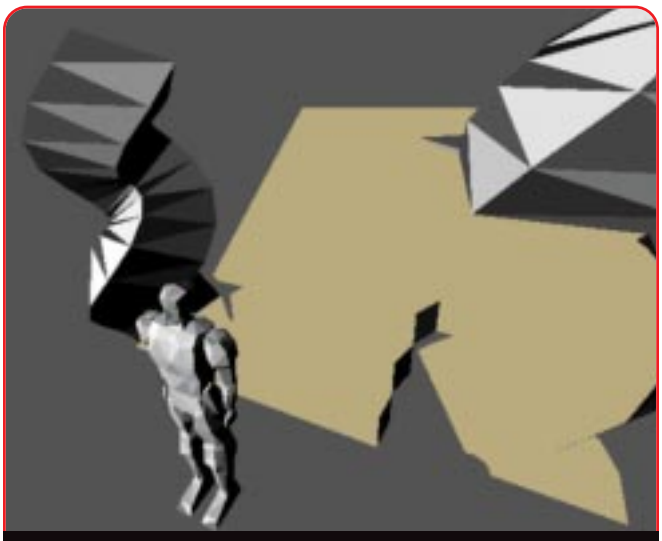


FIGURE 14. *The triangles that are visible in the light's field of view are displayed with colored flat rendering.*

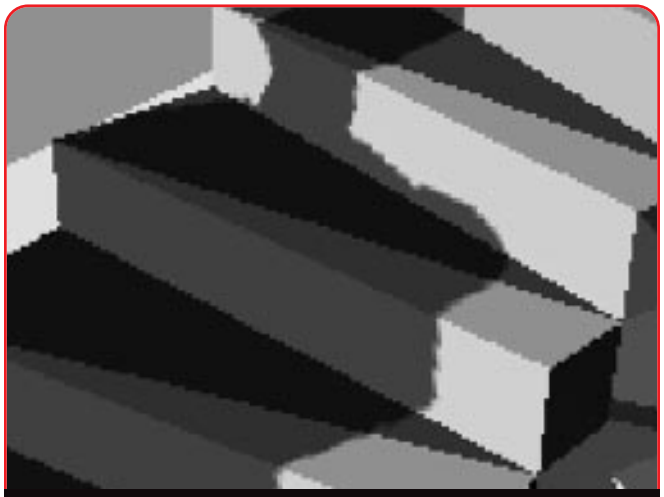


FIGURE 15. *Using an alpha shadow texture allows filtering on the edges of our shadow.*

FIGURE 16. *A RGB 565 pixel.*

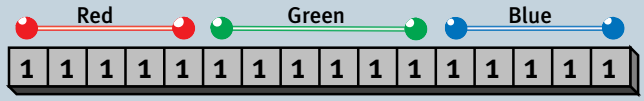
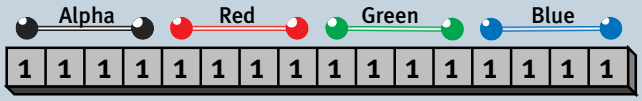


FIGURE 17. *A 16-bit pixel again, but in ARGB 4444 format.*



We also need to set our hardware to skip pixels that are of the background color in the shadow texture. The simplest way to do this is to render the shadow with an RGB color of 0x00000001 (ARGB value) on a 0x00000000 background. To reject the background but not the shadow, we can use the **grChromakeyValue()** function (or its equivalent in whatever API you happen to be using) to configure the rasterization device to reject the background pixels. However, the chroma-keying solution doesn't take advantage of bilinear filtering, which helps smooth out the otherwise pixilated texture.

A still better technique is to use a texture with an alpha channel (ARGB-4444 format in most hardware; I'll explain later how I generate an ARGB-4444 texture in a 565 frame buffer). We can use the alpha value in the texture to discard pixels of the background color. The point in using an alpha texture is to take advantage of filtering to obtain smoother edges. The alpha value will be interpolated in the same way as the colors. The edges of our shadows will appear smoother than if we'd used a simple color-key test. In some cases, we can even get an effect that resembles antialiasing. Figure 15 demonstrates the filtering capabilities of this technique. We can get sharper edges by increasing the size of the shadow texture, although this will slow down its generation. With higher fill-rates and Unified Memory Architecture (as on the Banshee, for example) the texture's size will be less of an issue.

Listing 3 shows how to configure Glide before drawing the shadow. Listing 4 shows the settings, which we perform once at the beginning of the program. Some of these may influence the rendering of the shadow texture.

Odds and Ends

The alpha combine is set to use both texture and iterated alpha at the same time to adjust the transparency of the shadow. The alpha of the texture encodes the mask of the shadow mask, and the iterated alpha allows for dynamically adjusting the transparency of the shadow. Using different degrees of transparency allows us to simulate variable light intensity. For example, if the spotlight is near the object, we would set the iterated alpha to 255.0 to get a black (opaque) shadow. On the opposite end of the spectrum, if the spotlight is far off or the light intensity is low, we would set alpha closer to 0.0 to get a very soft shadow.

