gd

GAME DEVELOPER MAGAZINE

MAY 1999

UBM
Tech

3D Audio: The Sound of One Hand Clapping

According to experts, sound is more closely coupled to the emotional center of our brain than our visual sense is. At least that's what Dave Rossum, the chief scientist at E-mu Systems/Creative Labs, says. But I believe him — audio has a mysterious and indescribable way of affecting a person's mood.

There are many firms trying to bring better audio to game players, which I find very encouraging. This issue, Jonathan Blow reviews the SDKs from a number of these companies and presents his findings. These programming libraries help create interesting audio effects using techniques such as head-related transfer functions (HRTFs), interaural time delay (ITD), crosstalk cancellation, and reverberation, often in conjunction with specific sound cards.

On the Outside Looking In

But it's with mixed emotions that I look upon the state of 3D audio today. I've experienced spatialized audio effects while playing games under controlled conditions, and I think it's being oversold. Using two speakers, if I keep my head just so, I may be tricked momentarily into believing a sound is coming from a point 90 degrees off center to my left or right. Once in a while, if I'm wearing headphones, a game will fool me into believing a sound is almost directly behind me. In every case though, the effect is fleeting and not very impressive. The only time I truly believed a sound was coming from directly behind me was during a demonstration of Psygnosis's LANDER in a sound room at Dolby Labs. And these sounds coming from behind weren't really the result of trickery, they were produced by some expensive stereo speakers that were actually mounted on the wall behind me. *That* was amazing and immersive. But I couldn't credit HRTFs and ITDs in that case.

Better Sound from Games: A Fact-Finding Mission

Don't get me wrong. I understand the limitations inherent to HRTFs, ITD, and so on, and what they bring to gaming is far better than anything we've had in the past. However, it's important for this industry to be candid with consumers about what 3D audio delivers using the hardware that most users have (two speakers), because positional audio is not well understood by consumers. One well-known company states on its web site that its sound card "immerses you in heart-pounding, adrenaline-rushing PC gaming and entertainment audio with multidimensional sounds and awesome effects from every direction." Every direction, perhaps, if you have a 4.1 (as good as it gets on a computer) speaker system. But most players don't have that much audio hardware hooked up to their computers. This kind of hyperbole doesn't help consumers trying to make informed decisions about the technology. People complain about the confusing terminology on the packaging of graphics cards, but at least in those cases, a person can research the term "MIP-mapping" and get a straight answer. With audio, it seems to be more of a hype sell. HRTFs, ITD, and other audio concepts aren't even mentioned on most sound card packages.

For that reason, I have to applaud Creative Labs for using the term "environmental audio" (as used in their Environmental Audio Extensions, or EAX, technology), because I think that it's a much more candid way of marketing these features than the term "3D audio." Creative Labs stresses that environmental audio isn't necessarily audio coming "from every direction." Truth in advertising is key. ■

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

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*
Editorial Assistant
Jennifer Olsen *jolsen@mfi.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 e: *alanger@mfi.com* t: 415.905.2156
Eastern Regional Sales Manager/Recruitment
Ayrien Houchin e: *ahouchin@mfi.com* t: 415.905.2788
International Sales Representative
Breakout Marketing e: *breakout_mktg@compuserve.com*
t: +49 431 801703 f:+49 431 801797

ADVERTISING PRODUCTION

Senior 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/Circulation Jerry M. Okabe
Assistant Circulation Director Sara DeCarlo
Circulation Manager Stephanie Blake
Circulation Assistant Kausha Jackson-Crain
Newsstand Analyst Joyce Gorsuch

INTERNATIONAL LICENSING INFORMATION

Robert J. Abramson and Associates Inc.
t: 914.723.4700 f: 914.723.4722
e: *abramson@prodigy.comz*

Miller Freeman
A United News & Media publication

CEO/Miller Freeman Global Tony Tillin
Chairman/Miller Freeman Inc. Marshall W. Freeman
President Donald A. Pazour
Executive Vice Presidents H. Ted Bahr, Darrell Denny, Galen A. Poss, Regina Starr Ridley
Sr. Vice Presidents Annie Feldman, Howard I. Hauben, Wini D. Ragus, John Pearson, Andrew A. Mickus
Sr. Vice President/Development Solutions Group KoAnn Vikören
Group President/Division SF1 Regina Ridley

BPA INTERNATIONAL

www.gdmag.com

BIT Blasts

News from the World of Game Development

New Products: Nichimen's next-generation Mirai, Intel's GPT supports DirectX 6.1, and Renderware is middleware for the next Playstation. **p. 7**

Industry Watch: Sony announces the next generation, Rival Studios is born, Interplay sees red ink, THQ moves on up, and other news. **p.8**

Product Reviews: Jeffrey Abouaf dissects Digimation's Bones Pro 2. **pp. 10-12**

New Products

by Wesley Hall

Smooth Moves from Mirai

NICHIMEN GRAPHICS just launched the much-hyped Mirai, and is now second out of the gate (post-Maya) with its next-generation 3D animation package.

Mirai targets game developers and high-end character animators, and is the latest incarnation of Nichimen's N-World suite of real-time content creation tools. The system incorporates features such as subdivision surface modeling, 2D and 3D paint, skeletal modeling, inverse kinematics (IK), biomechanical motion editing, a nonlinear motion editing system, particle systems and physics simulations, and photorealistic rendering. Nichimen calls this a "3D operating system," and seems particularly proud of Mirai's linked 2D and 3D editors and other structural features which allow you to work in a more creative, nonlinear fashion. The company also touts Mirai's sophisticated skeletal system which automatically enables skeletal objects for IK movement as you build them. Additionally, you can create constraints with just a couple of mouse clicks, and the IK algorithms allow for natural human movement. Biomechanical motion editing tools help blend motion and smooth cycles. In short, Mirai sounds like a boon if you want to animate the heck out of 3D bipedal characters.

The package is available for a suggested retail price of $6,495 and includes online documentation, a texture library, and a motion capture library from House of Moves. For Windows NT, Silicon Graphics, and the SGI Visual Workstations.

■ Nichimen Graphics Inc.
Los Angeles, Calif.
(310) 577-0500
http://www.nichimen.com

UV editing, a modeling window, a 2D image, and a 3D paint window allow you to do nonlinear work in Nichimen's Mirai.

IPEAK Family Upgrade

INTEL recently announced that all seven tools in the IPEAK family have been updated to improve efficiency and to support more hardware devices.

The major news is Version 2.0 of the Graphics Performance Toolkit (GPT), which now supports DirectX 6.1. The GPT measures 3D hardware accelerator performance, analyzes and records application workload, and analyzes the interaction of graphics hardware and software to help you isolate weaknesses in your game and optimize your 3D graphics performance. Other tools currently available in the IPEAK suite include the IPEAK Storage Performance Toolkit, the Intel Power Management Analysis Tool (IPMAT), Intel DVD Qualification and Integration Kit (DQUIK), Intel I/O Subsystem Performance Monitor (I/O Mon), and the Intel 1394 Integration Toolkit.

The new 2.0 version of GPT is priced at $279. All Intel Developer Forum attendees receive a free evaluation of the tools.

■ Intel Corp.
Santa Clara, Calif.
(800) 889-4290
http://developer.intel.com/design/ipeak/

Renderware 3 for Next Playstation

CRITERION SOFTWARE announced in March that it will be providing Renderware, the company's independent 3D graphics engine, as middleware for the next-generation Playstation.

The latest version, Renderware 3, provides a lightweight framework with very fast default implementations for an array of 3D-related operations. RenderWare provides you with a 3D graphics API with a simple object-based interface consisting of a small number of object types and a rich set of associated functions. You can customise existing Renderware 3 plug-ins or build totally new custom plug-ins. Renderware 3 is very small (about 100K) so as not to limit system resources.

At press time, Renderware pricing, availability, and details of some specific Renderware 3 plug-ins for the next-generation PlayStation were not yet available.

■ Criterion Software Ltd.
Guildford, Surrey, UK
+44 1 483 406 200
http://www.csl.com/

Industry Watch

by Alex Dunne

SONY REVEALS NEW CONSOLE PLANS. Sony took the wraps off the Playstation's successor, announcing that the project (the "Next Generation Playstation") would use DVD-ROM discs, and would play current PSX games thanks to a new I/O processor from LSI Logic. The new processor uses a 32-bit core identical to the current Playstation system. Sony announced no firm pricing details. Another interesting revelation is that the new console will play DVD movies and music, which makes it a direct competitor to Nuon-based DVD players. (VM Labs probably isn't thrilled.) The console is scheduled to go on sale in Japan by March 2000.

RADICAL DEPARTURE. Radical Entertainment recently lost some of its developers when the developers behind Fox Interactive's ID4 and EA's ESPN X Games Pro Boarder left to form their own company, Rival Studios. Rival Studios founder Darrin Brown has signed with Boston-based Dotted Line Entertainment, which is currently talking with publishers about the developer's first original title. A deal announcement is expected by E3.

SIERRA REORGANIZES. In one fell swoop, Sierra closed many of its far-flung development studios and relocated their teams to Sierra's headquarters in Bellevue, Washington. The company indicated that the 11 different locations that housed its developers weren't optimal, and reeled a number of these development and marketing teams back into the headquarters. The affected studios include Yosemite Entertainment (Oakhurst, Calif.), Pyrotechnix (Cincinnati, Ohio), Synergistic (Renton, Wash.) and Books That Work (Palo Alto, Calif.). All relocations are expected to be completed by E3. Sierra subsidiaries Berkeley Systems, Impressions Software, Papyrus Design Group, and Dynamix were not affected by the move, but the reorganization did not leave them unscathed, either. The restructuring resulted in about 180 layoffs throughout the company, including 30 at Dynamix alone.

EIDOS SIGNS ELIXIR STUDIOS. Eidos Interactive announced a publishing agreement with London-based Elixir Studios. Eidos will publish Elixir's first three products. The first is due in 2000. Elixir Studios was formed in 1998 by managing director Demis Hassabis, the cocreator of Bullfrog's Theme Park.

GT DOES CE. Wizardworks, a subsidiary of GT Interactive, is releasing a new line of games for Windows CE-based handhelds, under the product line of Games to Go!. The first five titles were released in early March for about $20 each. Thomas Heymann, GT's new chairman and CEO, said, "With projected annual hardware sales of $7.5 billion by the year 2003, the mobile computer market is a huge industry in the making." Heymann, who succeeded Ron Chaimowitz at the helm of the company, used to be president of The Disney Store. Chaimowitz now heads up Onezero Media, a GT subsidiary.

INTERPLAY SEES RED. Interplay reported that its net revenues for 1998 amounted to $126.9 million, compared with $120.1 million for the prior year, and amounting to a net loss of $28.2 million. The company lost $16.6 million in the fourth quarter alone. Interplay head honcho Brian Fargo attributed the poor fourth quarter results to the company's inability to ship Messiah and EWJ 3D, the fact that Baldur's Gate shipped late, and higher customer returns than usual in the fourth quarter. CFO Jim Wilson said that Interplay just negotiated an expanded line of credit and is addressing problems by reducing headcount by almost 20 percent.

ID NABS DEVINE. Graeme Devine, the cofounder and CEO of Trilobyte, landed a spot at id Software as a designer. Devine produced titles such as The 7th Guest and The 11th Hour.

THQ DOES MORE WITH LESS. THQ announced its revenue increased 141 percent to $215.1 million for the year ending December 31, 1998. Profits for the year came to $23.2 million, versus $9.3 million for fiscal 1997. The company had an especially impressive fourth quarter, thanks in part to strong sales of WCW/NWO Revenge for the Nintendo 64. Interestingly, THQ's growth in 1998 was achieved with fewer titles shipped than in 1997.

Mad action shots from WCW/NWO Revenge, THQ's money-maker.

UPCOMING EVENTS CALENDAR

May 9-13, 1999

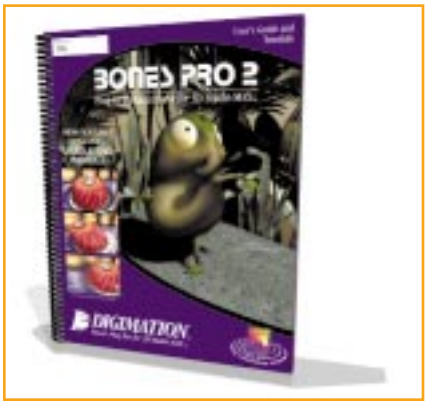
SD '99 West
Moscone Convention Cntr.
San Francisco, Calif.
Cost: variable
http://www.sdexpo.com

May 10-13, 1999

3D Design and Animation Conference
Santa Clara Convention Cntr.
Santa Clara, Calif.
Cost: Early Bird rates available
http://www.3dshow.com

May 12-15, 1999

E3 99: Electronic Entertainment Expo
Los Angeles Convention Cntr.
Los Angeles, Calif.
Cost: $400
http://www.e3expo.com



Digimation's Bones Pro 2

by Jeffrey Abouaf

Digimation's Bones Pro debuted in the days of 3D Studio DOS as the must-have kinematics chain-based deformation tool for character animation. Using Bones Pro is similar to using the boning systems found in most mid- to high-level 3D animation packages: the artist builds a hierarchical chain of bones scaled to fit and function as a skeleton within a character model composed of either a single mesh or multiple meshes. The mesh is then bound to the skeleton so that each bone exerts influence over a nearby section of vertices; animating the skeletal hierarchy deforms the mesh smoothly and naturally.

When 3D Studio Max 1.x and Character Studio 1.x appeared in 1996, Bones Pro Max expanded its function to remain a viable alternative to the native 3D Studio Max animation system and the Character Studio plug-in. Character Studio consisted of two parts: Biped, a two-legged inverse kinematic skeleton with its own footstep-driven animation system; and Physique, a mesh modifier for attaching and deforming the character mesh as the biped skeleton moved. Physique worked by assigning fixed vertex selection sets to specific bones. Animators critical of Character Studio 1.x had two principal complaints: freeform animation was more flexible than Biped's footstep-driven system, and the mesh-vertex selections were influenced only by single bones, not by and across multiple bones, resulting in unnatural movements and deformations.

Bones Pro Max was based on 3D Studio Max's space-warp technology, which meant that bones could influence the mesh in combination and proportionally, based on overlapping envelopes or bounding boxes radiating from each bone. All animation was freeform, and unlike Biped, the bones' size and scale could be animated (for instance, animating the scale of a chest bone makes the character appear to breathe). Complete control over movement, scale, and bone influence led many animators to use Bones Pro exclusively to animate their characters, and others to use it in combination with Character Studio 1.x (for example, using Character Studio for body movement and Bones Pro for facial animation). But last spring brought Character Studio 2; this long-awaited update addressed most of the limitations of version 1.x. The upgrade also incorporated many features formerly exclusive to Bones Pro: skeletal animation could be footstep-driven, freeform, or imported motion capture data, and could be converted from one form to another at will. In the new version, Physique defaults to envelope-based overlapping influence fields, which allows more than one bone to influence an area of skin and lets users assign weights for the influences. Character Studio 2's new features eclipsed those of Bones Pro Max in most respects.

Enter Bones Pro 2, with improved functionality and additional features: modifier-based application; support for multiple skeletal types, including Biped and Max bones; the capability to save, import, and export Bones Pro data (which is useful when more than one object is controlled by the same BP data); a more accurate bounding box influence calculation; support for forced vertex weighting; quick identification and assignment of unlinked vertices; and a new Bone Jiggler space warp for soft-body dynamics.

Bones Pro's tried and true utilities also bear mentioning (although these are the same utilities that shipped with Bones Pro 1): the Skeleton utility, for converting native 3D Studio Max bones into the box-based skeleton used by Bones Pro 2; Blend, a modifier for smoothing the intersection between attached or Boolean objects; and Snapshot Plus, a utility for creating stand-alone copies of a deformed mesh without the influence of any space warps (so you can essentially use a space warp as a modeling tool).

SKELETONS FROM BOXES AND BONES. In order to deform a mesh, Bones Pro 2 needs a skeleton in the form of geometry (regular geometric boxes linked together), a regular biped skeleton, a set of native 3D Studio Max bones, or a set of Max bones that have been converted to boxes using the Skeleton utility. If you're not using a biped, 3D Studio Max bones are the easiest way to create the skeleton, because they generate proper IK chains automatically and can be easily edited to fix distances between joints. If you use Bones Pro 2's Skeleton utility, it generates boxes around the bones and replaces the 3D Studio Max bones' IK controllers with Bézier and TCB controllers. The Skeleton utility will remove any end effectors (rotational or position) that you place on a native 3D Studio Max skeleton. However, once you've generated the skeleton boxes, you can set joint constraints just as you would with any other IK chain in 3D Studio Max. Figure 1 shows a skeleton created from 3D Studio Max bones (left) and the result after running the Skeleton utility. The boxes generated by the utility will retain some of the skeleton's information, such as the fact that they are hierarchically linked, the fact that they are the correct size, and the way in which multiple chains of 3D Studio Max bones are linked together (for example, the pelvis is root). But no joint constraints, terminators, or end effectors exist on the new box skeleton. You can resize (and animate the resizing of) any box-bone in the new skeleton to fit the character.

Alternatively, you can use 3D Studio Max bones directly to build the skeleton. Bones Pro 2 recognizes these better than did version 1, and if you take this approach, the bones remain subject to 3D Studio Max 2.x's IK controller. That is, you can place end effec-

Jeffrey Abouaf is an 3D artist and instructor. When he's not teaching or writing, it's rumored that he's conducting secret experiments on virtual humans. He can be reached at jabouaf@ogle.com and http://www.ogle.com.

tors, constrain joints, and set terminators using 3D Studio Max's most advanced IK abilities, and still enjoy the advanced capabilities of Bones Pro 2. The main difference between the approaches is how you might animate changes to bone scale: with geometry, you can animate x, y, z scale; with helpers, moving the bone extends or contracts it in relation to other bones, although you can't animate thickness. Either way works.