I'm currently using the same alpha value for all vertices, but one could consider using more exotic per-vertex alpha calculation (such as computing an alpha value that is a function of the distance between the vertex and the light source).

Previously, I talked about generating an ARGB 4444 texture from a RGB 565 frame buffer. This is a simple trick. We need a black texture with an alpha mask representing the figure of our object. I have a RGB 565 texture. Figure

LISTING 3. *Configuring Glide to draw the shadow (continued on page 52).*

```
void DrawShadow(Mesh &m)
{
   guAlphaSource( GR_ALPHASOURCE_TEXTURE_ALPHA_TIMES_ITERATED_ALPHA );

   grAlphaBlendFunction( GR_BLEND_SRC_ALPHA,
         GR_BLEND_ONE_MINUS_SRC_ALPHA,
         GR_BLEND_ONE,
         GR_BLEND_ZERO );

   grTexCombineFunction(GR_TMU0, GR_TEXTURECOMBINE_ADD);
   guColorCombineFunction(GR_COLORCOMBINE_TEXTURE_TIMES_ITRGB);
   grColorCombine(
      GR_COMBINE_FUNCTION_SCALE_OTHER,
      GR_COMBINE_FACTOR_LOCAL,
      GR_COMBINE_LOCAL_ITERATED,
      GR_COMBINE_OTHER_TEXTURE,
      FXFALSE);

   grTexClampMode(GR_TMU0,GR_TEXTURECLAMP_CLAMP,GR_TEXTURECLAMP_CLAMP);
```

LISTING 3. *Configuring Glide to draw the shadow (continued from page 51).*

```
   for (int i=0; i<m.nTriangles; i++)
   {
      int i1 = m.TriangleArray[i*3+0];
      int i2 = m.TriangleArray[i*3+1];
      int i3 = m.TriangleArray[i*3+2];

      if ( m.FlagArray[i] == 0) continue;

      if (
         ((m.VertexArray[ i1 ].flags | m.VertexArray[ i2 ].flags | m.VertexArray[ i3
].flags)
         & Vertex::CLIPZ) == 0)
      {
         v1.x    = m.VertexArray[ i1 ].px;
         v1.y    = m.VertexArray[ i1 ].py;
         v1.oow  = m.VertexArray[ i1 ].pz;
         v1.tmuvtx[0].sow = m.VertexArray[ i1 ].u * v1.oow;
         v1.tmuvtx[0].tow = m.VertexArray[ i1 ].v * v1.oow;
         v1.r = 255.0f;
         v1.g = 255.0f;
         v1.b = 255.0f;

         v2.x    = m.VertexArray[ i2 ].px;
         v2.y    = m.VertexArray[ i2 ].py;
         v2.oow  = m.VertexArray[ i2 ].pz;
         v2.tmuvtx[0].sow = m.VertexArray[ i2 ].u * v2.oow;
         v2.tmuvtx[0].tow = m.VertexArray[ i2 ].v * v2.oow;
         v2.r = 255.0f;
         v2.g = 255.0f;
         v2.b = 255.0f;

         v3.x    = m.VertexArray[ i3 ].px;
         v3.y    = m.VertexArray[ i3 ].py;
         v3.oow  = m.VertexArray[ i3 ].pz;
         v3.tmuvtx[0].sow = m.VertexArray[ i3 ].u * v3.oow;
         v3.tmuvtx[0].tow = m.VertexArray[ i3 ].v * v3.oow;
         v3.r = 255.0f;
         v3.g = 255.0f;
         v3.b = 255.0f;

         v1.a = 170.0f;// use the "Iterated Alpha" to adjust the
         v2.a = 170.0f;    // "transparency" of the shadow for performing
         v3.a = 170.0f;// special effects like gradients and so on...

         guDrawTriangleWithClip( &v1, &v2, &v3);
      }
   }
     grColorCombine( GR_COMBINE_FUNCTION_LOCAL,   // turn OFF the "texture mapping"
                 GR_COMBINE_FACTOR_NONE,
                 GR_COMBINE_LOCAL_CONSTANT,
     GR_COMBINE_OTHER_NONE,
                 FXFALSE );

   grAlphaBlendFunction( GR_BLEND_ONE,         // disable the "alpha blending"
         GR_BLEND_ZERO,
         GR_BLEND_ONE,
         GR_BLEND_ZERO );
}
```


16 shows the RGB values in a 16-bit 565 pixel. Figure 17 shows the same thing in an ARGB 4444 pixel. We need an alpha mask with a value of 1111b for each pixel. If we compare the bits of the 565 texture, it corresponds to a part of the 565-Red, 11110b exactly. So, all we need to do is to render a flat shape with the value 1111000000000000b and copy it as an ARGB 4444 texture. This gives us an alpha mask with all alpha pixels with a value of 1111b. Listing 5 shows how it's written the sample application.

The **grTexDownloadMipMap()** function loads the memory from the ShadowBitmap buffer into the Voodoo texture memory. Because 4444 and 565 textures are both 16 bits per pixel, the function doesn't care about how bits are organized inside. It copies a block of memory to the address specified on the board (as it happens here, zero). The interesting part of this operation is that we can set up the hardware for a specific texture format with the **grTexSource()** function. All we have to do is to fill the **GrTexInfo** structure with the constant **GR_TEXFMT_ARGB_4444** for the format member instead of **GR_TEXFMT_RGB_565**. The function will configure the texture mapping unit to decode an ARGB 4444 texture. Most boards now support this format, so don't worry about the compatibility of this technique.

Pros and Cons

As good as it is, this technique is far from perfect. Its most significant drawback is the low accuracy of the shadow (As compared to a polygonal shadow, which has sharper edges). In order to increase the shadow's accuracy, we have to increase the size of the shadow texture. Changing the size of the shadow texture from 256×256 to 512×512 would multiply the number of texels by a factor of four, which is significant with actual rasterizers (software or hardware), but is still possible. This is the main inconvenience of any sampling technique.

Furthermore, because we need a direction for doing the projection in the light's point of view, this technique can only handle directional lights. Omni lights, which cast light in every direction, cannot be used. The closest we could get to simulating an omni

light would be to create shadow maps for each object in the scene by pointing the spot towards each of them, and later compounding the effect of all the shadow maps on each object. This approach could be quite costly and may not be practical. Another constraint is the distance from the light to the object. Remember that we need to avoid clipping the object against the border of the texture. Thus, we can't place the object and the light too closely to one another. We must maintain a minimum distance between the light and the objects casting the shadows.

But this technique does offer significant advantages. The most important of these is its simplicity. Any 3D programmer could implement this method in a very short time. There are no complex mathematics or data to manipulate. It's just an intuitive technique that can provide great effects. And it's fast. Rendering the shadow texture doesn't cost much CPU time if it's handled by a hardware accelerator. A hardware **blit()** between the frame buffer and the texture memory will significantly speed up our shadow casting operation. We can even improve the performance by using a simpler mesh, although generating shadows from complex objects only slows down Step 1. The rest of the code (drawing the shadow) will run at exactly the same speed regardless of the complexity of the original mesh. In comparison, polygon-based shadow techniques take a bigger performance hit for higher-complexity objects.

The shadow map texture is a scalable technique. For scenes in which multiple objects are casting shadows, we can use differently sized texture shadow maps. For the main object, or for the object that is closest to the camera, we would use a high-resolution texture shadow map (512×512) and scale down to 32×32 or smaller for less important objects or objects that are farther away from the camera (small on the screen). It's analogous to an LOD technique, applied to the shadow texture. Finally, certain special effects could be based on this technique. One of these might consist of using opacity maps to create holes in the shadow maps (skeletons are a typical case). I trust *Game Developer*'s readers to adapt this technique to their needs, and I look forward seeing good-looking shadows in future games. ■

Acknowledgements

The author would like to thank the following people for their great support : Eliane Fiolet, Eric Smolikowski, Xavier Gerbier, Alexandre Macris, Miky Larsen.

Special thanks to Denis Amselem at 3Dfx for Glide support.

LISTING 4. *Our Glide settings.*

```
grCullMode(GR_CULL_POSITIVE);
grDepthBufferMode( GR_DEPTHBUFFER_WBUFFER );
grDepthBufferFunction( GR_CMP_LEQUAL );
grDepthMask( FXTRUE );
grTexFilterMode(GR_TMU0,GR_TEXTUREFILTER_BILINEAR,GR_TEXTUREFILTER_BILINEAR);
grGammaCorrectionValue((float)0.8);
grDitherMode(GR_DITHER_DISABLE);
```

LISTING 5. *Generating an ARGB 4444 texture from a RGB 565 frame buffer.*

```
// STEP-2
// read the rendered picture from STEP-1
// and use it as a texture
//-----------------------------------------
grLfbReadRegion(GR_BUFFER_BACKBUFFER,    // copy from the LFB to a buffer in
          0,                             // x                // SYSTEM MEMORY
          0,                             // y
          256,                // width
          256,                // height
          256*2,              // stride in bytes
          ShadowBitmap
          );
LoadTextureOnVoodoo();                              // download from the buffer in SYSTEM
MEMORY

This reads the 565 framebuffer in a system memory buffer allocated by the program, but the
real trick is in the LoadTextureOnVoodoo() function.

void LoadTextureOnVoodoo(void)
{
          GrTexInfo info;

          info.smallLod        = GR_LOD_256;
          info.largeLod        = GR_LOD_256;
          info.aspectRatio     = GR_ASPECT_1x1;
          info.format          = GR_TEXFMT_ARGB_4444;
          info.data            = ShadowBitmap;

          grTexDownloadMipMap(
                    GR_TMU0,
                    0,                                          // start addr
                    GR_MIPMAPLEVELMASK_BOTH,
                    &info
                    );

          // this should be done only once only since
          // I always use the same adress and texture format
          grTexSource(GR_TMU0,
                    0,                                          // startaddr
                    GR_MIPMAPLEVELMASK_BOTH,
                    &info
                    );
}
```