CHARACTER STUDIO WITH BONES PRO 2. Of great interest to animators is how well Bones Pro 2 integrates with Character Studio 2. You can use a biped as a skeleton for Bones Pro 2 (instead of Physique). Or you can link new bone boxes to the biped, animating the biped with Physique and the additional bones with Bones Pro 2. Or, you can apply Physique to all the bones, and then apply Bones Pro 2 to the same bones and the same mesh; Physique handles the underlying animation and Bones Pro 2 acts as a modifier on top of that. These different approaches show the flexibility of this upgrade. But the details of *when* and *why* you might use them together can be confusing, and sadly there's no documentation on point. For example, would you ever use Character Studio 2 and Bones Pro 2 together to animate an arm movement?

Clearly, integrating Bones Pro 2 with Character Studio 2 is elegant for adding facial animation to a biped character. This technique also compensates for some limitations still present in Biped/Physique, such as the inability to animate biped bone-scale to simulate breathing. You can apply the Bones Pro 2 modifier on top of Physique — a first — and animate using each of the applications up and down the stack.

Unlike Character Studio 2, Bones Pro 2 works with meshes, not NURBS or lattice-type space warps — a notable limitation. Applying Bones Pro 2 to a NURBS Bones Pro 2 object automatically converts it to a mesh; 3D Studio Max 2.x won't let you convert an editable mesh back to NURBS. You simply can't apply Bones Pro 2 to a lattice space warp.

WORKING WITH BONES PRO 2. Bones Pro 2 is a 3D Studio Max Object Space Modifier (OSM), meaning it's applied as a modifier to the target mesh object, not as a space warp, as was Bones Pro 1. This has advantages for cases in which you animate other modifiers in the stack. Within the Bones Pro 2 rollout, you designate the specific bones that you wish to affect the mesh (Figure 2). You then work in Bone and/or Vertex Sub-Object levels to visualize and adjust the influence of any bones on mesh vertices.

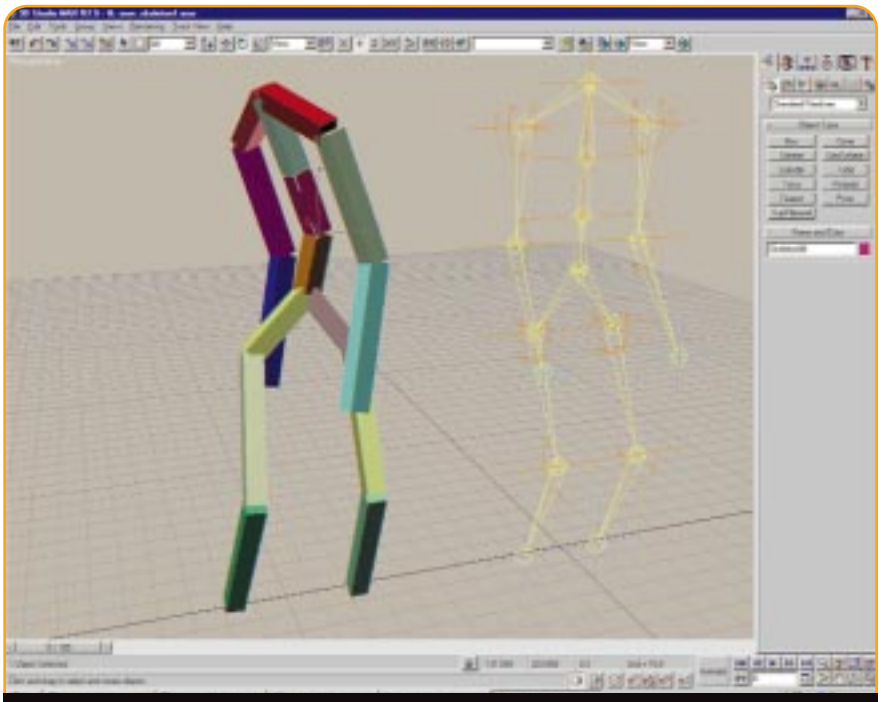


FIGURE 1. *Bones Pro 2 can make use of native 3D Studio Max bones (right), geometric boxes, a biped skeleton, or an IK chain of boxes (left) generated with the Skeleton utility.*

Bones Pro 2 displays relative influence strengths using color-coding. You can adjust Strength and Falloff in real-time and see the influence reflected as a change in color. In Bones Pro 1, relative influence was located in a pop-up window with its own set of controls. Bones Pro 2 is better organized: the viewer is in the viewport window, takes on the view of that viewport, and groups the relevant buttons in the command panel. This valuable visualization tool should be implemented throughout 3D Studio Max; now if we could only paint vertex weights, à la Maya's artisan….

The vertex Sub-Object level sports an improvement. While Bones Pro 1 used a bounding box or enveloping approach to set bone influence according to bone-vertex proximity, Bones Pro 2 lets you override this setting by typing in specific vertex weights and examining the effect. (Vertex weighting is also a feature in Physique 2.0, but there it appears intended for special-case rather than general usage.) Bones Pro 2 also lets you specifically identify any unassigned vertices and either include or exclude them as influenced by the selected bone. For anyone working with Bones Pro 1 or Physique, where it's common to see some vertices left behind when you first apply the modifier, these blanket inclusions and exclusions are a big help.

The Bone Jiggler feature will likely be Bones Pro 2's main selling point, because it introduces the appearance of soft-body dynamics (jiggles) as secondary mesh motions. I was able to create a short animation in which a character's belly jiggles as the character jumps. A special belly box-bone was added in the stomach and linked as child of the spine bone. I applied the Bone Jiggler space warp to the bone (not the mesh), and animated those settings (Figure 3) over 100 frames. You'll need to enable inertia and oscillation jiggling in the Bones Pro 2 modifier settings for jiggling to work. (These two settings enable an effect that is much like the motion created by the Hypermatter plug-in and, to a

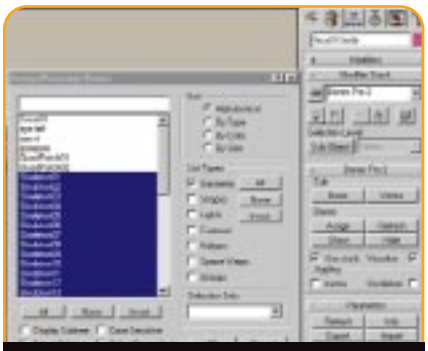


FIGURE 2. *Bones Pro 2 is applied to the target mesh as a 3D Studio Max modifier. You then assign the bones affecting that mesh.*

lesser degree, by the freeware Lag modifier. The advantage is that Bones Pro 2 is easier to use than Hypermatter, doesn't require a proxy object to be swapped for the original, and generally has been reworked to optimize usage in conjunction with Character Studio 2. And the Lag modifier lacks the extensive controls that are included here.) Six variables control jiggle movement, which can be constrained by axis. As motion, and not simulation, the Bone Jiggler works where visual accuracy will suffice (it requires less calculation than physical simulation).

Bones Pro 2 files can now be saved out in their own file format (.BPM), and the structure and animation data can be exported in .TXT format. These formats work well if you have several characters that need to be driven by the same skeleton. Bones Pro 2 makes full use of the modifier stack, meaning that you can use it for morphing objects, as with Smirk, Mix, FFD, or other modifiers. Or you might use the Snapshot Plus utility to generate morph targets. Either way, this approach is an improvement over the options that version 1 offered.

Bones Pro 2 offers new integration with Maxscript, which perhaps holds the greatest potential for this new version. Most functions can be called with their own script terms. As Maxscript plays an increasingly more significant role in production, the ability to write, for example, a slider-driven facial animation system using native 3D Studio Max bones and Bone Pro 2 is a great possibility.

WHAT'S LEFT FOR BONES PRO 2. However versatile, 3D Studio Max is criticized for its IK, especially when compared with Softimage, Maya, or Nichimen's achievements in this area. Specifically, I refer to IK handles, spline handles, or intelligent skeletons, in which multiple bones are controlled by one handle or a spline. These kinds of features save the animator time by constraining motions to the most natural movements. With Bones Pro, Digimation has an opportunity to bring advanced technology to this part of the 3D Studio Max environment.

My other wishes are comparatively modest. I'd like to apply Bones Pro 2 to NURBS without converting to a mesh, and it would be interesting to apply Bones Pro 2 to a lattice. Also, the documentation, which is good as far as it goes, could be more comprehensive in addressing how to get optimal results with Bones Pro 2 and other plug-ins, such as Character Studio, Smirk, and others. Bones Pro 2 has been improved to work with many other plug-ins — it would be helpful to have some dos and don'ts in this area.

OVERALL. Character Studio 2 stole much of Bones Pro 1's thunder, making it obsolete to all but facial animators and those who want total control in keyframing forward and inverse kinematic chains. Bones Pro 2 keeps the product alive. It makes native 3D Studio Max bones easier to work with; is easier to use than 3D Studio Max bones; allows great flexibility in determining bone influences and vertex weights; and integrates a usable, color-coded editor/viewer. The Bone Jiggler feature adds soft-body secondary

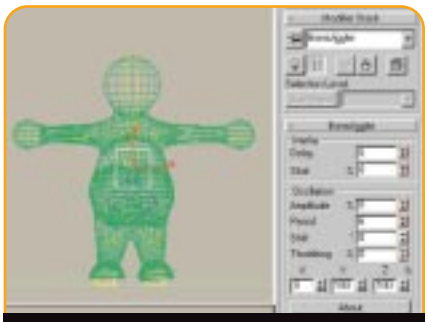


FIGURE 3. *The Bone Jiggler space warp includes six animatable characteristics, which can be constrained to the x, y, and/or z axes, to produce a very specific motion.*

dynamics, a natural extension for this type of product. The Blend, Skeleton, and Snapshot Plus utilities, while not new, are appropriate complements, and keep the product useful.

Overall, the improvements in Bones Pro 2 — in feature set, usability, and stability — make it beneficial to animators. The $395 price appears appropriate in the context of other 3D Studio Max plug-ins. The fact that Digimation developed this product in-house over several years is a plus — tech support has been good and maintenance releases have been issued on a regular basis. In November 1998, Kinetix announced a strategic partnership with Digimation Inc. to provide Kinetix third-party developers with full software publishing services, including packaging, documentation, quality assurance, marketing, and support. This product is useful and easy to use; I look forward to its continued development. ■

Bones Pro 2: ★★★★★

Company: Digimation Inc.
St. Rose, La.
(800) 854-4496
(504) 468-7898
http://www.digimation.com

Price: $395.00

System Requirements: Bones Pro will run on any system capable of running 3D Studio Max, such as a Pentium, Pentium Pro, or Pentium II with 64MB RAM (128MB recommended).

Pros:
1. Supports many skeleton types and makes it easy to edit or include/exclude vertices from bone influences.
2. Bone Jiggler is useful because it precludes having to use more complex plug-ins for soft-body effects.
3. Support for Maxscript.

Cons:
1. Could use advance IK manipulators, such as IK or spline handles to drive skeletal movement.
2. No support for NURBS or lattices.
3. Documentation could be extended to address the nuances of using Bones Pro 2 with other Max plug-ins.

Devil in the Blue-Faceted Dress: Real-Time Cloth Animation

I've been describing methods of dynamic simulation using mass and spring systems for the past couple of months. These techniques dramatically increase the realism in your real-time graphic simulation. One of dynamic simulation's key benefits is that it creates a scaleable game experience.

Users with more powerful systems get a more realistic experience, while users with less powerful systems are still provided with a complete experience. It's a situation analogous to the use of levels of detail in your 3D models. Particularly in the PC market, where target systems can vary widely, these techniques have become a crucial weapon in the developer's arsenal.

For a current project, I decided to maximize the use of dynamics to increase realism wherever possible. The project focuses on characters in moody interior environments. It occurred to me that the use of cloth animation in my scenes would be crucial to creating the mood I was trying to establish.

Traditional Cloth Animation in Games

Cloth animation is tricky. Even in the world of high-end computer graphics, it's difficult to get right. Most of the time, it's wise to avoid the whole issue. Anyone who has ever created a female character in a skirt is familiar with the problem of the legs poking through the cloth mesh during animation. This is pretty difficult to fix, especially if animation requires a variety of motions. It's particularly tricky if you are applying motion capture data to a character. Unfortunately, it's also a really obvious animation problem that any end user can spot. These cloth animation problems are the reason why most digital characters are clothed in tight-fitting gear, such as skin hugging stretch pants.

Most loose clothing doesn't look natural in digital art because it's static. It doesn't move along with the body. It's possible to morph the shape of the skirt to match the motion of the character, but this requires quite a bit of detailed animation work. Likewise, deforming the skirt with a bone system can be effective, but not necessarily realistic.

For my work, I wanted to create realistic cloth in the environments and on the characters. My hardware accelerated graphics rasterization freed the processor power necessary to make this possible. So, I set about creating a real-time cloth simulation.

The Latest "Springy" Fashions

The mass and spring dynamics simulation I developed in a recent column ("Collision Response: Bouncy, Trouncy, Fun" March 1999) proved effective for simulating soft body objects in real time. I thought it should be possible to use these techniques to create a cloth simulation. In fact, several of the commercial cloth animation systems for 3D animation programs such as 3D Studio Max, Softimage, and Maya use similar techniques. So how do I go about creating a piece of cloth?

I am going to be using the same spring force formulas for the cloth simulation as the ones I used in the March column. If you are unfamiliar with the dynamic forces generated by springs, you should go back and read the March column or at least take a look at the March source code on the *Game Developer* web site (http://www.gdmag.com).

I start by creating a rectangular grid, and then connect each point to neighboring points with springs, as you can see in Figure 1A. These springs define the rough structure of the cloth and so I refer to them as structural springs.

The devil wears an animated-cloth blue dress.

Jeff Lander prefers to wear comfortable loungewear when hanging out writing code at Darwin 3D. Drop him a note and let him know what the fashion conscious are wearing this spring at jeffl@darwin3d.com.

The resulting cloth patch looks pretty good and requires few springs. However, once I run the simulation, problems appear immediately as shown in Figure 1B.

The simple spring connections are not enough to force the grid to hold its shape. Much like the box in the March column, there are simply no springs to maintain the shape. If I held on to only one point, the entire surface would collapse into a single line creating a rope. Not exactly what I wanted, but it points out something I want to address a bit later.

I really want to keep the model from shearing too much. That is, I want the space between diagonal elements of the model preserved. So, I just add a few more springs to the grid along the diagonals creating a group of shear springs, as you can see in Figure 2A. Run this new structure through the simulation and the results are much better, as you see in Figure 2B.

This new form of cloth works pretty well hanging from hooks on the wall. However, if you drop the cloth on the floor, it wads up into a big mass of springy spaghetti. The reason for this failure is that the model is still incomplete. If you look at the structure in Figure 2A, you may see that there is nothing to keep the model from folding along the edges of the structural springs, much as you fold a handkerchief. The fibers that comprise actual cloth run the length of the fabric and generally resist folding and bending. In order to simulate this effect adequately, I need to do a little more work.

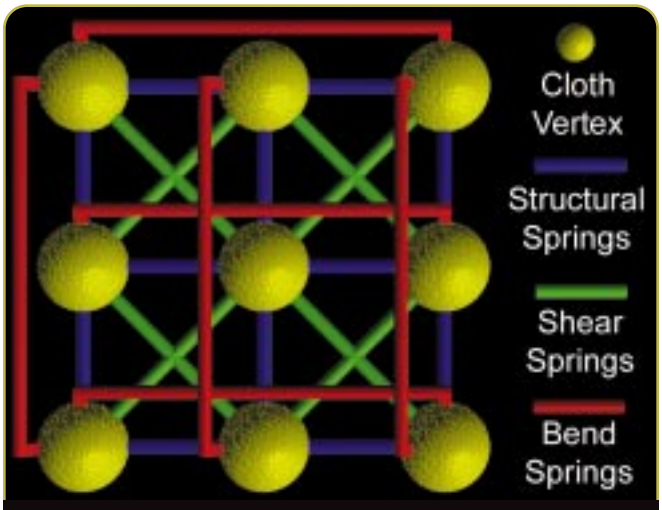


FIGURE 3. *The interconnection of cloth springs.*

My research uncovered two methods for dealing with this problem. The first minimized the bend between two adjacent cells by using the dot product to determine the angle of bend. The second method simply added an extra set of springs called flexion or bend springs to apply the bend force. I created the bend springs by stretching a spring across two cells alongside the structural springs. These springs end up connecting every other cell in the cloth mesh.

I prefer the second method because it works within the existing spring system without the need for a new method for calculating forces. I also get the benefit of having only one calculation to optimize later. For other applications, it's possible that the angle minimization method may work out better.

I now have a sufficient spring network to simulate a variety of different types of cloth. You can see how all the springs are connected in Figure 3.

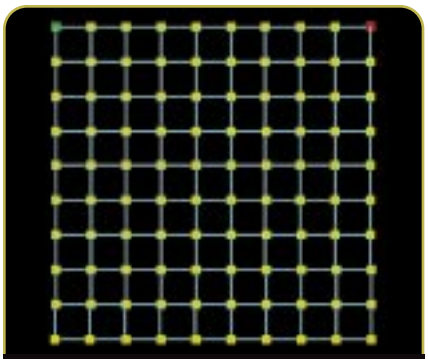


FIGURE 1A. *A simple cloth grid.*

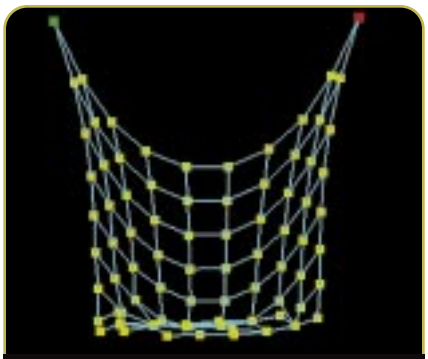


FIGURE 1B. *Stretched cloth grid.*

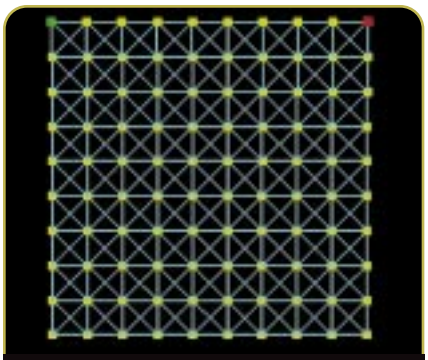


FIGURE 2A. *Added shear springs.*

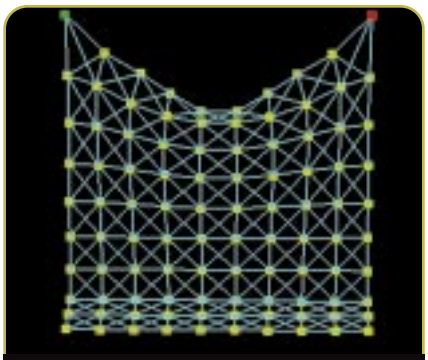


FIGURE 2B. *A better cloth model.*

Stretch without Tearing

These three types of springs make it possible to simulate a variety of different cloth types. By varying the stiffness of the springs, it's possible to simulate anything from stiff cardstock to stretchy nylon. For example, spandex would have very flexible structural and shear springs to allow strong stretching capability. Paper, on the other hand, is very resistant to shearing and stretching, so its springs would be very stiff. When considering how much a material will bend, a surface such as cardboard should have the very stiff bend springs to make it resistant to folding. I find experimenting with different values for spring stiffness the only real way to find adequate surface properties.

Stiff springs can make a numerical simulation unstable. To combat this,

FIGURE 4. *Collision boxes allow for the draping effect on this tablecloth.*

it's important to use a good numerical integrator. The midpoint method and Runge-Kutta integrators developed last month seem to do the trick nicely.

Making It Move

I already have a simulator from the March column that is capable of handling a cloth patch. I can even apply gravity to it and lock the position of individual vertices. That's pretty interesting, but it needs some improvement to come alive. In March, I also discussed the use of planes for collision. With this same method, I can create collision boxes that enable me to simulate a tablecloth draped over a table, as you see in Figure 4.

This model is interesting and realistic looking but not terribly animated. In fact, in this case it's probably better to freeze the simulation and avoid the constant recalculation. Unless, of course, the wind kicks up or someone pulls on the corner.

For characters, a moving box is not the most realistic way to displace the cloth. Moving bounding spheres allow much more pleasing character animation. Fortunately, this is easy to add to the simulation. Determining whether a point is inside a sphere is very easy. If the distance from the point to the center of the sphere is less than the radius of the sphere, the point is on the inside. If a point in the cloth is found inside a sphere, I have a penetrating collision. Just like handling collisions in the March simulator, I need to back up the simulation time to find the actual point of contact. In a sphere, contact takes place when the distance of the point to the sphere's center is equal to the radius of the sphere. Now that the contact point has been established, I need to resolve the collision. The collision normal, *N*, between a point and a sphere is the vector between the point of contact and the center of the sphere. You can see this in Figure 5. Fortunately for me, the rest of the collision response is handled just like the collision with a plane. This means my existing collision response code works great.

I can now add collision spheres to my simulation. The cloth slides realistically off the spheres. You can see how the simulation looks with two collision spheres in Figure 6. By animating the spheres along with the 3D model, I can get a nice animated hip sway and other alluring effects. The motion of the cloth continues after the animation stops, creating entertaining effects that are diffiicult to achieve with traditional animation techniques.

Problems to Avoid and Ignore

The simulation has a couple of problems. The first is that the way to simulate cloth realistically is to use a lot of points in the simulation. This takes more computation time. High-end animation programs rely on a great number of particle points for realism. Of course, in other fields, hour-long render times are perfectly acceptable. In a real-time game, however, this won't get you on the cover of any game magazines. You have to sacrifice realism for speed. This is another good area for scaling game performance. If the system is running

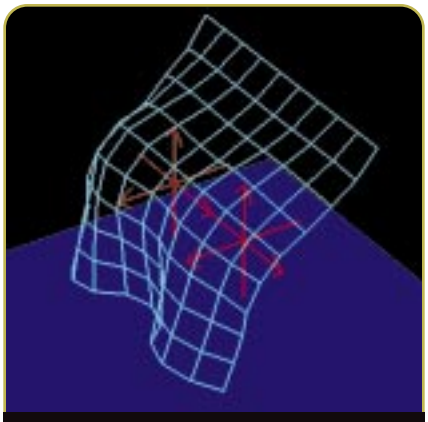
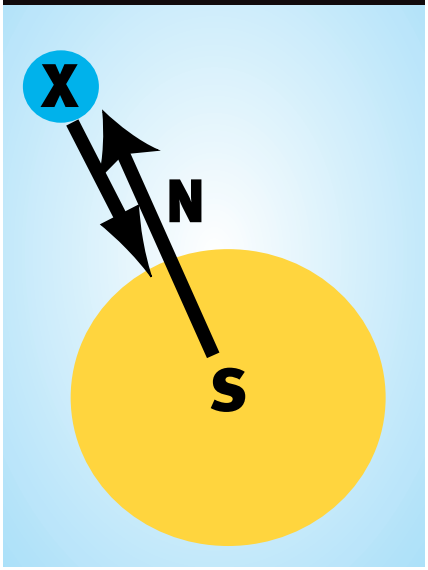


FIGURE 6. *Cloth hanging from a pair of invisible hips.*

FIGURE 5. *Collision with a sphere.*



quite fast, subdivide the cloth patches a little more. Game players with a white-hot system should have smooth-looking cloth.

Another problem is that each spring acts independently. This means that each spring can be stretched to a great extent. In many cases, the amount of stretch can exceed 100 percent. This is not very realistic. Actual fabric will not stretch in this manner. The problem I have is that I am using linear springs when fabric actually displays a nonlinear spring behavior. As the amount of stretch increases, the strength of the spring increases also. The fabric will also stretch to some limit and then if the force continues, it will rip. This is not what I want (at least for now). This issue, which Xavier Provot (see For Further Info) calls "the Super-Elastic Effect," is difficult to handle. Increasing the spring strength dynamically can lead to instability problems just like any other stiff spring problem. Provot suggests checking the amount of stretch in each spring, and if it exceeds a set deformation limit, the springs are adjusted to achieve this limit. While I agree this solves a definite problem, a second pass through the springs is costly. For the effects I have attempted to achieve, I can live with super-elastic cloth.

My collision system is pretty primitive. To make things easy, I only collide the vertices of the mesh with the objects. As it stands, if a sphere is small or the fabric stretches too much,

the sphere will pass right through it. I also don't handle self-collisions. That is, the fabric can pass through itself without penalty. This could be corrected by placing bounding spheres at each vertex. However, applying the sphere collision test between each vertex gets expensive. So, I just limit the situation so that either the cloth doesn't pass through itself, or so the effect isn't too noticeable.

Taking It to the Limit

Once the system is working, it's fun to see how it can be extended. I mentioned the issue of tearing and ripping after the fabric stretches too far. I can monitor the spring lengths. If they exceed a limit, the spring can be removed from the system, effectively tearing the fabric. I think this would be a great way to simulate a cannonball tearing through the mainsail of a tall ship. This same method of breaking a spring would work for a simulation of a rope as well. After all, a rope is really just a one-dimensional version of the cloth patch.

Another dynamic effect can be achieved by manipulating the flexion springs. With these springs in place, the fabric will resist folding. However, if I selectively delete one of these springs, the fabric will be able to fold nicely where the springs are missing. I don't know where I can use that yet, but I'm sure I can find a way.

ACKNOWLEDGEMENTS

Special thanks to Chris Hecker of Definition 6 and Rob Wyatt of Dreamworks SKG for discussing the issue with me.

FOR FURTHER INFO

• Provot, Xavier. "Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior," *Graphics Interface*, 1995, pp. 147-155. Also available in electronic form at http://www-rocq.inria.fr/syntim/research/provot

• Baraff, David and Andrew Witkin. "Large Steps in Cloth Simulation," *Proceedings of SIGGRAPH 1998*, ACM SIGGRAPH, pp. 43-54.

There are also fabric simulations available for many professional 3D animation packages available either as plug-ins or integrated into the software. I do not know what techniques these products use with the exception of one. Colin Withers of Topix created a fabric simulation for Softimage based on the Provot paper. Graciously, Topix released the source code for this plug-in to the public. See http://www.topix.com for more info.

The Application

The application this month was actually pretty easy to build. It's essentially the same as last month's application, but with a few additions. There's a function that creates the cloth patch in a sort of macro fashion. You can set the spring settings for the three types of springs. You can also drop some collision objects around and watch them interact. Find the application and the source at http://www.gdmag.com. ■

See Jane Walk

See Jane run. See Jane do a triple spin kick with a half twist, drop to a crouch, draw her 9mm, and waste three zombies.... If we took a cross section of the characters populating today's game environments, we'd end up with an extremely rich and diverse cast.

They run the gamut from multilegged crab-like villains to spiky-haired, sword-toting heroes. Today's character artists have been working overtime to bring us some of the most awe-inspiring and nightmare spawning characters ever seen on a video display. To spring fully to life, these characters need to move convincingly and with, well, *character*.

Back in my October column, ("It's About Character") we took the characters from pencil sketch to fully textured model. In this month's feature, we'll look at the final step in the process required to bring these actors to life — character animation. We'll discuss some of the different tools and techniques available to today's character animator, provide some general tips, and point out some of the pitfalls to avoid. Finally, we'll go on a field trip to House of Moves, the West Coast's premier motion capture studio, to look at the advantages and disadvantages of using motion capture in a production environment.

There are several methods currently in use for animating characters, each with its own particular problems and advantages. Identifying the right method early on will save you time and money. Choosing the wrong one can cost you both. All of these methods can be broken down into four basic categories: classical, rotoscoping, motion capture, and procedural animation (we won't cover the last one this month).

Classical Animation

Classical animation, or animating by hand, is by far the most common method artists use today. For the purposes of our discussion, classical animation includes any and all forms of hand-generated motion, and uses any of the variety of skeletal animation tools available. Classical techniques require the least amount of preparation, and are supported by almost every tool set. Most production houses prefer to use this method of animation due to these lax technological restrictions. The experience base for this technique is the largest, and reference materials abound. Still, skilled character animators are increasingly hard to find, and the classical method depends solely on the skills and talent of the animator to bring characters to life.

Design documents may also dictate classical animation as a production technique. With nonbipedal characters, for example, your other options are pretty limited. Both motion capture and rotoscoping depend on having a real-world analog in order to work. Furthermore, an extremely stylized look may dictate a nonrealistic style of motion. You wouldn't expect to see motion capture used for SUPER MARIO or CRASH BANDICOOT. Results may vary, but it's possible to achieve motion-capture-like animation by hand if the animator is skilled and spends sufficient time on the motion.

TIPS AND TECHNIQUES. Classical animation depends almost entirely on the creativity of the animator involved. Because most if not all of the animation comes directly from the animator's head, the motions will be an interpretation of things that the animator has either seen or experienced. Having an extensive library of books, videos, and other references can serve to speed the process and raise the level of quality of classical animation. Figure 1 shows an

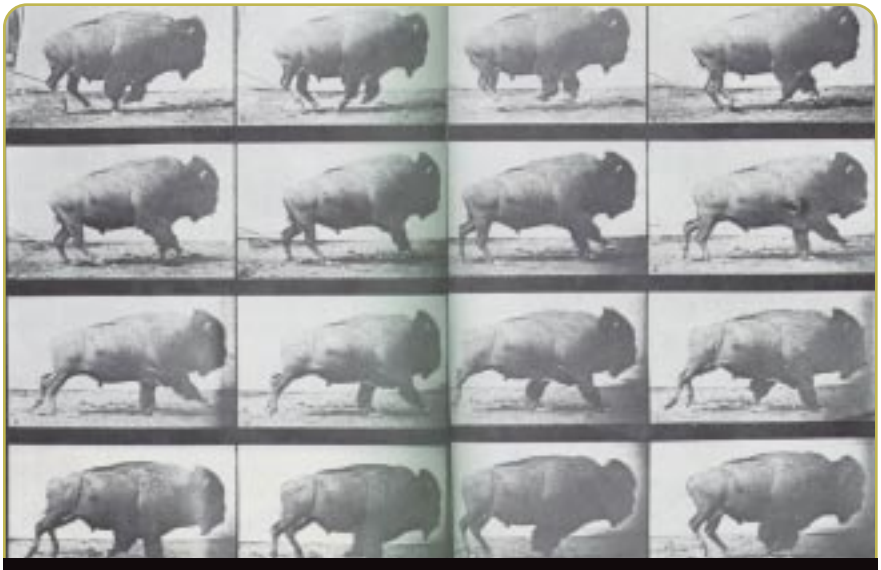


FIGURE 1. *Running sequence from Muybridge's classic* Animals in Motion.

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.*

excerpt from Muybridge's *Animals in Motion*. An excellent reference, the two-book series provides an in-depth treatment of some of the more common styles of motion for humans and selected animals. Studying an actual real-world animation on film can be just as helpful. For example, looking at a Jackie Chan video frame by frame can provide you with valuable insight into how a martial arts move is performed. For animal reference, check out the local video store for the National Geographic documentary series, which covers almost every species and body-style imaginable.

COMMON PITFALLS. As with most things in life, in animation you can't get something for nothing. Classical animation depends entirely on the skill set and efforts of the animator, and the result is totally scalable. The more time you spend adding subtle nuances to an animation, the better it's going to look. Good animation takes time. Sure, you can do a run cycle that works with four keyframes, but don't expect it to look anything like motion capture.

Every animator has his or her own style of working. Consequently, the results will vary slightly between animators. This can cause problems if you have several animators on a team working on similar character sets, because achieving a uniformity of motion can be problematic. Consider having one animator work on all the bipeds, one on all the four-legged types, and so on (or one work on all the male characters, another on all the females, and so on). Probably the worst thing you can do is split up the work for a single character between multiple animators. If splitting a character among animators is absolutely necessary, consider putting one animator on all the combat moves and the other on all the standard interactive ones.

Rotoscoping

Although not as widely used as other methods, rotoscoping nevertheless provides near-motion-capture realism at a fraction of the cost. Somewhere between motion capture and classical animation, the methodology of rotoscoping is fairly straightforward. First, you take video footage of an actor performing a series of moves.

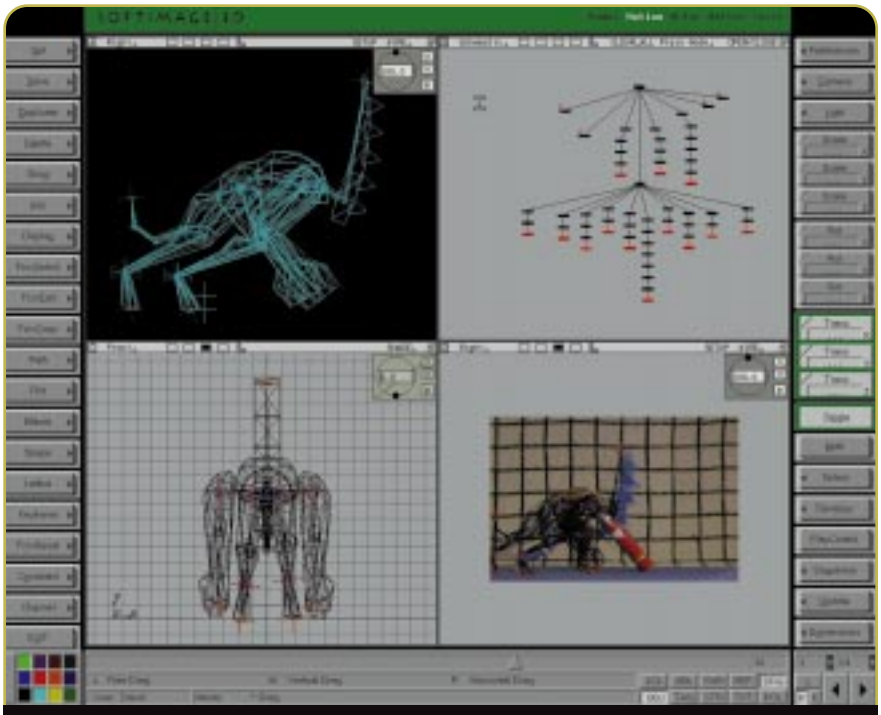


FIGURE 2. *Rotoscoping in action with Ripcord's FLESH & WIRE.*

Then you transfer the film to a digital format and bring it into the animation software, frame by frame, as a backdrop over which the skeleton is animated. The technique requires a significant amount of preparation, and the technical restrictions on the sequence require that the camera angle of the backdrop be similar to that used for animation.

When compared to classical animation, rotoscoping typically cuts down the time required to achieve the animations. Furthermore, because of the visual aid, the results can be very true to life. Finally, with the added flexibility of keyframing by hand, the animator can choose to use only part of the rotoscoped scene as a guide, and animate the rest of the sequence with other methods. Thus, both rotoscoped and classical animation can be mixed together on a single character. If pickup animations need to be added by hand at some point, the difference from the rotoscoped animations will be indistinguishable to the average player.

Despite its advantages, choosing to use rotoscoping can be a hurdle for some studios because expertise in this method is very limited. The technique has its limitations, too, because the footage you can use is limited to real-world animation. Furthermore, because all of the keyframes are actually being set by hand, the rotoscoped data only serves as a crutch and the animator's skill set still needs to be fairly high.

Figure 2 shows a rotoscoping scene in Softimage taken from Ripcord's FLESH & WIRE (see my March 1999 column for a full description). In this example, a Softimage skeleton has been overlaid onto a previously captured image (bottom right quadrant). Note that although the skeleton is not quite humanoid, the animators have been able to outfit an actor with a suit that resembles the character's anatomy. In this way, when they're animating, they can approximate the resultant motion with a high degree of accuracy.