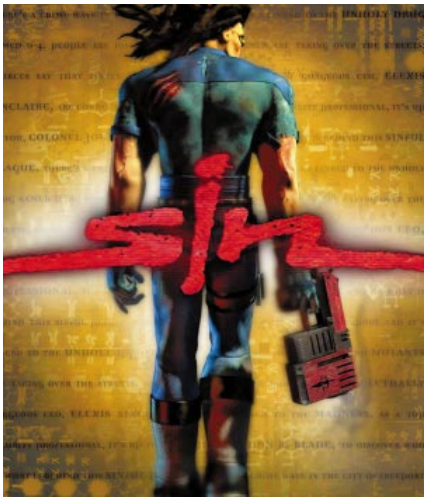
POSTMORTEM

Ri al E e ai me SIN

by Scott Alden

SIN is an original first-person shooter based on the QUAKE II engine with enhancements. Our previous effort here at Ritual Entertainment had been the highly acclaimed QUAKE MISSION PACK #1 – SCOURGE OF ARMAGON. SIN's story was developed during the completion of the SCOURGE OF ARMAGON, and the main level design started in earnest immediately after the release of the mission pack.

SIN's focus was on its story and characters; we wanted to breathe new life into the first-person genre. Instead of being a mindless shooter that simply progresses from level to level, SIN has interactivity beyond that of many shooters. One of our primary design goals was to implement Action-Based Outcomes (ABOs), meaning that a player's actions on certain levels will have an effect in later levels. From the outset, we decided to license the QUAKE engine and get started right away, and later integrate the QUAKE II source code to gain the benefits that it brought.



We displayed the initial prototype of SIN at the 1997 E3 in Atlanta, Georgia. The demo showed off the Geothermal Plant level, some of the original weapons, and the original monster design. We proceeded to flesh out the game design by describing all of the locations that the SIN world would encompass. SIN's initial design featured over 40 levels, 31 of which made it into the final game.

SIN's interactivity and ABOs represented a big part of the development effort. We spent a total of about three weeks behind closed doors just brainstorming each level's interactive features — features that had essentially no effect on the game's final outcome. We would play through a single level in the morning, then after lunch we'd bombard the level designer with ideas for making the level in question more interactive. Once, while playing through the bank level over and over, someone came up with the idea of placing an ATM machine in the level where players could access certain character's accounts. This detail turned out to be one of the most notable things in the game. We also discussed the possibility of blowing up the dam level, but in the end we scrapped the idea due to the amount of extra work we would have to do to get the level to look right. We also spent about four hours hammering out how SIN was going to end. These

Scott Alden (aldie@ritual.com) is a graduate of University of Florida Computer Engineering Department (1993). He worked as a developer support engineer at 3Dfx for a year, and then went to Ritual Entertainment as one of the three programmers on the SIN project. His next project will be programming on HEAVY METAL: F.A.K.K. 2. Ritual's Beau Anderson (beau@ritual.com) and Richard "Levelord" Gray (levelord@ritual.com) were of invaluable help in writing the art and level design portions of this article, respectively.

meetings were no holds barred, and we had to be careful not to stomp all over other people's work.

We had completed most of the initial level design by December 1997. With the game's design fully realized, we began to add the major content. Then all hell broke loose — we got the QUAKE II source code. So, with the source in hand, we began to rewrite of the game. We didn't really intend to rewrite the game, it just sort of happened. The QUAKE II engine's new capabilities and features allowed us to add lots of new ideas. Our animation system with bones and nodes allowed us to attach any weapon or object to any character. This feature let us have many different grunts/soldiers by attaching different weapons to the same model. The surface system that we added let us have context-specific footsteps and ricochet sounds depending on the material types. We also added an interactive music system that changed moods during action segments of the game.

Finally, our scripting system let us add some internally developed features fairly late in the development cycle. A level designer might, for instance, need a special command to perform a particular type of interactive element in his level. Because it was so easy to add commands to the scripting language, the programmer would usually oblige. We added two to three commands to the scripting language daily. In the end, we had over 400 script commands total. So, while we did experience some feature creep, and though these last minute details did push our release date back even farther, we were able to add a lot of detail near the end of SIN's development cycle.

For about three months before the release, deathmatch tuning took place about two nights a week. We would play the most recently created level, and then the e-mail flood would begin. Change this, change that, and so on.... During the final weeks before the game's release, we'd spend about 12 hours a day working on single-player functionality, and about 3 hours a day on multiplayer. Once multiplayer functionality was to our liking, we froze it. We didn't change a single bit of multiplayer during the last couple of weeks. We did keep playing it to make sure that anything we did for the single-player portion didn't break any of the multiplayer stuff.