TIPS AND TECHNIQUES. Rotoscoping's big advantage is that it can be done totally in-house with a single video camera and a darkened office. The sets can be extremely primitive. As we saw in the previous example, some sort of reference grid is useful for the animator. A very low-tech, large sheet covered with duct tape strips will serve this purpose.

Using an actor can help as well, and you need only go down to the nearest university or high school drama department to find a few starving students willing to do the work. If you do use actors, try to keep the same actor playing the same character. If the software you're using doesn't support real-

CLASSICAL	ROTOSCOPING	MOTION CAPTURE
Advantages: •Extremely flexible; animations can be created on-the-fly, in-house •Character diversity; no restrictions on character type or movement style •Universal tool support •Can be cost effective	**Advantages** •Faster than classical animation •Realistic, predictable motion •Ability to blend motion between rotoscoping and classical animation •Relatively low cost •Lower animation skill set required vs. classical animation	**Advantages** •Extremely fast •Can be cost effective •Versatility; some otherwise complex moves are made simple •Realistic, true-to-life motion •In-house animation skill set requirement is minimal compared to other two methods
Disadvantages: •Time intensive: animating by hand takes the longest •Higher personnel requirements: needs a highly skilled animator on staff •Animation uniformity: multiple animators can mean multiple styles.	**Disadvantages** •Significant time commitment; you're still animating by hand •Rotoscoping shots are limited to real world animation •Setup time and porting images into animation software.	**Disadvantages** •Preparation time is essential •Difficulties involved in mixing methods; motion capture data stands out •What you see is what you get; changing the data afterwards is difficult.

time animated backgrounds, try assigning a series of images as a texture map to a dummy object which you can then use as a backdrop. (Remember to keep the aspect ratio of your backdrop object the same size as your image, otherwise you'll get some nasty distortions.)

COMMON PITFALLS. Here again, the animator's effort and skill level can make all the difference. Rotoscoping is really only one step up from pure classical animations. So, the formula for success is the same: the more time spent, the higher quality level of the animation. Work hard to plan out the animations before shooting the footage. Paging through hundreds of megabytes of unnecessary video footage wastes the time of everybody involved.

Motion Capture

Until they're exposed to it for the first time, most people think that motion capture is just an expensive technique reserved for movies and commercials. Yet, several successful franchises owe their success to this technology, and several major development houses use it almost exclusively for their projects. To get the scoop on motion capture, we visited the Los Angeles-based House of Moves (http://www.moves.com), the premier motion capture studio on the West Coast.

THE MOTION CAPTURE PROCESS. For those still not familiar with the process, motion capture is a method by which the motion of an actor is digitized through a series of sensors attached to the actor's body. The capture process is extremely fast, and House of Moves (HOM) is able to capture more than 150 moves in a single day. The motion capture staff then sets to work cleaning up any noise generated in the capture process. The next step is to attach the data to a skeleton. This can either be done in-house, or by the large staff of animators working at HOM. Figure 3 shows an example of a motion capture skeleton being worked on by the company. The small circles represent the markers attached to the actor's body. By the time you're done with a motion capture session, you end up with a set of animation sequences no different from those generated by your in-house animation staff, but with one major exception — the animations have the fidelity of motion and nuance of form recognizable only in motion capture.

Compared to the previous two methods, the motion capture process is extremely fast. An entire set of animations for a game can be generated in just a few months. Consider that for a project calling for only 200 animations, an average animator capable of churning out 5 to 10 high-quality animations per week would take anywhere between 5 and 10 months to get the same work done. If you're aiming for a realistic look, nothing looks better than motion capture.

For this reason, the motion capture process is also cost efficient. If the same animator in the above example is being paid an annual salary of $50K, then the cost to the project to generate the same 200 animations by hand is anywhere from $20K to $40K. This exceeds the cost of most motion capture shoots.

Despite these advantages, the decision to use motion capture is not one to be undertaken lightly. Many horror stories ciruculate about a budding developer who spent thousands of dollars on motion capture data that was never used

FIGURE 3. *Motion capture skeleton set up at House of Moves.*

FIGURE 4. *Motion-captured scene from* PARASITE EVE.

FIGURE 5. *Elaborate stunt set up at House of Moves.*

in production. Although it's an extremely useful technique, motion capture is not applicable to every situation, so knowing when it is applicable can save time and money for everyone involved.

TIPS AND TECHNIQUES. You should choose motion capture if your game contains humanoid or bipedal characters; your game engine supports animation data at a high frequency (animation fidelity pared down to 5 to 10 FPS loses most of the look of the motion capture data); you have a solid understanding of what animations you will need so you won't be changing the data after you get it; or your game requires complex animations that would be difficult to achieve through other methods.

You should not choose motion capture if your game contains no bipedal characters; your game design is not completely solid (and you don't know what animations you will need); or your game design calls for non-realistic, stylized motion.

Even with these general guidelines, there are some other specific times when motion capture really beats all contenders. Most fighting games — the TEKKEN series, for example — owe much of their success to the lightning quick, realistic motion of their characters. This is something that is extremely hard to animate by hand. Sports titles are another genre ideally suited to motion capture. Finally, any military-style, or close-order combat scenario is a good candidate for motion capture.

Most of the problems with motion capture occur because the developers don't realize the amount of preparation involved in doing a successful shoot. Preparation is the key to any successful endeavor, but with motion capture it is absolutely critical. (See Melianthe Kines's *Game Developer* articles, "Planning a Motion Capture Shoot," September 1998 and "Directing a Motion Capture Shoot," October 1998).

Plan to spend a few weeks outlining every animation on the list to be captured. When you go to do the shoot, you are going to be given what you ask for — if you don't know what you want, prepare to be surprised by the result.

Use actors. Hiring an actor is a fraction of the total motion capture cost, and the correct body style can make all the difference to how the motion looks. For example, if your main character is a petite female martial arts expert, you don't want to capture a bulky computer programmer. Spend time interviewing actors to use for the capture shot, or work with the studio to find the right fit. Once you've settled on an actor or set of actors, take a few days to go over each animation in detail with the actor. Doing this will ensure that the actor gives you what you want, and you may find that after seeing the performance, you want to adjust it slightly or change it all together.

Once you've received the data, be prepared to use it *as is* with only slight modifications. Greatly modifying the animations will prove to be tedious and time-consuming, and will often remove the subtle nuances which give motion capture its unique look. And while there are many good reasons to choose motion capture, the overwhelming one is it's unparalled realism of motion.

Wrap-Up

At the end of the day, the choice is yours. Horror stories abound about teams that have jumped the gun with one method or another. Spend the extra time to determine which method is best for your game. The money and time dividends you earn will pay off in an enjoyable development experience. ■

Dolby and Aureal: Contrasts in Audio

If you sliced up a PC's architecture and ranked the components most relevant to game developers, you'd end up looking first at the raw CPU power and bus performance. Graphics might come next. Unfortunately, audio would probably end up near the bottom of the list.

Despite audio's stepchild status, development teams are recognizing the importance of improving quality in *every* technology sector in order to provide a well-rounded game-playing experience. In the audio section, developers are adding realism in the form of 3D audio, and are considering the quality of audio in light of the growing interest in digital audio technologies. The new 3D audio capabilities and the rise in audio quality expectations creates a market with interesting dynamics. Nowhere is this dynamic more interesting to observe than in the contrasts between Dolby Laboratories and Aureal Semiconductor. Both companies are similar in their desire to bring their respective audio technologies to bear on the PC games market, but their different approaches and products point toward increasing conflicts (and synergies) in this oft-neglected side of the development business.

Complementary Differences

Dolby's background lies in the film, television, and recording industries. The company's noise reduction and surround sound technology is readily found in the consumer electronics market. Most of us are familiar with the ubiquitous Dolby logo on our audio systems and VCRs.

In the case of Dolby surround sound technologies, the area in which Dolby brings most to the PC, the technology is essentially transparent. A normal stereo's capabilities may be realized by the consumer via a simple playback system upgrade. The Dolby Digital AC-3 technology (upon which the company bases its surround sound license) allows for a number of channels or speakers through bandwidth reduction, and thereby makes multichannel audio readily deployable. But this is not necessarily an interactive technology. It does not allow interactive positioning of sound, and the developer must pre-encode positional audio for playback. This is all well and good at the movies, but it may not be what the average game developer wants to do. This playback advantage is pretty consumer friendly, however, and doesn't require a great deal of initiation on the part of the user.

Aureal approaches surround sound with the interactive in mind. The company's focus on interactivity stems from Aureal's concentration the desktop computer market. So, while Aureal provides technology for multichannel audio, it's designed for two-speaker playback. Aureal does support four speaker configurations, but this isn't a significant part of the company's strategy. In fact, Aureal is a Dolby licensee too. Like many PC companies before them, Aureal crosses paths with Dolby at only one junction: Dolby Digital audio is one of the approved DVD-Video and DVD-ROM formats.

Toni Schneider, Aureal's vice president of advanced audio technology says, "Dolby is the master of soundtracks and prerecorded (as in, non-interactive) audio. Because games are an interactive medium, Dolby's technologies have had less of an impact on gaming than they did on movies. We consider our A3D interactive 3D technology complementary to something like Dolby ProLogic or Dolby Digital."

The Business of Technology

So, in many ways, Aureal and Dolby are standing at different ends of the audio spectrum. This separation becomes even more obvious when we examine the ways in which the two make money.

Tom White of the MIDI Manufacturers' Association (a respected audio analyst who provided excellent background information for this article) says, "Dolby is a licensing company, with a strong existing business in the recording, film, and broadcast industries. It licenses proprietary technology which assists in reducing noise, reducing bandwidth requirements, and producing multichannel surround sound (AC-3). Its customers are hardware and software vendors. Dolby makes nothing and sells nothing tangible. It is privately owned and likes it that way.

"Aureal, in contrast, is a public company that has been fueled by outside

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.

investment and has only recently started to earn significant revenue due to increased demand for their ICs. Of course, the other 3D audio companies have also been suffering losses as they all invest heavily to create a market. But I think it is apparent that Aureal expects to make the bulk of their revenue from chip sales, not licensing."

This means that Aureal's business model is completely opposite Dolby's. The audio chip business is volatile, and Aureal's market success is primarily attributable to developer and brand recognition of the A3D API and technology (not its Vortex hardware, which is used on Diamond's Monster Sound audio cards). Nevertheless, Aureal recently announced that the company and its licensees had completed shipments of more than five million PCI audio products enabled by Aureal's A3D positional audio standard. So, a mixture of hardware and licensing is creating momentum for Aureal. Dolby is, obviously, a market force in its own right.

The Standards Minefield

Both companies have to negotiate the issue of audio standards if they wish to succeed in the future. Dolby is involved in open standards only in so far as that may allow the company to collect licensing revenue. The company contributed to MPEG development so that it could collect from the patent pool, knowing that MPEG would be used by satellite broadcasters and would likely be mandated by various countries. Dolby's power of influence can also be seen in the DVD-Audio specification. At the 11th hour the specification was modified to include AC-3 as a standard format.

Toni Schneider of Aureal says, "Aureal has tried to strike a balance between strongly supporting both open standards such as Microsoft's DirectSound 3D and our own proprietary A3D standard. There are simple reasons why both types of standards are needed in the PC space. Open standards, especially ones endorsed by Microsoft, are a requirement to make a feature such as 3D audio a baseline feature, and eventually a legacy feature on all platforms. Proprietary standards such as A3D are required to allow companies to innovate and push the technology envelope."

For both Dolby and Aureal, the threats to PC audio standards come from the outside. In Dolby's case, the threat always exists that PC audio standards might find their way into the living room and onto consumer electronics products in much the same way that Dolby is now appearing on the PC through DVD. Aureal, on the other hand, has to be more concerned about Creative's influence than Dolby's.

Both companies wish Microsoft would give either of them the edge by incorporating the features closest to their respective hearts into DirectSound 3D. Yet, any help Microsoft may give to either company could actually result in diminishing returns — unless you own a standard, no one will pay you for it. Can anyone imagine Microsoft giving another company that kind of opportunity?

Tony Schneider of Aureal has this perspective: "The latest version of DirectSound 3D enables solid baseline 3D audio positioning. It's pretty much at the level of our original A3D that was published in '96. Our latest version, A3D 2, adds a wealth of new features that go beyond 3D positioning of sounds and into the area of 3D geometry-based acoustics. We hope that our advanced features (for example reflection and occlusion or obstruction of sound waves by walls and door ways) will eventually be adopted by Microsoft."

The PC Audio Challenge

Dolby made an interesting point about its technology. Gary Valan, Dolby's director of computer audio initiatives, said, "Dolby surround sound technologies are used in broadcasts of major sporting events, and the need for this technology is becoming just as important in sports genre game titles." So, the more mainstream games go, the more the trend plays into the hands of a company such as Dolby. After all, Dolby is as mainstream a high-end technology as you can find. If Dolby conquers PC audio it won't be due to DVD alone. Games must be a part of the picture because DVD has little viability as a playback medium for audio and video on the PC.

What's more, in current PC systems delivery of Dolby Digital tracks occurs through an audio subsystem that is separate from other PC audio tracks such as MIDI and WAV. This separation might require two separate speaker systems. Speaker companies are attempting to address this problem with new all-in-one systems, but without some standards in this area the market is likely to languish due to incompatibilities and consumer confusion.

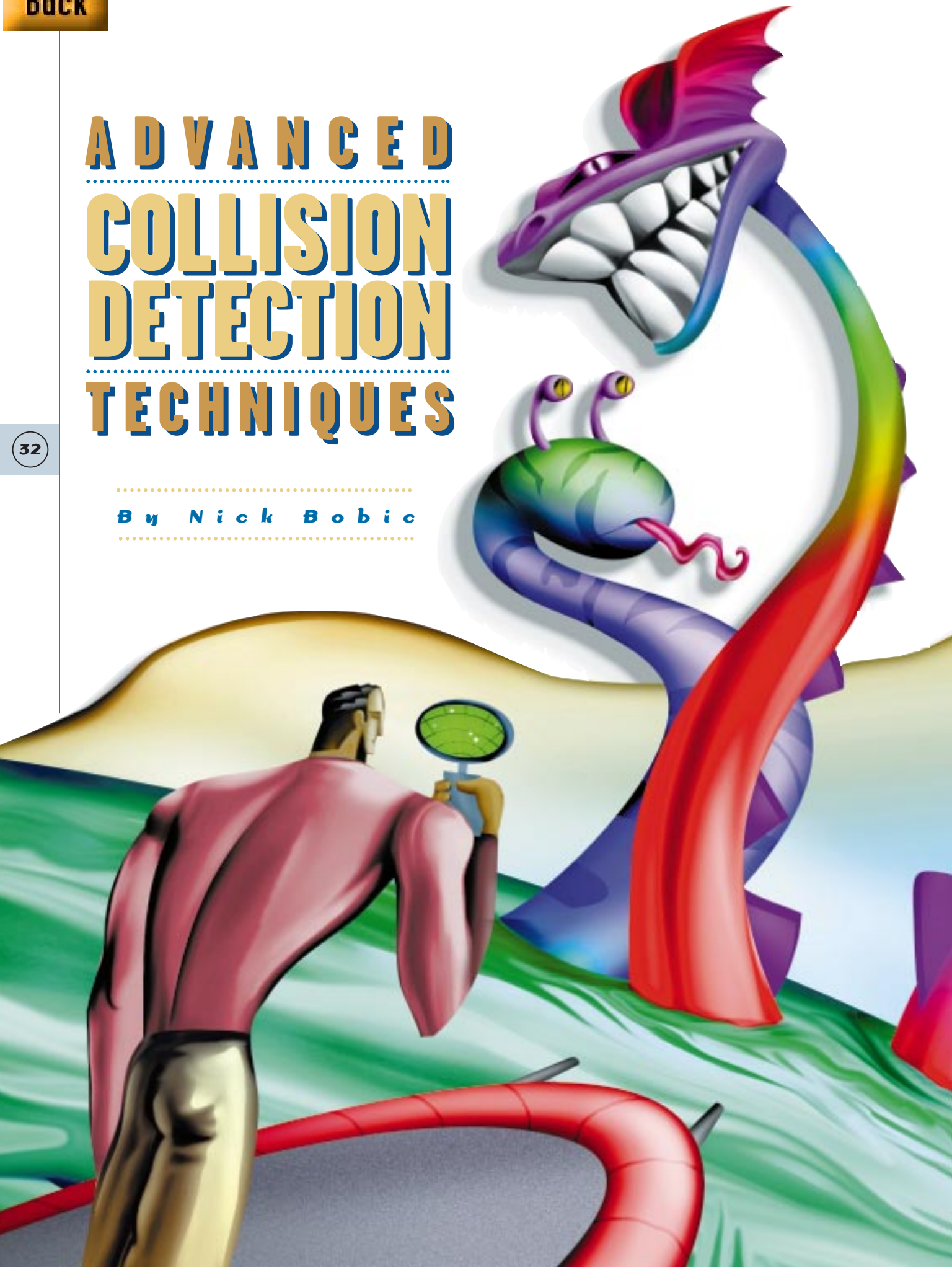
Aureal has its own problems. Primarily, the company has to be successful and profitable at some point. It's a public company, whereas Dolby has the luxury of privacy. Nevertheless, Aureal is the Phoenix that rose from the ashes of Media Vision, the multimedia hardware maker of the early nineties. The fact that Aureal is where it is today is testament to a success of sorts. But Aureal must maintain the value of its technology against the standards of DirectSound 3D and the proprietary power of Creative's EAX.

In the final analysis, both Dolby and Aureal have good news and bad news awaiting them. Tom White says, "The impracticality of multiple speakers in most desktop environments will encourage virtualization and interactive 3D for the most part, with Aureal and Creative having the lion's share of these markets. But the future of digital entertainment may not be on the desktop, but in the living room, comprised of video game consoles, set-top boxes, DTV receivers (already using AC-3) and multispeaker Dolby Digital or Dolby Surround compatible sound systems. If Dolby can secure their future in PC audio, they stand to dominate both markets, as it is very unlikely that anyone from the PC side will step in and replace Dolby in the living room."

Dolby then has to look at DTS, and George Lucas's THX among other competitors. It's not just a game, it's all entertainment, and that's why audio is such an important component in the game developers' world. Maybe it doesn't have its deserved status today, but it will tomorrow, and the day after that. To that end, Dolby is planning on virtual Dolby surround sound on two speakers, as well as a Dolby headphone. ■

ADVANCED COLLISION DETECTION TECHNIQUES

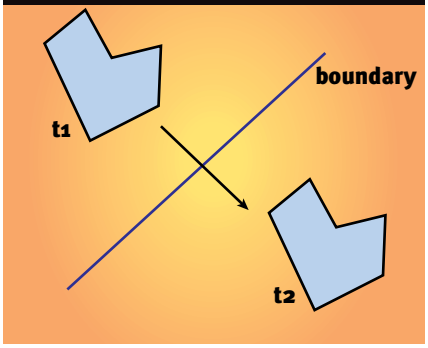
By Nick Bobic



ince the advent of computer games, programmers have continually devised ways to simulate the world more precisely. PONG, for instance, featured a moving square (a ball) and two paddles. Players had to move the paddles to an appropriate position at an appropriate time, thus rebounding the ball toward the opponent and away from the player. The root of this basic operation is primitive

Nick Bobic is trying not to work 14 hours a day with very little success. Any new collision tips and tricks should be sent to nickb@cagedent.com.

FIGURE 1. *Time gradient and collision tests.*



(by today's standards) collision detection. Today's games are much more advanced than PONG, and most are based in 3D. Collision detection in 3D is many magnitudes more difficult to implement than a simple 2D PONG game. The experience of playing some of the early flight simulators illustrated how bad collision detection can ruin a game. Flying through a mountain peak and surviving isn't very realistic. Even some recent games have exhibited collision problems. Many game players have been disappointed by the sight of their favorite heroes or heroines with parts of their bodies inside rigid walls. Even worse, many players have had the experience of being hit by a rocket or bullet that was "not even close" to them. Because today's players demand increasing levels of realism, we developers will have to do some hard thinking in order to approximate the real world in our game worlds as closely as possible.

This article will assume a basic understanding of the geometry and math involved in collision detection. At the end of the article, I'll provide some references in case you feel a bit rusty in this area. I'll also assume that you've read Jeff Lander's Graphic Content columns on collision detection ("Crashing into the New Year," January 1999; "When Two Hearts Collide," February 1999; and "Collision Response: Bouncy, Trouncy, Fun," March 1999). I'll take a top-down approach to collision detection by first looking at the whole picture and then quickly inspecting the core routines. I'll discuss collision detection for two types of graphics engines: portal-based and BSP-based engines. Because the geometry in each engine is organized very differently from the other, the techniques for world-object collision detection are very different. The object-object collision detection, for the most part, will be the same for both types of engines, depending upon your current implementation. After we cover polygonal collision detection, we'll examine how to extend what we've learned to curved objects.

The Big Picture

To create an optimal collision detection routine, we have to start planning and creating its basic framework at the same time that we're developing a game's graphics pipeline. Adding collision detection near the end of a project is very difficult. Building a quick collision detection hack near the end of a development cycle will probably ruin the whole game because it'll be impossible to make it efficient. In a perfect game engine, collision detection should be precise, efficient, and very fast. These requirements mean that collision detection has to be tied closely to the scene geometry management pipeline. Brute force methods won't work — the amount of data that today's 3D games handle per frame can be mind-boggling. Gone are the times when you could check each polygon of an object against every other polygon in the scene.

Let's begin by taking a look at a basic game engine loop (Listing 1). A quick scan of this code reveals our strategy for collision detection. We assume that collision has not occurred and update the object's position. If we find that a collision has occurred, we move the object back and do not allow it to pass the boundary (or destroy it or take some other preventative measure). However, this assumption is too simplistic because we don't know if the object's previous position is still available. You'll have to devise a scheme for what to do in this case (otherwise, you'll probably experience a crash or you'll be stuck). If you're an avid game player, you've probably noticed that in some games, the view starts to shake when you approach a wall and try to go through it. What you're experiencing is the effect of moving the player back. Shaking is the result of a coarse time gradient (time slice).

But our method is flawed. We forgot to include the time in our equation. Figure 1 shows that time is just too important to leave out. Even if an object doesn't collide at time t_1 or t_2, it may cross the boundary at time t where $t_1 < t < t_2$. This is especially true when we have large jumps between successive frames (such as when the user hit an afterburner or something like that). We'll have to find a good way to deal with this discrepancy as well.

We could treat time as a fourth dimension and do all of our calculations in 4D. These calculations can get very complex, however, so we'll stay away from them. We could also create a solid out of the space that the original object occupies between time t_1 and t_2 and then test the resulting solid against the wall (Figure 2).

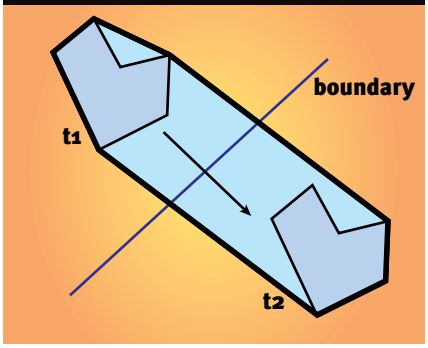
An easy approach is to create a convex hull around an object's location at two different times. This approach is very inefficient and will definitely slow down your game. Instead of construct-

LISTING 1. *Extremely simplified game loop.*

```
while(1){
        process_input();
        update_objects();
        render_world();
}

update_objects(){
        for (each_object)
                save_old_position();
                calc new_object_position  {based on velocity accel. etc.}
                if (collide_with_other_objects())
                        new_object_position = old_position();
                        {or if destroyed object remove it etc.}
}
```

FIGURE 2. *Solid created from the space that an object spans over a given time frame.*



ing a convex hull, we could construct a bounding box around the solid. We'll come back to this problem once we get accustomed to several other techniques.

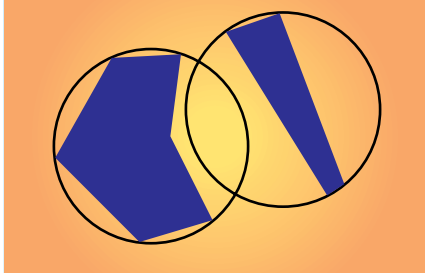
Another approach, which is easier to implement but less accurate, is to subdivide the given time interval in half and test for intersection at the midpoint. This calculation can be done recursively for each resulting half, too. This approach will be faster than the previous methods, but it's not guaranteed to catch all of the collisions.

Another hidden problem is the collide_with_other_objects() routine, which checks whether an object intersects any other object in the scene. If we have a lot of objects in the scene, this routine can get very costly. If we have to check each object against all other objects in the scene, we'll have to make roughly

$$\binom{N}{2}$$

(N choose 2) comparisons. Thus, the number of comparisons that we'll need to perform is of order N^2 (or $O(N^2)$). But we can avoid performing $O(N^2)$ pair-wise comparisons in one of several ways. For instance, we can divide our world into objects that are stationary (collidees) and objects that move (colliders) even with a v=0. For example, a rigid wall in a room is a collidee and a tennis ball thrown at the wall is a collider. We can build two spatial trees (one for each group) out of these objects, and then check which objects really have a chance of colliding. We can even restrict our environment further so that some colliders won't collide with each other — we don't have to compute collisions between two bullets, for example. This procedure will become more clear as we move on, for now, let's just say that it's possible. (Another method for reducing the number of pair-wise comparisons in a scene is to build an octree. This is beyond the scope of this article, but you can read more about octrees in *Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods*, mentioned in the "For Further Info" section at the end of this article.) Now lets take a look at portal-based engines and see why they can be a pain in the neck when it comes to collision detection.

FIGURE 3. *In a sphere-sphere intersection, the routine may report that collision has occurred when it really hasn't.*



Portal Engines and Object-Object Collisions

Portal-based engines divide a scene or world into smaller convex polyhedral sections. Convex polyhedra are well-suited for the graphics pipeline because they eliminate overdraw. Unfortunately, for the purpose of collision detection, convex polyhedra present us with some difficulties. In some tests that I performed recently, an average convex polyhedral section in our engine had about 400 to 500 polygons. Of course, this number varies with every engine because each engine builds sections using different geometric techniques. Polygon counts will also vary with each level and world. Determining whether an object's polygons penetrate the world polygons can be computationally expensive. One of the most primitive ways of doing collision detection is to approximate each object or a part of the object with a sphere, and then check whether spheres intersect each other. This method is widely used even today because it's computationally inexpensive. We merely check whether the distance between the centers of two spheres is less than the sum of the two radii (which indicates that a collision has occurred). Even better, if we calculate whether the distance squared is less than the sum of the radii squared, then we eliminate that nasty square root in our distance calculation. However, while the calculations are simple, the results are extremely imprecise (Figure 3).

But what if we use this imprecise method as simply a first step. We represent a whole character as one big sphere, and then check whether that sphere intersects with any other object in the scene. If we detect a collision and would like to increase the precision, we can subdivide the big sphere into a set of smaller spheres and check each one for collision (Figure 4). We continue to subdivide and check until we are satisfied with the approxima-

FIGURE 4. *Sphere subdivision.*

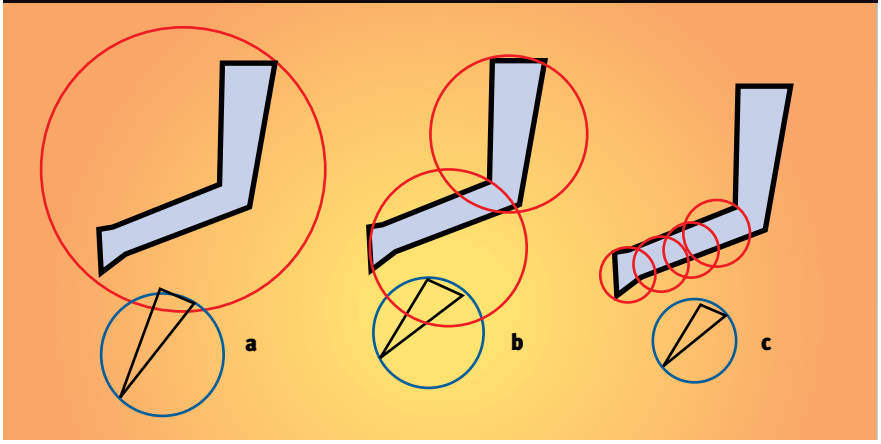
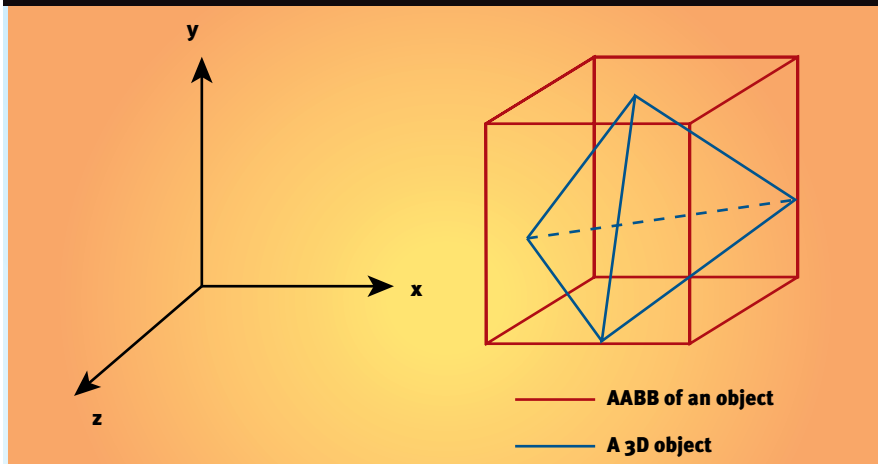


FIGURE 5. *An object and its AABB.*



tion. This basic idea of hierarchy and subdivision is what we'll try to perfect to suit our needs.

Using spheres to approximate objects is computationally inexpensive, but because most geometry in games is square, we should try to use rectangular boxes to approximate objects. Developers have long used bounding boxes and this recursive splitting to speed up various ray-tracing routines. In practice, these methods have manifested as octrees and axis-aligned bounding boxes (AABBs). Figure 5 shows an AABB and an object inside it.

"Axis-aligned" refers to the fact that either the box is aligned with the world axes or each face of the box is perpendicular to one coordinate axis. This basic piece of information can cut down the number of operations needed to transform such a box. AABBs are used in many of today's games; developers often refer to them as the model's bounding box. Again, the tradeoff for speed is precision. Because AABBs always have to be axis-aligned, we can't just rotate them when the object rotates — they have to be recomputed for each frame. Still, this computation isn't difficult and doesn't slow us down much if we know the extents of each character model. However, we still face precision issues. For example, let's assume that we're spinning a thin, rigid rod in 3D, and we'd like to construct an AABB for each frame of the animation. As we can see, the box approximates each frame differently and the precision varies (Figure 6).

So, rather than use AABBs, why can't we use boxes that are arbitrarily oriented and minimize the empty space, or error, of the box approximation. This technique is based on what are called oriented bounding boxes (OBBs) and has been used for ray tracing and interference detection for quite some time. This technique is not only more accurate, but also more robust than the AABB technique, as we shall see. However, OBBs are lot more difficult to implement, slower, and inappropriate for dynamic or procedural models (an object that morphs, for instance). It's important to note that when we subdivide an object into more and more pieces, or volumes, we're actually creating a hierarchical tree of that starting volume.

Our choice between AABBs and OBBs should be based upon the level of accuracy that we need. For a fast-action 3D shooter, we're probably better off implementing AABB collision detection — we can spare a little accuracy for the ease of implementation and speed. The source code that accompanies this article is available from the *Game Developer* web site. It should get you started with AABBs, as well as providing some examples of source code from several collision detection packages that also implement OBBs. Now that we have a basic idea of how everything works, let's look at the details of the implementation.

Building Trees

Creating OBB trees from an arbitrary mesh is probably the most difficult part of the algorithm, and it has to be tweaked and adjusted to suit the engine or game type. Figure 7 shows the creation of successive OBBs from a starting model. As we can see, we have to find the tightest box (or

FIGURE 6. *Successive AABBs for a spinning rod (as viewed from the side).*

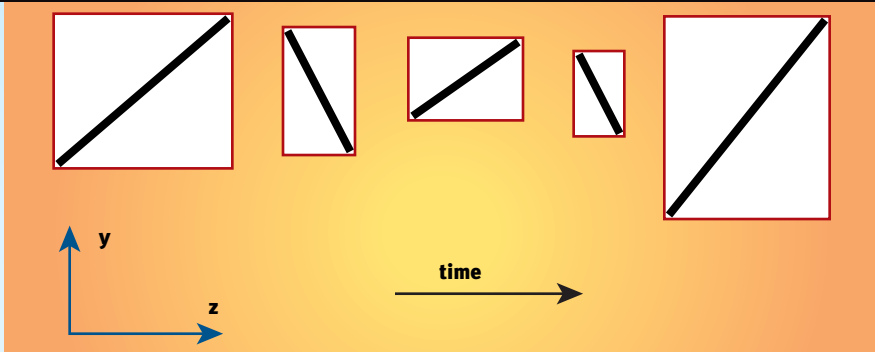


FIGURE 7. *Recursive build of an OBB and its tree.*

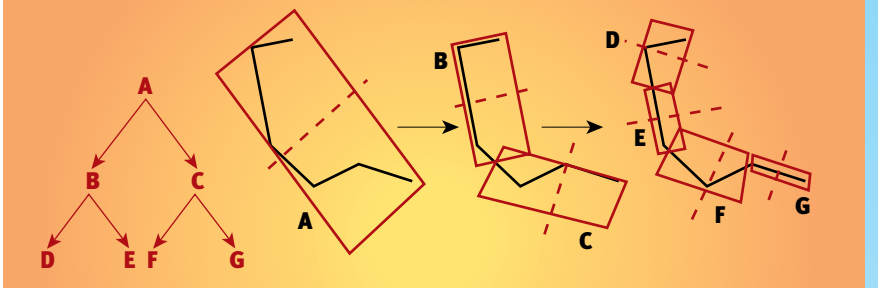
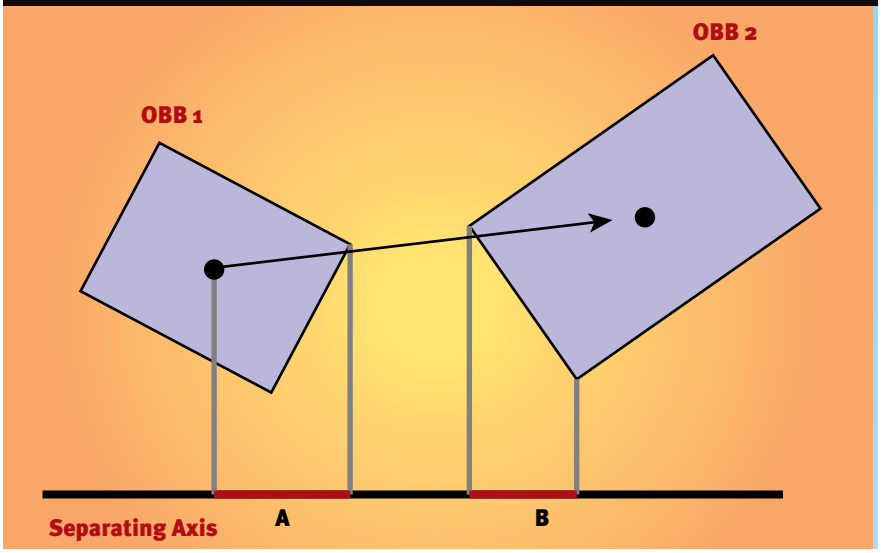


FIGURE 8. *Separating axis (intervals A and B don't overlap).*



volume, in the case of 3D) around a given model (or set of vertices).