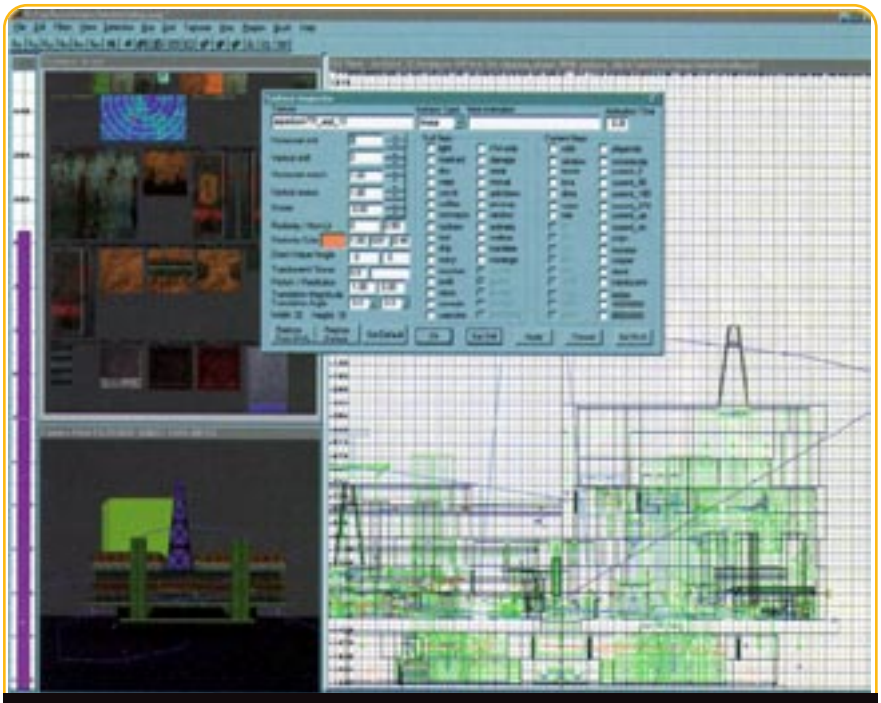
Things That Went Right

1. POWER IN NUMBERS. When we started designing SIN, one of the first things we did was to form a tribal team approach. This is the way the company is run as well. There are two main camps of project management. In the classical Roman Empire approach, decisions are trickled down through a rigid pyramid-shaped line of designers, managers, and finally the implementers. Conversely, in the Attila the Hun approach, the entire group is given equal input and has equal weight in most decisions.

We opted for the tribal approach because it offers the greatest pool of ideas from which to dip and it adds a certain synergy to the whole of the game's design. Often a project can become myopic and narrow-minded in the hands of only one or two grand designers. We've noticed that the number three is magical. The greatest game designs have resulted from one person's initial idea somehow supplemented by a second person and then

SIN

Ritual Entertainment
Dallas, Texas
http://www.ritual.com
Release date: November 1998.
Intended platform: Windows 95/98/NT
Project budget: $2 million
Project length: 20 months
Team size: The SIN team was made up of three programmers (Scott Alden, Mark Dochtermann, and Jim Dose), six level designers (Patrick Hook, Levelord, Tom Mustaine, Charlie Wiederhold, Matthias Worch, and Mike Wardwell), four artists (Beau Anderson, Robert Atkins, Michael Hadwin, and Joel Thomas), one project manager (Joe Selinske), one support person (Don Macaskill), one business person (Harry Miller), and one sound person (Zak Belica).
Artist/Level Designer workstation: 200MHz Pentium Pro with 128MB RAM
Programmer workstation: 300MHz Pentium II with 128MB RAM
Critical applications: 3D Studio Max, Photoshop 3.0 and 4.0, and MSVC++ 5.0



SinED, SIN's level editor.

finally finessed by a third. It's that third derivative that usually ends as an ultimate idea. For example, the oil rig level originally started out as a cinematic boat ride up to the rig. Someone came up with the idea of letting players circle around the rig until they sniped off all the guards walking around. This version was pretty good, but players could ride around the rig 20 times before they managed to get all the guards. So we decided to have the boat wait on each side of the rig until the player had killed the two guards walking around that side.

2. LICENSING THE QUAKE ENGINE. Licensing the QUAKE engine gave us a very stable base from which to begin. We were able to add new features and effects and then try them out pretty quickly. We also rewrote entire parts of the engine and heavily modified other parts. We wanted a higher level of interactivity than was available in the QUAKE game code, so we completely rewrote the game event system and AI code. We also built a character rendering system from scratch. The character system we used is a single-mesh hierarchical system. Finally, we added a bone-definition system so we could attach things such as guns or spears or spew out bubbles from any particular point. Even though we wrote a lot of the game code from scratch, the QUAKE II code was useful as an educational tool for game programming.

3. ANIMATION SYSTEM. The animation system was tied into our .DEF file system. The .DEF file is the extension we ended up using for all of our model text files. A .DEF file defines each character's animations and event triggers for specific frames of animation. The .DEF file is in plain text, so an artist can make updates to the file when he changes an animation. For example, if an artist changes the rocket launcher's firing animation, he can redefine in which frame the rocket would be fired in the game. Later, we discovered that we could develop a lot of special effects with the .DEF file system. We were able to create muzzle flashes, smoke, rocket trails, and various particle effects on the client without having to send over temporary entities with the networking system. We made the networking architecture so streamlined that firing a bullet only sent over one byte of information in the network packet.