There are several ways to precompute OBBs, and they all involve a lot of math. The basic method is to calculate the mean of the distribution of vertices as the center of the box and then calculate the covariance matrix. We then use two of the three eigenvectors of the covariance matrix to align the box with the geometry. We can also use a convex hull routine to further speed up and optimize tree creation. You can find the complete derivation in the Gottschalk, Lin, and Manocha paper cited in the "For Further Info" section.

Building AABB trees is much easier because we don't have to find the minimum bounding volume and its axis. We just have to decide where to split the model and we get the box construction for free (because it's a box parallel with the coordinate axes and it contains all of the vertices from one side of the separating plane).

So, now that we have all of the boxes, we have to construct a tree. We could use a top-down approach whereby we begin with the starting volume and recursively subdivide it. Alternatively, we could use a bottom-up approach, merging smaller volumes to get the largest volume. To subdivide the largest volume into smaller ones, we should follow several suggested rules. We split the volume along the longest axis of the box with a plane (a plane orthogonal to one of its axes) and then partition the polygons based upon which side of the partitioning axis they fall (Figure 7). If we can't subdivide along the longest axis, we subdivide along the second longest. We continue until we can't split the volume any more, and we're left with a triangle or a planar polygon. Depending on how much accuracy we really need (for instance, do we really need to detect when a single triangle is collided?), we can stop subdividing based on some arbitrary rule that we propose (the depth of a tree, the number of triangles in a volume, and so on).

As you can see, the building phase is quite complex and involves a considerable amount of computation. You definitely can't build your trees during the run time — they must be computed ahead of time. Precomputing trees eliminates the possibility of changing geometry during the run time. Another drawback is that OBBs require a large amount of matrix computations. We have to position them in space, and each subtree has to be multiplied by a matrix.

Detecting Collisions Using Hierarchy Trees

Now, let's assume that we have either our OBB or AABB trees. How do we actually perform collision detection? We'll take two trees and check whether two initial boxes overlap. If they do, they might intersect, and we'll have to recursively process them further (recursive descent). If, along the descent, we find that the subtrees do not intersect, we can stop and conclude that no intersection has occurred. If we find that the subtrees do intersect, we'll have to process the tree until we hit its leaf nodes to find out which parts overlap. So, the only thing we have to figure out is how to check whether two boxes overlap. One of the tests that we could perform would be to project the boxes on some axis in space and check whether the intervals overlap. If they don't, the given axis is called a separating axis (Figure 8).

To check quickly for overlap, we'll use something called the Separating Axis Theorem. This theorem tells us that we have only 15 potential separating axes. If overlap occurs on every single separating axis, the boxes intersect. Thus, it's very easy to determine whether or not two boxes intersect.

Interestingly, the time gradient problem mentioned earlier could easily be solved by the separating axis technique. Remember that the problem involved determining whether a collision has occurred in between any two given times. If we add velocities to the box projection intervals and they overlap on all 15 axes, then a collision has occurred. We could also use an structure that resembles an AABB tree to separate colliders and collidees and check whether they have a possibility of collision. This calculation can quickly reject the majority of the cases in a scene and will perform in an $O(N \log N)$ time that is close to optimal.

Collision Techniques Based on BSP Trees

BSP (Binary Space Partitioning) trees are another type of space subdivision technique that's been in use for many years in the game industry (DOOM was the first commercial game that used BSP trees). Even though BSP trees aren't as popular today as they have been over the past couple of years, the three most licensed game engines today — QUAKE II, UNREAL, and LITHTECH — still use them quite extensively. The beauty and extreme efficiency of BSP trees comes to light when we take a look at collision detection. Not only are BSP trees efficient for geometry culling, we

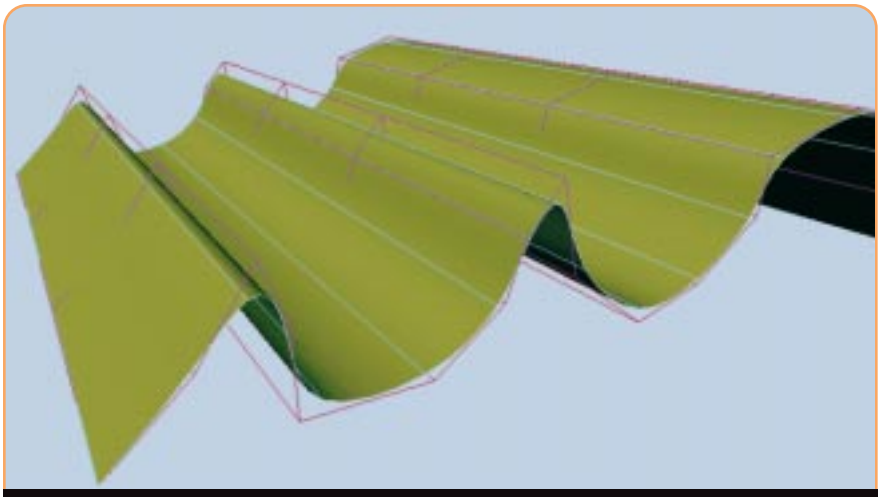


FIGURE 9. *Hull of a curved object.*

also get very efficient world-object collision almost for free.

The BSP tree traversal is the fundamental technique used with BSPs. Collision detection basically is reduced to this tree traversal, or search. This approach is powerful because it rejects a lot of geometry early, so in the end, we only test the collision detection against a small number of planes. As we've seen before, finding a separating plane between two objects is sufficient for determining that those two objects don't intersect. If a separating plane exists, no collision has occurred. So, we can recursively traverse a world's tree and check whether separating planes intersect the bounding sphere or bounding box. We can increase the accuracy of this approach by checking for every one of the object's polygons. The easiest way to perform this check is to test whether all parts of the object are on the same side of the plane. This calculation is extremely simple. We can use the Cartesian plane equation, $ax + by + cz + d = 0$, to determine the side of the plane upon which the point lies. If the equation is satisfied, then our point lies on the plane. If $ax + by + cz + d > 0$, then the point is on the positive side the plane. If $ax + by + cz + d < 0$, then the point is on the negative side the plane.

The only important thing to note is that for a collision not to occur, all of the points of an object (or a bounding box) have to be on either the positive or the negative side of a given plane. If we have points on both the positive and negative side of the plane, a collision has occurred and the plane intersects the given object.

Unfortunately, we have no elegant way of checking whether a collision has occurred in between the two intervals (although the techniques discussed at the beginning of this article still apply). However, I have yet to see another structure that has as many uses as a BSP tree.

Curved Objects and Collision Detection

Now that we've seen two approaches to collision detection for polygonal objects, lets see how we can compute the collision of curved objects. Several games will be coming out in 1999 that use curved surfaces quite extensively, so the efficient collision detection of curved surfaces will be very important in the coming year. The collision detection (which involves exact surface evaluation at a given point) of curved surfaces is extremely computationally intensive, so we'll try to avoid it. We've already discussed several methods that we could use in this case, as well. The most obvious approach is to approximate the curved surface with a lowest-tessellation representation and use this polytope for collision detection. An even easier, but less accurate, method is to construct a convex hull out of the control vertices of the curved surface and use it for the collision detection. In any case, curved surface collision approximation is very similar to general polytope collision detection. Figure 9 shows the curved surface and the convex hull formed from the control vertices.

If we combined both techniques into a sort of hybrid approach, we could first test the collision against the hull and then recursively subdivide the patch to which the hull belongs, thus increasing the accuracy tremendously.

Decide for Yourself

Now that we've gone over some of the more advanced collision detection schemes (and some basic ones, too), you should be able to decide what type of system would best suit your own game. The main thing you'll have to decide is how much accuracy you're willing to sacrifice for speed, simplicity of implementation (shorter development time), and flexibility. ■

FOR FURTHER INFO

- Note: Links to these and many more collision detection sites and online papers can be found at http://www.cagedent.com/nickb
- H. Samet. *Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods.* Addison Wesley, 1989.
- For more information about AABBs take a look at J. Arvo and D. Kirk. "A survey of ray tracing acceleration techniques," *An Introduction to Ray Tracing*. Academic Press, 1989.
- For a transformation speedup, check out James Arvo's paper in Andrew S. Glassner, ed. *Graphics Gems*. Academic Press, 1990.
- S. Gottschalk, M. Lin, and D. Manocha. "OBBTree: A hierarchical Structure for rapid interference detection," *Proc. Siggraph 96*. ACM Press, 1996. has contributed a great deal to the discussion of OBBs in terms of accuracy and speed of execution.
- S. Gottschalk. *Separating axis theorem*, TR96-024, UNC Chapel Hill, 1990. You can find the BSP FAQ at http://reality.sgi.com/cgi-bin/bspfaq
- N. Greene. "Detecting intersection of a rectangular solid and a convex polyhedron," *Graphics Gems IV*. Academic Press, 1994. introduces several techniques that speed up the overlap computation of a box and a convex polyhedron.

3D Audio SDK Round-Up

by Jonathan Blow

Positional audio has become an expected feature for today's first-person 3D games. As a result, software vendors have released a number of commercial SDKs for simulating 3D audio, and many new consumer sound cards support some kind of built-in 3D audio processing. To determine the capabilities of the major SDKs, I recently evaluated the strengths and drawbacks of five different tools.

What to Look for in a 3D Audio SDK

Game developers need to know what kinds of effects an SDK will help them create. I've presented a list of features that are important when considering an SDK for 3D and environmental audio processing. I've divided the list into two categories: the quality of the 3D effects being produced, and the engineering issues created and/or solved by using the SDK.

Quality of 3D Effects

• **Volume attenuation.** Sounds that occur further from the listener tend to be quieter than sounds that are closer. How sophisticated is the SDK's method of computing a sound's volume? How intuitive is the method for a developer to use?

• **Frequency attenuation.** Low-frequency components of a sound tend to survive better than high-frequency components when traveling large distances or through obstacles. How well are these effects modeled? How efficiently are they modeled?

• **Positional delay.** Your ears hear sounds at different times based on where the sound occurs with respect to your head. For example, if a gunshot occurs to your right, then your right ear will hear the shot slightly before your left ear. Because the speed of sound isn't all that high, the sound must travel further to reach your left ear. To help fool the listener into perceiving sound as 3D, the SDK will delay one channel of the sound with respect to the other. This effect is often referred to as the interaural time differential (ITD).

• **Head-related transfer function (HRTF).** The listener's body acts as a filter to shape the frequency spectrum of a sound based upon where the sound is occurring. These frequency alterations give the listener strong cues about the position of the sound. Therefore, it's important to simulate them. The reshaping of a frequency spectrum based on its source position is described by the HRTF for that position. The HRTF itself is a function of position, so it changes as a sound moves.

The HRTF is the most important component of a system that simulates 3D audio, so it should be considered carefully when choosing an SDK. It's good to know how well an SDK approximates HRTF, how expensive this approximation is, how much freedom the developer is given to make trade-offs of speed versus quality, and so on. Unfortunately, the information about what parts of the HRTF are simulated, and how accurately, is usually regarded by a vendor as a trade secret. Therefore, it can be difficult as a developer to have real data with which to make an informed choice.

• **Reverberation and resonance.** Sounds occurring in enclosed spaces will reverberate within that space. Certain frequencies will be resonant and others will be dampened. You should know what tools the SDK provides for creating reverberation effects, how easily they can be used, and how expressive they are.

• **Doppler shifting.** As a sound-emitting body travels, the sound waves that it emits will be compressed in its direction of travel, causing frequencies to be shifted upward. Sound waves moving in the direction that the sound is traveling away from will be decompressed, shifting their frequencies downward. Developers need to know what facilities for Doppler shifting the API provides, and how good the quality of the shifting is.

• **Crosstalk processing and handling speaker configurations.** When the listener uses speakers instead of headphones, each ear hears the output of all the speakers. This configuration can hinder attempts to trick the listener into thinking a sound is positional, so the SDK must compensate for this effect. For example, if a sound coming from the right speaker should be heard primarily by the right ear, the SDK must create a version of the samples that is the opposite of what the left ear is going to hear from the right speaker, then play that sound from the left speaker. These sounds should meet at the left ear and cancel each other out. This cancellation signal will also have to be processed by an HRTF, but not the same HRTF that is used for the left and right speakers' primary outputs. How well the SDK handles crosstalk for differing speaker configurations, and how expensive this processing is should be of interest to game developers.

It was an accident! He didn't mean it! If you know what to do when Baby turns blue, tell jon@bolt-action.com.

Engineering Considerations

• **Streaming capabilities.** Does the SDK allow you to stream in samples from a user-defined callback, so that you have full freedom to pull samples from anywhere you want or even make them up dynamically? How many other forms of streaming (for example, streaming from a very large file) does the SDK provide for your convenience? Are there any odd engineering constraints placed on our application?
• **Support for hardware acceleration.** If the game player owns a sound card with 3D or environmental audio effects built in, will the hardware be used? How well with the sound cards capabilities be used? You also ought to know whether supporting a given sound card will require you to do any extra coding.
• **Cross-platform portability.** If you tie your game to a particular audio SDK, it's critical to know whether you'll be able to release a version that runs on multiple operating systems.
• **Introspection.** How well does the SDK let you look at what's going on in the system? The more data it gives you to debug, profile, and optimize, the better.
• **Documentation and stability.** An SDK should be well documented, and it should be fairly bug free. The alternative could be a steep learning curve and inconsistent performance from your game.

Comparing the SDKs

This review focuses on the following SDKs: Microsoft's DirectSound 3D, Creative Labs' EAX 1, Aureal Semiconductor's A3D 1.2, Rad Game Tools' Miles Sound System 5 (which includes RSX 3D Audio, recently purchased from Intel), and Qsound Labs' Qmixer 4.13. These SDKs perform at various layers of abstraction, and each offers a slightly different set of services. We can't set them all out in a row, look at each, and simply declare a single winner. Such a comparison wouldn't make sense because these SDKs aren't all the same kind of tool. Instead, I'll look at them one by one, describe the scope of the problem each attempts to address, and evaluate each product's effectiveness at solving those problems.

DirectSound 3D (DirectX 6.1)

DirectSound 3D is provided by Microsoft as the standard way to play 3D sounds from Windows. Recent versions of DirectSound 3D support hardware acceleration and are extendable. Beyond that, DirectSound 3D contains a set of software routines to perform 3D audio processing when hardware isn't available, or to use as fallbacks when hardware-accelerated buffers are all in use.
• **Volume attenuation and positional delay.** DirectSound 3D provides a fair implementation of both these effects.
• **Frequency attenuation.** DirectSound 3D's software routines perform some frequency attenuation. The main components that it implements include rear processing (so that sounds coming from behind are muffled) and interaural muffling (where sounds that cross your head before reaching an ear are muffled). The facilities provided are fairly basic.
• **HRTF.** The HRTFs in DirectSound 3D are of poor quality and are usually unconvincing.
• **Doppler shifting.** DirectSound 3D contains provisions for Doppler shifting. Your game controls the amount of Doppler shift on an object by setting the velocity vector on that object.
• **Crosstalk.** DirectSound 3D doesn't attempt to handle crosstalk at all. In fact, its documentation doesn't even acknowledge crosstalk as an issue. This discrepancy, combined with the poor HRTFs, results in 3D audio that is generally ineffective.
• **Streaming capabilities.** Streaming in DirectSound 3D uses the basic DirectSound streaming conventions. DirectSound will notify the application every time a sound buffer is ready to receive new data. The application is responsible for performing all of the actual streaming operations from secondary media.
• **Documentation and stability.** The documentation for DirectSound 3D is extensive and of fair quality. It tends to be skimpy on the details of what 3D manipulations are actually performed, and primarily sticks to descriptions of the functions and their arguments. The HTMLHELP program that comes with DirectX 6 is a terrific documentation browser, much better than searching through a .PDF or .DOC file as one must do with the other SDKs.

DirectSound 3D has no major stability problems. Stability problems that arise are likely to be the fault of a vendor writing faulty DirectSound drivers.
• **Cross-platform portability.** DirectSound 3D runs under Windows.
• **Other engineering issues.** The Windows version of your game probably already requires a recent version of DirectX, so deciding to use DirectSound 3D wouldn't incur any additional packaging dependencies. The mixers and effects processors in DirectSound 3D tend to be significantly slower than the software mixers and filters provided by the other APIs.

EAX 1

EAX (Environmental Audio eXtensions) is an extension to DirectSound 3D by Creative Labs. It expands DirectSound 3D's hardware support to include reverberation control for EAX-enabled hardware (such as Creative Lab's Soundblaster Live!).

EAX lets you specify a set of three properties for reverberation in the listener's environment: volume, decay time, and damping factor. The SDK also provides a set of 26 presets so that you can get good, quick results without much programming. These presets include environments such as padded cell, auditorium, and underwater.

EAX will provide extra distance cues by adjusting the volume of a sound's reverberation relative to its primary signal (as a realistic sound recedes into the distance, its primary volume will attenuate, but its reverberated signals won't fall off so quickly).
• **Volume attenuation, frequency attenuation, positional delay, HRTF, Doppler shifting, crosstalk, and streaming capabilities.** EAX 1 presents no new functionality in these areas beyond what DirectSound 3D already provides. The quality of each of these effects depends upon the hardware that the game player has. One major drawback of EAX, as well as the rest of the products in this review, is that a severe drop in sound quality (and even presence of effects such as reverberation) occurs when the computer runs out of hardware-accelerated sound buffers and is forced to start using DirectSound 3D software emulation.
• **Reverberation.** The reverberation modeling in EAX 1 is extremely simple. It deals only with a global and vague environment of no definite shape. There's no way to say, "This sound

happened in a sewer pipe, but it's come out of the sewer pipe and is being heard at street level." Creative Labs has indicated that it will increase the sophistication of reverberation modeling in EAX in future versions to include listener-based reverb effects.

• **Cross-platform portability.** Because EAX is an extension to DirectSound 3D, it's tied to Windows.

• **Hardware support.** The EAX SDK is a programming layer for controlling effects on specific hardware that has built-in EAX support. No software implementation of the effects is available. Of course, this means that if the user's hardware doesn't support EAX, your extra programming won't provide any extra benefit.

• **Introspection.** EAX works using DirectSound 3D property lists, which can be set and inspected using DirectSound 3D. EAX also provides functions for getting preset properties and reverberation mix values.

• **Documentation and stability.** The EAX 1 Developer's Kit documentation is a 31-page booklet, and it must have taken some effort to make it that long, because there really isn't much to say. EAX is simple to use. There seem to be no stability problems.

A3D 1.2

A3D (which stands for Aureal 3D), like EAX, is an extension to DirectSound 3D. And just as EAX provides additional control over Creative Labs audio hardware, A3D gives you extra control over audio processing when it's used in conjunction with 3D sound cards that use Aureal's 3D chips (such as Diamond Multimedia's Monster Sound).

• **Volume attenuation, positional delay, HRTF, Doppler shifting, and crosstalk.** The Aureal hardware has good HRTFs which provide significantly better effects than DirectSound 3D software processing**.** A3D also does a solid job of the other major tasks in 3D positional audio. However, these benefits are provided even when you use raw DirectSound 3D, because they're essentially benefits that the accelerator hardware provides — not the A3D SDK.

• **Frequency attenuation.** The A3D SDK provides a significant new function, `SetHFAbsorb()`, which lets you adjust a sound's high-frequency rolloff. This is the amount by which higher frequencies are attenuated with distance. This feature would be used most effectively in outdoor games.

• **Reverberation.** A3D provides no facilities for controlling reverberation.

• **Cross-platform portability.** Because A3D is an extension to DirectSound 3D, it's tied to Windows.

• **Hardware support.** The point of A3D is to support hardware that incorporates Aureal's chips. That it does.

• **Introspection.** You can call a function called `GetHFAbsorb()`, which tells you the current rolloff factor of a channel.

• **Documentation and stability.** The A3D 1.2 developer documentation is a 38-page Microsoft Word file. Unfortunately, I found this documentation to be extremely poor despite its length. It spends most of its time describing what is obviously an empty SDK framework. The one important function, `SetHFAbsorbFactor`, is mislabeled in the documentation as a duplicate listing for `GetHFAbsorbFactor`. The sparse description of the function contains almost no useful information, forcing the programmer to resort to extensive experimentation to figure out exactly what the function does. It would have been helpful to have the documentation explain which exact frequencies are attenuated, how much attenuation is considered normal, what kind of attenuation-controlling curve is determined by the single floating-point input parameter, and so on. The SDK has no stability problems, but then again, it doesn't do much.

• **Other engineering concerns.** The SDK features a resource manager for the benefit of the programmer, the point of which is to reserve 3D-accelerated channels on the hardware so that higher-priority sounds can use them. When all accelerated channels are filled up, the resource manager either kills old sounds or prevents new sounds from playing. When the user's hardware supports only a low number of accelerated channels (such is the case with Diamond's Monster Sound, which has eight channels), the killed and stillborn sounds are quite noticeable — and extremely objectionable. The resource manager is billed as a "significant and dramatic improvement over DirectSound 3D," but this functionality is nothing a programmer couldn't create in a page or two of code, if so desired.

The resource manager was an attempt to address a serious problem common to hardware-based solutions: when a system runs out of accelerated channels, sudden drops in sound quality and frame rate can be caused by DirectSound 3D's software emulation. The best long-term solution to this problem is for hardware companies to manufacture sound cards with more channels. In a year or two, that will be the case, just as we now have 3D accelerators that support more fill rate than game developers need.

In the meantime, however, you have two ways around the problem. Either you can carefully segregate your sounds into those that play on hardware and those that play in software, or, if segregation isn't practical or possible, you can ignore the hardware acceleration and use software emulation exclusively. The latter is a perfectly reasonable solution, especially given choices such as Qmixer and Miles (which I'll discuss in a moment).

• **Future directions for A3D.** As this article went to press, A3D 1.2 was the only SDK available from Aureal. However, A3D 2 was in the alpha stage and will be released by the time you read this. A3D 2 is a truly outstanding leap forward, for which Aureal should be commended.

Aureal's next generation of hardware should be capable of some quite sophisticated sound processing. It will model the way solid objects reflect and occlude sound waves. A game will use the A3D 2 SDK to tell the hardware about the environment by rendering polygons to the card every frame, much as one draws polygons to a graphics accelerator using OpenGL. Each sound polygon has a material associated with it, and differing materials will have effects on sound. For instance, a noise passing through a window will be filtered quite differently than the same noise passing through a steel door. Usually, these sound polygons will be of a much lower resolution than the polygons displayed on the screen.

A3D 2 will support high-frequency rolloff, as did A3D 1.2. It will also give you the option to turn off reflections altogether (to reduce the processing load on the system), model first reflections (sound waves that bounce off of an object once before reaching your ears), or model late reflections (second-order reflections, which are the result of a sound strong enough after one reflection

to be considered for further reflections). The SDK will also provide a tagging system for game developers to designate surfaces as reflectors (as opposed to sound occluders, which are surfaces that block sounds by absorbing them) and for handling parallel surface reflectors (to reduce load on the hardware).

MILES SOUND SYSTEM 5

The Miles Sound System has been around for quite a while, and it has grown with each release. Support for 3D audio is the newest addition to Miles. It also supports the playing and sequencing of DLS, MIDI music, and Red Book audio. In this review, however, we're only going to examine the 3D features built into Miles.

Rad Game Tools made a deal with Intel to acquire Intel's RSX (Realistic Sound eXperience) audio system, which now forms the core of Miles's software 3D processing. Miles itself is an API at the same level as DirectSound 3D, wherein you designate a number of sound sources in space and describe their properties. However, it's more sophisticated and feature-filled.

• **3D effects.** The Miles Sound System doesn't implement 3D effects in any specific way. It allows a game to enumerate all of the 3D providers available and uses the one the game wishes to use. Because RSX is built into Miles and will always be available as a baseline, we will primarily discuss RSX's merits in this category. DirectSound (2D and 3D), EAX, and A3D are also supported, as is a fast and simplified software 3D engine, separate from RSX, which Rad provides for the benefit of slower machines. For users who have multispeaker systems, a Dolby Surround Sound provider is supported as well.

• **Volume and frequency attenuation and positional delay.** RSX does fairly well at these, slightly better than DirectSound 3D.

• **HRTFs.** The HRTFs provided by RSX are fairly good. They are better than those used by DirectSound 3D, but they still generally cannot match what is available in hardware.

• **Doppler shifting.** Miles supports Doppler shifting, but suffers from an annoying inconsistency: the shift is usually computed using velocities that you explicitly give the program, unless your 3D provider is RSX, in which case the shift is automatically computed from changes in object positions. Because the way in which you control the Doppler effect (and the amount of control you have over it) changes depending on the 3D provider you use, you may have to write more than one version of Doppler control in your game. This is the paradox of all umbrella-style APIs: they attempt to be generalized, but we may still end up writing extremely specialized code to get them to work generally. (Programmers who have used Direct3D Immediate Mode are quite familiar with this phenomenon.)

• **Reverberation.** The parameters that Miles provides for controlling reverberation are very similar to the parameters used by EAX.

• **Other 3D effects.** When hardware accelerated buffers are exhausted, Miles falls back to fast software routines for handling effects such as reverberation. So, when using Miles with EAX, for instance, the difference between accelerated and unaccelerated sounds is much less jarring than when you use EAX by itself.

Because Rad's strategy with Miles is to have it support as many 3D audio technologies as possible, Miles doesn't let you control the emission cone of a sound source, a technology used by DirectSound 3D and Qmixer. For example, using DirectSound 3D and Qmixer, you can declare that most of the energy of a sound is focused in a cone 60 degrees wide. I found Miles's lack of support for sound cones a bit strange, because while Miles allows you to specify the orientation of a 3D sound source, orientation doesn't affect the sound much unless the source is also assumed to have some nonspherical emission properties.

Miles doesn't explicitly support the smooth interpolation of values. What this means is that if you move a sound-emitting object or change its volume, your application cannot control the amount of time that it takes to ramp up or down the volume — this is chosen by the 3D provider you happen to be using. Smooth interpolation is good for preventing pops and other audible discontinuities from creeping into sound effects. This isn't to say that smooth interpolation doesn't take place; when using RSX, for example, it will. It's just that you have no control over the interpolation.

• **Streaming capabilities.** Miles has terrific streaming capabilities. In addition to streaming large .WAV files, it can stream ADPCM-compressed waveforms and MPEG Layer 3 (MP3) files. However, the raw interface it provides to drop samples into a buffer appears to be at a lower-level than Qmixer, and this lower-level access doesn't provide any additional benefit — it just makes it harder to use.

• **Cross-platform portability.** The Miles Sound System runs on DOS, Windows 3.1, Windows 95/98, and Windows NT.

• **Introspection.** If you're using DirectSound as a 3D provider, Miles will let you query for a DirectSound object that you can then manipulate normally. You can also instruct Miles to stop handling the processing of individual sound buffers if you'd like to control all properties of a buffer yourself. This is extremely nice when you want to accomplish a specific task that the larger framework doesn't support directly. Even better, the API lets you measure what percentage of available CPU time each channel is using. This can be terrific when you're trying to optimize your game.

• **Documentation and stability.** Miles has good documentation. It often points out specific things you'd like to know, such as whether a given API call for setting some property of a sound will affect a buffer that has not yet started playing. Also — and this is brilliant — the documentation for most important API routines tells you which sample program provided with the SDK will best demonstrate the use of that routine. I've noticed no stability issues with Miles either. However, because Miles supports 3D audio by wrapping around a variety of lower-level 3D providers, the overall stability of this solution depends upon the provider that your application chooses.

• **Other engineering concerns.** There is some weird doubling-up of API calls within Miles that can make the system difficult to use. For example, the function `AIL_set_sample_volume()` lets you set a buffer's playback volume. But it's only for 2D sounds. If you want to change the volume of a 3D sound, you have to call `AIL_set_3D_sample_volume()`. It's as though the folks at Rad felt that they couldn't afford extra `if` statements within the API, and I found this strange.

On the other hand, if you spend the

The OpenAL Vision

In 1998, a group of game developers got together to write down a design document for a portable audio API called the Open Audio Library, or OpenAL. The white paper has yet to be finished, but the need and the vision are still alive.

This year, Aureal introduced the pivotal wavetracing feature to the hardware audio market. With positional 3D audio already in place, scene-geometry aware sound rendering means a major shift towards hardware acceleration for much more immersive sound environments. Established sound APIs become obsolete along with the legacy boards they support, and the lack of an open standard holds back developers and vendors.

Toni Schneider of Aureal says, "OpenGL has shown that an open, cross-platform API can achieve both innovation and ubiquity. It is conceivable that our A3D 2 API and technology could serve as a basis for such an effort because it has been designed as a cross-platform solution. It includes a soft emulation engine, and the API is actually very close to OpenGL. All of our wavetracing 3D acoustics calls mirror the way OpenGL processes 3D geometry."

Aureal has working silicon and software, and is already engaged in standardization efforts with the IA-SIG. The company could well take the lead in designing an OpenAL API, adopting the SGI strategy for promoting an open, flexible standard — that means an Architecture Review Board, an extension mechanism, and a well-documented specification that is readily available to developers.

But the OpenAL vision isn't only about an open standard. In five years, the industry might see single-board, perhaps even single-chip solutions for 3D graphics and audio, with geometry processors handling both transformations of textured polygons and sound-reflecting surfaces. Neither game programmers nor hardware engineers want to duplicate operations, be it at the application, driver, bus, or silicon level. Consequently, design and calling conventions of OpenGL and OpenAL should be interchangeable wherever there is a true similarity in the semantics of operation. One day, a single driver might provide both for your game coding needs. For more information on the OpenAL effort, see http://www.geocities.com/SiliconValley/Hills/9956/OpenAL

– Bernd Kreimeier and Terry Sikes

money for Miles, there are many good utilities inside the libraries that your application can use. For instance, you can save and load .WAV files, or compress and decompress them using formats such as ADPCM and MP3.

For Miles to work properly with your completed game, you need to install a number of files into the same directory as the game executable. For basic functionality, this only amounts to three files. However, it can grow to as many as ten files if you want to include modules that can detect all the different 3D providers, which is a feature you surely will want.

QMIXER 4.13

The Qmixer SDK concentrates on 3D sound effects. It doesn't attempt to handle music or provide generalized utilities as Miles does. On the other hand, Qmixer provides more control over 3D sound effects than Miles does. Other than that, it's an API at the same level of abstraction as Miles. You would use Qmixer instead of DirectSound 3D. Qmixer is developed by Qsound Labs, which has been making 3D audio chips for some time now.

• **3D effects, HRTF, and crosstalk.** Qmixer's software processing is very fast compared to other SDKs, especially the buffer-mixing. It also provides more control over the effects performed on sound buffers. For example, you can control the speed with which position updates (and panning positions for 2D sounds) are interpolated, to create more precisely the effects that you want.

Qmixer positional processing algorithms aren't based on the same HRTF and crosstalk cancellation ideas discussed at the beginning of this article. It uses a proprietary approach. However, this alternative approach produces results that are certainly better than what's built into DirectSound 3D and many other software APIs. Newer hardware (such as the Soundblaster Live! or the Monster Sound MX300) tends to provide better effects than what Qmixer delivers, but if hardware acceleration is present (interfaced through DirectSound 3D), then Qmixer can use that rather than its own software algorithms.

As for speaker support, Qmixer's software engine only generates two-speaker output. However, you can tell Qmixer that you're using a four-speaker setup, and that information will be passed along to DirectSound (if it's used). Thus, if a hardware accelerator capable of dealing with four speakers is present in the user's system, then four-speaker output will be enabled. Also, Qmixer provides headphone processing as an output option, and I find that Qmixer's effects sound much better through headphones than through speakers.

• **Reverberation.** Qmixer provides no support for reverberation.

• **Introspection.** Qmixer lets you inspect all the values that you can set. Also, if you're using DirectSound, you can grab a pointer to the DirectSound object and play with that. Qmixer also contains a great debugging hook that lets you log all output from the mixer into a .WAV file for later analysis.

• **Doppler shifting.** The Doppler effect provided by Qmixer is of poor quality: aliasing often creeps into the sound (perhaps due to high-frequency foldover) and other weird processing artifacts become audible, such as artificial-sounding tones that scale with the Doppler shift. The Doppler code is also slightly buggy (see my comments of Qmixer's stability to follow). This is one area where the Qmixer developers have emphasized speed over quality. After hearing its results, you will probably turn off the Doppler effect (and perhaps simulate it yourself).

• **Streaming capabilities.** Qmixer can stream large .WAV files. Alternatively, your application can provide a callback to alert Qmixer every time it wants to fill a buffer with samples. This callback mechanism is simple and nice. It works well.

• **Hardware acceleration.** Qmixer can make use of sound hardware as identified by DirectSound 3D.

• **Cross-platform portability.** Qmixer runs on Windows and MacOS.

• **Documentation and stability.** The Qmixer documentation is passable, but could use improvement. It has improved greatly during the past 15 months, but any given revision tends

to contain listings for outdated functions and may omit important details of new routines. In general, the documentation is too vague about what's going on. For example, there's a function called `QSWaveMixGetVolume` that returns the volume of a certain channel. However, the documentation doesn't specify whether it returns the set point for that channel's volume (for example, the value that you told Qmixer to set the volume to) or the interpolated volume that Qmixer uses internally as it's smoothing out volume changes. You must experiment to find out which it is.

Qmixer tends to have minor stability problems. For example, in the more recent versions of the SDK, if you move a Doppler-shifted object too quickly (so that it approaches or exceeds the speed of sound) or set the speed of sound to be very low, the code may generate floating exceptions or other weird errors. Problems such as this tend to be fixed within a few patches, but it's slightly unnerving that the SDK makes it out the door in this state. On the other hand, I've found that the bugs haven't been enough negate the usefulness of the SDK.

• **Other engineering concerns.** Qmixer comes as a .DLL that you have to redistribute with your game. This is a bit annoying, but not as bothersome as the Miles file bonanza.

The Qmixer SDK also comes with a free, stereo-only counterpart called QMDX, which you can download from the Qsound web site; if you code to the Qmixer API and decide that it doesn't meet your needs, you can do a text search-and-replace on your files to convert them to support QMDX. Qsound generously lets you distribute your game without paying a licensing fee if you decided to use QMDX rather than the full-blown Qmixer tool.

A future version of the Miles Sound System will support the use of Qmixer as a 3D provider. Because the Miles SDK is more of an umbrella system that lets you pick and choose various lower-level audio providers, this support won't provide as much fine-tuned control over all of the Qmixer features. Thus, the version of Qmixer that ships with Miles will be feature-reduced. Developers who wish to use the Qmixer provider through Miles in a shipping game will have to pay a licensing fee. The good news is that this licensing fee will be lower than the fee for licensing Qmixer directly, in part because of the decreased technical support load that will be placed on Qsound.

What to Choose

Which SDK best suits your needs? If you want your game to run on that growing market of MacOS machines with juicy ATI Rage 128 graphics accelerators, then Qmixer is the way to go — either that or just code up the Windows version of your sound system using DirectSound 3D and write completely separate code for the MacOS and other platforms. This is a sorry situation. The establishment of an OpenAL would be a terrific improvement (see the sidebar, "The OpenAL Vision").