4. ARTIST CONTROL. The artists had total control over the integration of the art into the game. Any artist had the capability to place a character in the game with a set of basic animations and AI. The artist could test the character (examine its skin, invoke its animation, and so on) within in the game very easily. We were able to attach any item or weapon to a bone location with one command. This flexibility allowed for a lot of tweaking to take place at the artist level. Artists tend to be much more critical of details, so giving them the ability to fix minor glitches without bugging a programmer was welcome. Furthermore, most of our artists were able to cross over into other disciplines, whether building models, skins, animations, or textures, or any number of other related tasks. This sharing of duties was important because of the large volume of art we had to complete.

5. OUR SCRIPTING SYSTEM INCREASED LEVEL DESIGNERS' CONTROL. We added a flexible scripting system so the level designers could create interactivity on their own. Previously, implementing most of the more interesting characteristics of levels was in the hands of the programmers. With our extensive scripting system in place, however, the programmers could focus on other areas rather than spending time writing specific pieces of game code for every level designer.

As I mentioned previously, the final version of our scripting language comprised over 400 commands. Level designers had intricate control over every aspect of level geometry, character animations, paths, and player interactions with the characters. Level designers could go far beyond the simple whizzing gizmos and script entire scenes of characters and machinery and gunfights. And, because we integrated the AI with the scripting system, the level designers were able to create a lot of specialized content. In the SIN world, bums will chat with you and give you clues, and civilians will cower in fear or run away.

SIN's scripting system was actually a full-blown multithreaded language — the level designers became programmers on their levels as well as architects. Besides, making these modifications was the most fun and rewarding aspect of level design. Level designers gained a lot of freedom, but only at the expense of time and effort on their part. To borrow terms from the movie industry, level designers have become the set designers, casting directors, directors, lighting engineers, gaffers… they have control over it all.

Things That Went Wrong

1. Young and naïve. Ritual is a new company and it went through a lot of growing pains during SIN's development. Many new faces came into the company, and many left. Actually, it is only now that the dust has settled that any real sense of a *bona fide* team can be seen. Our newly formed tribe felt little sense of cohesion, as most members were basically strangers to each other. A real team requires lot of faith and trust in order to act as a unit while performing something as creative as game design. This fact was especially evident in our decision-making processes. A clear vision was often muddled by too many inputs; settling on specifics often became impossible.

We've also identified an almost crippling realization that comes to all game developers during their first full game: We call it the Making Games Isn't a Totally Cool Job Syndrome. Crunch time sets in far too early, and a game's development cycle soon becomes a 24-hour-a-day, 7-day-a-week ordeal that lasts a year or more. Only hardened veterans know that the effort will prove worthwhile in the end, and that the end can be a long way off from the seemingly endless now.

2. Integration of the Quake 2 source code late in the project. SIN was originally scheduled for a Spring 1998 release, and we didn't get the QUAKE II source until late December 1997. We started the porting in early January, and it took a lot longer than we had anticipated. Because the QUAKE II source was drastically different from the original QUAKE engine that we started with, major systems were rewritten instead of ported. We rewrote the scripting system three times during the course of development. This was a major setback — finding issues with levels after they had been completed forced the level designers to rework each level, and in most cases, they just started over from scratch. We definitely learned our lesson in regard to trying to implement a major technology change midstream in a project.

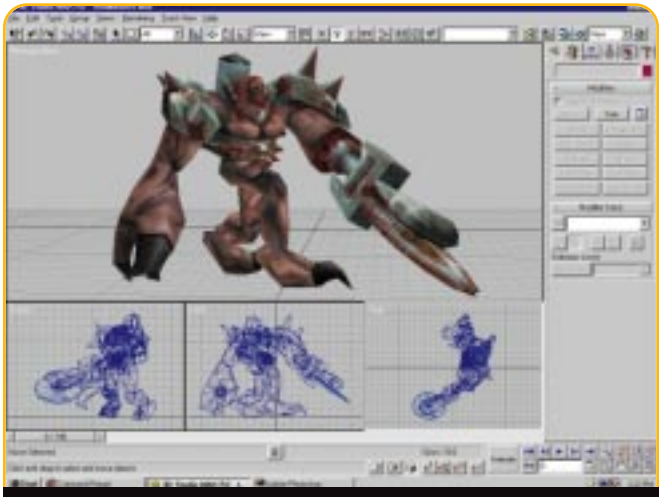
We also had problems using QUAKE II's new tools (qbsp3, qvis3, and qrad3) with our maps. A lot of changes went into the tools before any of the old levels could be compiled and used in the game. We had to write a complete texture grabber utility because our texture and surface properties format was so different than QUAKE's. Major additions were made to QUAKE's level editor (QE4), which we then renamed SinED.

3. Overcomplicated systems. Developers at Ritual are very sensitive to the needs of the first-person shooter gaming community, and one the reasons that QUAKE has been such a success is that it is easily changed and modified by the end user. Nonprofessionals in this community made hundreds of modifications and total conversions, and we wanted SIN to be just as flexible.

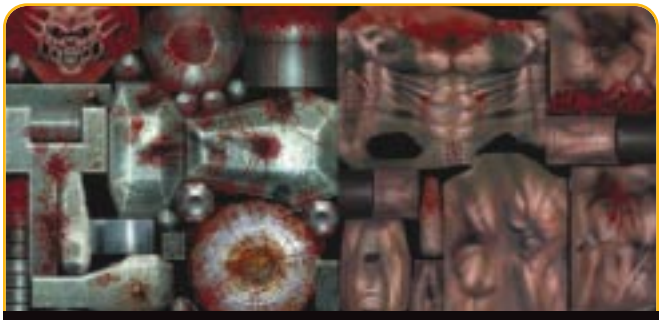
Most of the major systems in SIN can be changed without rewriting any

source code. Just make a simple change to a plain text file, and voilà, you have new behavior and effects. However, writing generalized source code is a lot harder than writing special case functions just to perform one task. Accommodating our goal of an easily modifiable game added a lot of extra development time to the new systems that we'd created.

An example of this is the console system, which we developed from scratch. This system had major ramifications in the game, and took about three to four man-months of programming time. We had not planned for consoles during SIN's design stage, and each level designer was responsible for the console content. It took a lot of planning and work to make a console, because the text messages were presented using a full layout language (like HTML). Something as seemingly simple as an ATM machine in a bank required tweaking over the course of several months. Moreover, the level designers already had a lot of information to absorb in creating a SIN level. Add 100 more console commands to the 400 script and AI commands, and you are just asking for trouble. Because consoles took so much work and effort, when crunch mode hit, consoles were underutilized and in some cases



Modeling ThrallMaster in 3D Studio Max.



ThrallMaster's skin.

dropped out from the level and replaced with a push button.

4. Inconsistency and too many assets. We lacked a standardized method of texturing models from the outset. We started out using the QUAKE method of planar projecting the front and back of a character. Planar projection worked well, but about halfway thorough the project, it became apparent that this method wasted a lot of texture space and wasn't the right way to map, especially with less organic models. When we got per-face mapping and 3D Studio Max 2, we took more care to unwrap the models carefully and use the texture space better. Most of the original models were redone, but a few slipped through due to time constraints.

Because it was our first major project as a team, we spent quite a bit of time just seeing what worked and looked good in the engine. We created over 3,000 textures, but many simply didn't look good in the game or were too general; we only used about a third of the original texture library. We also built over 400 in-game models, including around 35 characters. This huge model library made for some very diverse and interesting environments, but the overwhelming number of models represented a lot of work; sometimes, the quality suffered because of the sheer quantity of content that had to get done.

5. Too many cool things isn't too cool. The more that's in a game, the more there is to go wrong. Although creating the gadgets and gizmos and

character AI and environment interactions and such are some of the best aspects of level designing, level designing in particular has become far too complicated for any single individual to control. Level designing isn't simply making square rooms with connecting hallways anymore, and gone are the days of simply placing characters on a map and relying on their inherent AI to control them. The demand for realism has requires almost unattainable detailing to successful levels. Each and every character must be placed with paths and directions to escape points and attacking advantages, and most all their behavior must be scripted. Environmental interaction (entities) is also growing exponentially as level designers now have control over just about everything in their levels. A normal level can now take more than two months to complete, and play testing all of these added enhancements can be a nightmare. Lighting and sound have also become complicated and burdening. Level designing will soon become a multidisciplined department due to these increasing demands. We actually tried this organizational approach, but being a totally new concept, its actual implementation may have hindered more than helped SIN.

Wrap Up

When we finished SIN, we were pretty convinced that it was free of show-stopper bugs. Unfortunately, a few major bugs slipped through, and we immediately started working on the 1.01 patch. These bugs represented a big letdown to the team, because we literally had been in crunch time almost the entire previous year. We took a heavy beating from the online community, but after the patch was released, things started shaping up and some good reviews began to appear.

We definitely learned our lesson in regard to compatibility testing. In this day and age, it's extremely hard to test all possible hardware and software configurations, but at a minimum you should test all the major hardware vendors' peripherals. In hindsight, we should have released a quick, small patch to fix some of the major bugs, but

we worked on lots of little bugs that popped up during the first two weeks following the game's release. The 1.01 patch was about a 20MB download, but Activision offered a CD to anyone who sent in an e-mail requesting the patch.

In summary, SIN was a major project, and we spent a lot of time making it fun. If we had to do it all over again, we probably wouldn't make the conversion over to the QUAKE II source code; that was a major pitfall in the project. The entire team took a well-deserved vacation over the Christmas holiday and has returned refreshed and ready to begin development of HEAVY METAL F.A.K.K. 2 and and a yet to be announced project.■

by Mark S. Pierce

Public PC? Not Happening.

As a head of coin-op product development for Atari Games, and a speaker at past CGDCs (now the GDC), the number one question I hear is "What do you think about the Public PC?" I have to answer with the sad truth — which is quite different from the stance of well-meaning but misguided representatives of the Open Arcade Architecture Forum (OAAF).

As we enter 1999, there hasn't yet been a commercially viable product using this standard. I will be so bold as to say there never will be. A few games such as KICK IT have been represented as being a Public PC or Open Arcade games. This is not true. KICK IT is merely a product which uses an "off-the-shelf" PC to drive the video. If KICK IT is a Public PC game, then we could claim the same at Atari with our Cyrix-based AREA 51: SITE 4 product, as could the folks at Midway with their Intel-based HYDRO THUNDER. Using off-the-shelf PC motherboards is a trend, not a standard.

To those still intent on tackling the brutal coin-op amusement device market armed with the Public PC, I offer these following words of advice.

MONEY-MAKER? You are now in the business of selling a machine that the customer will buy purely on the basis of its proven ability to earn back its cost in a 10- to 20-week timeframe. No customer of your product will hail your awesome, innovative idea if it does not allow them to make a profit. If another game earns more and costs less, then the operator will purchase that product instead of yours. Make sure your hardware is cheap (less than $800).

The operator won't pay extra for your game just because it's a new standard (this only happens in a techno-centric geekish dream). The commonly-touted speculation that there are sure to be other games from other developers will be a highly doubtful data point to a guy forking over $5,000 plus to buy your box. The "Universal Evergreen Cabinet" is a myth. The chances that two games from separate publishers will come out using the same hardware and controls are very small. To date, I am unaware of any coin-op developer ever releasing a title on anything other than their own hardware. (This might change with a true standard but we haven't seen it yet. Naomi, from Sega, has the best chance at this.) Additionally, technology moves so fast that the "standard" board this year will be outdated and non-competitive within 18 months. Your $5,000 game will have to justify itself with what it alone can earn.

ARCADE-SPECIFIC DESIGN. The idea commonly touted by the OAAF, that a developer can easily and profitably port their consumer title to the arcade, is false. Games that are financially successful in consumer markets generally do not perform well in the arcade environment. You need to redesign your game from the ground up to give the player approximately 90 seconds of intense fun per quarter. Compare the mechanics in a game like the PC-based DIABLO to our recent coin-op hit GAUNTLET LEGENDS and you will begin to understand the differences.

CRASHES. The machine you sell will have to be reliable. Not to make Mr. Gates mad, but it has yet to be proven that Windows 98 is robust or efficient enough to be an operating system for the arcade environment. Even if you overlook the ridiculous disk access times, having a coin-op game crash as often as a home PC is unacceptable to the arcade operator.

THIS ISN'T JUST SOFTWARE ANY MORE. Think software development costs are bad? You are now in the business of manufacturing a regulated device that must be reliable and cost effective, in addition to developing a game. Inventories of parts will cost an average of $1,500 to $3,000 a unit. You need to be ready to spend $1.5 to $3 million just to create the 1,000 units necessary to recoup your investment in development. You'll need to be UL and FCC approved. Your customers, (the distributor and the operators), will require customer support for these devices so that they can continue to earn money on a regular basis. While all these obstacles are not insurmountable, they are indicative of many of the

Continued on page 63.

Image by Laura Pool

For the past five and a half years, Mark Stephen Pierce has been senior vice president of coin-op product development/ executive producer at Atari Games in Milpitas, Calif. During his tenure as VP, he and his design group have delivered PRIMAL RAGE, T-MEK, a novelty game called HOOP IT UP, AREA 51, WAYNE GRETZKY 3D HOCKEY, SAN FRANCISCO RUSH, MAXIMUM FORCE, MACE, RUSH THE ROCK, CALIFORNIA SPEED, SITE 4, and GAUNTLET. Despite the tie, he is not a suit. You can reach him at pierce@agames.com.

Continued from page 64.
hidden aspects of being in the manufacturing business as opposed to being in the software business.

The Future

What will happen with the PC in the arcade industry? Will the X86 family of processors replace the current prevailing use of proprietary, embedded systems? Will a new standard be built around the readily available PC hardware open the coin-op market to new developers? Here's the opinion of a self-confirmed skeptic.

Coin-op amusement devices will increase the use of off-the-shelf PC motherboards and graphics display technology, coupled with their own custom I/O JAMMA-compatible boards. There are already games with off-the-shelf PC motherboards in them. The price/performance ratio of these motherboards and graphics cards at the lower end of the PC spectrum is too attractive for a manufacturer to pass up.

The chance of a global standard coming to the fore is slim. Because of the cost effectiveness of low-end, off-the-shelf PC motherboards, the use of these solutions will continue to increase. To avoid theft of their valuable intellectual property, these systems will have security features that prevent piracy, and probably hamper others in publishing software kits. This will effectively prevent the evolution of a truly open system.

Financial reality, not the hardware, is the real challenge in our business. Electronic Arts and Acclaim both realized this when they tried to enter our fold. They both spent about $20 to $50 million each trying to start arcade divisions. After a few years they both shut down these units with no intention of returning. Do not think your challenges will be much different.

So, if you want to make a coin-op game, forget about this yet-to-be-established, ill-defined, two-year-old standard, and go design a cost-effective device with a hardware solution that fits the needs of your game. Make sure it earns enough money on location tests. Your customers will only purchase your product because the return on investment justifies the purchase price. ■