If your sound SDK budget is minimal, you'll want to stick to DirectSound 3D, along with explicit support for A3D and/or EAX. If, due to limited manpower, you have to pick A3D or EAX to support, I recommend EAX, because I find that its effects are much more noticeable to users. If A3D 2 fulfills its promises, however, it will eclipse both A3D 1.x and EAX.

If you're doing a lot of heavy-duty audio and music processing, I recommend Miles because of its scope and the variety of tools that it provides. To get an extremely fast software mixer, you can use the upcoming Qmixer provider for Miles. If you care very much about the amount of control that you have over your 3D sounds, use Qmixer directly. Using Qmixer directly won't allow you to specify reverberation (which Miles does), but you can still get reverberation on EAX cards by querying the DirectSound object from Qmixer, then querying that object for EAX support via the EAX SDK. ■

PRODUCT INFORMATION

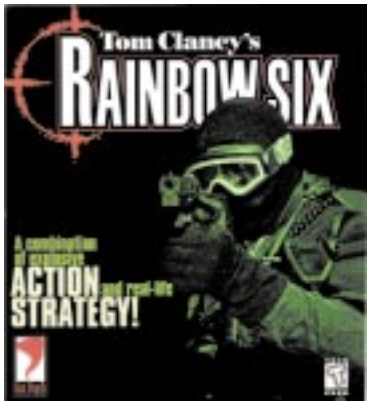
DirectSound 3D. Free. http://www.microsoft.com/directx/default.asp
EAX 1. Free. http://www.sblive.com
A3D 1.2. Free. http://www.a3d.com
Miles Sound System 5. Starts at $3,000. http://www.radgametools.com
Qmixer 4.13. Starts at $5,000. http://www.qsound.com

POSTMORTEM

Red Storm Entertainment's Rainbow Six

by Brian Upton

RAINBOW SIX and Red Storm Entertainment both came into being during the same week. When the company was formed in the fall of 1996, the first thing that we did was to spend a weekend brainstorming game ideas. That initial design session generated over a hundred possibilities that we then winnowed down to a handful that we thought had star potential. The only one that we unanimously agreed we had to build was HRT — a game based on the FBI's Hostage Rescue Team.



The Concept

It was a long road from HRT to RAINBOW SIX, but along the way, the basic outline of the title changed very little. We knew from the start that we wanted to capture the excitement of movies such as *Mission: Impossible* and *The Dirty Dozen* — the thrill of watching a team of skilled specialists pull off an operation with clockwork precision. We also knew that we wanted it to be an action game with a strong strategic component — a realistic shooter that would be fun to play even without a QUAKE player's twitch reflexes.

From that starting point, the title seemed to design itself. By the time we'd finished the first treatment a few weeks later, all the central game-play features were in place. We expanded the scope of

Brian Upton is the director of product design at Red Storm Entertainment. He has a Masters degree in computer science from the University of North Carolina at Chapel Hill and spent ten years as a graphics programmer before making the jump to full-time game designer. He can be reached at brian.upton@redstorm.com.

game (rechristened BLACK OPS) beyond hostage rescue to encompass a variety of covert missions. Play would revolve around a planning phase followed by an action phase, and players would have to pick their teams from a pool of operatives with different strengths and weaknesses. Combat would be quick and deadly, but realistic. One shot would kill, but the targeting model would favor cautious aiming over the running-and-gunning that was typical of first-person shooters.

Ironically, the only major element that we hadn't developed during those first few weeks was the tie-in to Tom Clancy's book. Clancy was part of the original brainstorming session and had responded as enthusiastically as the rest of us to the HRT concept, but he hadn't yet decided to make it the subject of his next novel. Because we had moved away from doing a strict hostage rescue game, we batted around a lot of different BLACK OPS back stories in our design meetings, ranging in time from the World War II era to the near future. For a while, we considered setting the game in the 1960s at the height of the Cold War, giving it a very *Austin Powers*/*Avengers* feel. We eventually converged on the RAINBOW SIX back story in early 1997, but we didn't find out that we would be paralleling Clancy's novel until almost April. Fortunately, we'd been sharing information back and forth the whole time, so bringing the game in line with the book didn't involve too much extra work. (If you compare the game to the novel, however, you'll notice that they have different endings. Due to scheduling constraints, we had to lock down the final missions several months before Clancy finished writing. One of the pitfalls of parallel development…)

The Production

Originally, the RAINBOW SIX team consisted of me and one other programmer. Red Storm started development on four titles straight out of the gate, and all the teams were woefully understaffed for the first few months. The first RAINBOW SIX artist didn't come on board until the spring of 1997, with a full-time producer following shortly after. With such a small group, progress was slow. During that first winter and spring, all that we had time to do was throw together a rough framework for what was to follow. This lack of resources up front would come back to haunt us later.

Because we were so understaffed, we tried to fill the gaps in our schedule by licensing several crucial pieces of technology. The first was the 3D renderer itself. Virtus Corp., our parent company, was working on a next-generation rendering library for use in its own line of 3D tools. We decided to save ourselves work by building on top of the Virtus renderer, rather than developing our own. At first, this seemed to be an ideal solution to our problem. Virtus had been doing 3D applications for years, and the renderer that its engineers were working on was a very general cross-platform solution that ran well with lots of different types of hardware acceleration.

We also went out of house for our networking technology. We had researched a variety of third-party solutions, including Microsoft's DirectPlay, but we weren't satisfied with any of them. Just as we were on the verge of deciding that we'd have to write our own library, a local development group within IBM contacted us. The group's engineers were interested in finding uses for their powerful new Java-based client/server technology. The technology, called Inverse, was designed to allow collaborative computing between large numbers of Java applets. The IBM engineers wanted to see how it would perform in a number of different application domains, including games. Inverse supported all of the features that we wanted in a networking solution, such as network time synchronization and reliable detection of disconnects, so after much deliberation we decided to use it for RAINBOW SIX. Eventually, we would come to regret both of these third-party technology decisions, but not until months later in the project.

Over the summer of 1997, we acquired most of the motion capture data that was used for animating the characters in the game. One of the advantages of working with Tom Clancy was that he put us in touch with a wide variety of consultants very quickly. Among the many experts we spoke with to get background information on counter-terrorism were two close-quarters combat trainers who worked for the arms manufacturer Heckler and Koch. When it came time to do our motion capture, these trainers volunteered to be our actors. They spent a couple of days at the Biovision studios in California being videotaped running through every motion in the game. Using real combat trainers for our motion cap-

RAINBOW SIX

Red Storm Entertainment
Morrisville, N.C.
(919) 460-1776
http://www.redstorm.com
Release date: August 1998
Intended platform: Windows 95/98
Team size: Sixteen full-time and six part-time developers
Critical development hardware: 400MHz Pentium II w/64MB RAM and a 3D accelerator
Critical development software: Microsoft Visual C++, Sourcesafe, Hiprof, Boundschecker, and 3D Studio Max.

The wide variety of Rainbow Six characters were animated with motion capture data.

ture data represented one of our better decisions. While a professional actor might have been tempted to overdo the motions for effect, these guys played it absolutely straight — the results are impressive. The game's characters come across as serious and competent, and are twice as scary as a result.

Our crisis came in October of 1997. We'd been working hard all summer, but (although we refused to admit it) we were slipping further and further behind in our schedule. Partially, the delays were the result of my being completely overloaded. Partially, they were the result of the ambitious scale of the project: because the plot of Clancy's evolving novel was driving our level design, we'd committed ourselves to creating sixteen completely unique spaces — a huge art load. And partially, they were the result of the fact that the "time-saving" technology licenses that we'd set up were proving to be anything but.

Inverse was a great networking solution — for Java. Unfortunately, we wrote RAINBOW SIX in C++. Our initial research had suggested that mixing the two would be trivial. However, in practice the overhead involved in writing and debugging an application using two different languages at the same time was staggering. The interface code that tied the two parts together was larger than the entire networking library. It became clear that we'd have to scrap Inverse and write our own networking solution from scratch if we were ever going to get the product out the door.

(As a side note, we did continue to use Inverse for our Java-based products: last year's POLITIKA and this year's RUTHLESS.COM. The problems we faced didn't arise from the code itself, but from mixing the two development environments.)

We also had problems with the Virtus rendering library. As we got deeper and deeper into RAINBOW SIX, we realized that if the game was going to run at an acceptable frame rate, we were going to have to implement a number of different renderer optimizations. Unfortunately, the Virtus renderer was a black box. It was designed to be a general-purpose solution for a wide variety of situations — a Swiss Army knife. With frame rates on high-end systems hovering in the single digits, we quickly realized that we would need a special-purpose solution instead.

In early November 1997, we put together a crisis plan. We pumped additional manpower into the team. We brought in Erik Erikson, our top graphics programmer, and Dave Weinstein, our top networking programmer, as troubleshooters. I stepped down as lead engineer and producer Carl Schnurr took over more of the game design responsibilities. The original schedule, which called for the product to ship in the spring, was pushed back four months. The artists went through several rounds of production pipeline streamlining until they could finally produce levels fast enough to meet the new ship date. Finally, we took immediate action to end our reliance on third-party software. We wrote an entire networking library from scratch and swapped it with the ailing Java code. Virtus graciously handed over the source code for the renderer and we totally overhauled it, pulling in code we'd been using on DOMINANT SPECIES, the other 3D title that Red Storm had in progress at the time. All this took place over the holiday season. It was a very hectic two months.

From that point on, our development effort was a sprint to the finish line. The team was in crunch mode from February to July 1998. A variety of crises punctuated the final months of the project. In March, I came back on board as lead engineer when Peter McMurry, who'd been running development in my place since November 1997, had to step down for health reasons. As we added more and more code, builds grew longer and longer, finally reaching several hours in length, much to the frustration of the overworked engineers. The size of the executable started breaking all our tools, making profiling and bounds checking impossible. In order to make our ship date, we had to cut deeply into our testing time, raising the risk level even higher.

On the upside though, the closer we got to the end of the project, the more the excitement started to build. We showed a couple of cautious early demos to the press in March 1998 and were thrilled by the positive responses. (At this point, we were so deep into the product that we had no idea of what an outsider would think.) The real unveiling came at the 1998 E3 in Atlanta, Ga. Members of the development team ran the demos on the show floor — for most us, that was the longest stretch we'd had playing the game before it shipped. Almost all of the final gameplay tweaks came out of what we learned over those three days.

What Went Right

1. A COHERENT VISION. Throughout all of the ups and downs in the production process, RAINBOW SIX's core game play never changed. We estab-

Game play involves a planning phase followed by an action phase.

The various mission levels called for the creation of sixteen completely unique spaces.

lished early on a vision of what the final game would be and we maintained that vision right through to the end. I can't overstate the importance of this consistency. Simply sticking to the original concept saw the team through some really rough parts of the development cycle.

For one thing, this coherent vision meant that we were able to squeak by without adequate design documents. Many parts of the design were never written down, but because the team had a good idea of where we were headed, we were able to fill in many of the details on our own. Even when we had to perform massive engineering overhauls in the middle of the project, a lot of the existing art and code was salvageable.

Our vision also did a lot for morale. Many times we wondered if we'd ever finish the project, but we never doubted that the result would be great if we did. It's a lot easier to justify crunch hours when you believe in where the project is going.

2. An efficient art pipeline. The art team tried out four or five different production pipelines before they finally found one that would produce the levels that we wanted in the time that we had available. The problem was that we wanted to have sixteen unique spaces in the game — there would be almost no texture or geometry sharing from mission to mission. Furthermore, instead of creating our own level-building tool, we built everything using 3D Studio Max. Thus, artists had more freedom in the types of spaces that they could create, but they didn't have shortcuts to stamp out generic parts such as corridors or stairwells — everything had to be modeled by hand.

Eventually, the art team settled on a process designed to minimize the amount of wasted effort. Before anyone did any modeling, an artist would sketch out the entire level on paper and submit it for approval by both the producer and art lead. Then the modelers would build and play test just the level's geometry before it was textured. Each artist had a second computer on his desk running a lightweight version of the game engine so he could easily experiment with how the level would run in the game.

3. Tom Clancy's visibility. A good license won't help a bad game, but it can give a good game the visibility it needs to be a breakout title. When we first approached members of the gaming press with demos of Rainbow Six in the spring of 1998, they had no reason to take us seriously — we had no track record, no star developers, and no hype (O.K., not much hype...). We were showing a quirky title with a less-than-state-of-the-art rendering engine in a very competitive genre. With much-anticipated heavyweights such as Sin, Half-Life and Daikatana on the way, having Clancy's name on the box was crucial to getting people to take a first look at the title. Fortunately, the game play was compelling enough to turn those first looks into a groundswell of good press that carried us through to the launch.

4. Reworking the physics engine. In February 1998, we completely overhauled the Rainbow Six physics engine, which turned out to be a win on a variety of fronts. We'd retooled the renderer during the previous month, but our frame rate was still dragging. After running the code through a profiler, we figured that most of our time was going to collision checks — checks for characters colliding with the world and line-of-sight checks for the AI's visibility routines.

The problem was that every time the physics engine was asked to check for a collision, it calculated a very general 3D solution. Except in the cases of grenade bounces and bullet tracks, a

A 2D floor plan, created for the purpose of collision detection, also helped players in both the planning and action interfaces.

3D collision check was complete overkill. Over the next month, we reworked the engine to do most of its collision detection in 2D using a floor plan of the level. These collision floor plans would be generated algorithmically from the 3D level models.

The technique worked. In addition to getting the frame rate back up to a playable level, it also made collision detection more reliable. The game engine also used the floor plans to drive the pathfinding routines for the AI team members. Players would view these same floor plans as level maps in both the planning and action interfaces. By figuring out how to fix our low frame rates, we wound up with solutions to three or four other major outstanding engineering issues. Sometimes, the right thing to do is just throw part of the code out and start over.

5. TEAM COHESION. Red Storm employs no rock stars and no slackers. Everyone on the RAINBOW SIX team worked incredibly long hours under a tremendous amount of pressure, but managed (mostly) to keep their tempers and their professional focus.

What Went Wrong

1. LACK OF UP-FRONT DESIGN. We never had a proper design document, which meant that we generated a lot of code and art that we later had to scrap. What's worse, because we didn't have a detailed outline of what we were trying to build, we had no way to measure our progress (or lack thereof) accurately. We only realized that we were in trouble when it became glaringly obvious. If we'd been about the design rigorous up front, we would have known that we were slipping much sooner.

2. UNDERSTAFFING AT THE START. This point is closely related to the previous point. Because we didn't have a firm design, it was impossible to do accurate time estimates. Red Storm was starved for manpower across the board, and because we didn't have a proper schedule, it was hard to come to grips with just how deep a hole we were digging for ourselves. There were always plenty of other things to do in getting a new company off the ground besides recruiting, and we were trying to run as lean as possible to make the most of our limited start-up capital. Given the circumstances, it was easy to rationalize understaffing the project and delaying new hires.

Additionally, I badly overestimated my own abilities. For Red Storm's first year, I was working four jobs: VP of engineering, lead engineer on RAINBOW SIX, designer on RAINBOW SIX, and programmer. Any one of these could have been a full-time position. In trying to cover all four, I spent all my time racing from one crisis to the next instead of actually getting real work done. And because I was acting as my own manager, there was no one to audit my performance. If one of the other leads was shirking his scheduling duties or blowing his milestones, I'd call him on it. But on my own project, I could always explain away what should have been clear warning signs of trouble.

3. RELIANCE ON UNPROVEN TECHNOLOGY. Our external solutions for rendering and networking both fell through and had to be replaced with internally developed code late in the development cycle. In both cases, we were relying on software that was still under development. The core technology was sound, but we were plagued with inad-

equate documentation, changing programming interfaces, misunderstood performance requirements, and heavy integration costs. Because both packages were in flux, we failed to do a thorough evaluation of their limitations and capabilities. By the time it became obvious that neither was completely suited to our needs, it was too late to push for changes. In retrospect, we would have saved money and had a much smoother development process if we'd bitten the bullet early on and committed ourselves to building our own technology base.

4. Loss of key personnel. Losing even a junior member of a development team close to gold master can be devastating. When our lead engineer took ill in February 1998, we were faced with a serious crisis. For a few frantic weeks, we tried to recruit a lead from outside the company, but eventually it became obvious that there was no way we could bring someone in and get them up to speed in time for us to make our ship date in July 1998. Promoting from inside the team wasn't a possibility either — everyone's schedule was so tightly packed that they were already pulling overtime just to get their coding tasks done; no one had the bandwidth to handle lead responsibilities too.

Ultimately, I wound up stepping back in as lead. This time, however, we knew that for this arrangement to work I'd have to let my VP duties slide. The rest of management and the other senior engineers took up a lot of the slack, and Peter had set a strong direction for the project, so the transition went very smoothly. (After his health improved Peter returned to work at the end of the project, putting in reduced hours to finish off the RAINBOW SIX sound code.)

5. Insufficient testing time. We got lucky. As a result of our early missteps, the only way we could get the game done on time was to cut deeply into our testing schedule. We were still finding new crash bugs a week before gold master; if any of these had required major reengineering to fix, we would have been in deep trouble. That the game shipped as clean as it did is a testament to the incredible effort put in at the end by the engineering team. As it was, we still had to release several patches to clean up stuff that slipped through the cracks.

In the End...

RAINBOW SIX's development cycle was a 21-month roller coaster ride. The project was too ambitious from the start, particularly with the undersized, inexperienced team with which we began. We survived major overhauls of the graphics, networking, and simulation software late in the development cycle, as well as two changes of engineering leads within six months. By all rights, the final product should have been a buggy, unplayable mess. The reason it's not is that lots of very talented people put in lots of hard work.

I'm not going to say that RAINBOW SIX is the perfect game, but it is almost exactly the game that we originally set out to make back in 1996, both in look and game play. And the lessons that we've learned from the RAINBOW SIX production cycle have already been rolled into the next round of Red Storm products. Our current focus is on getting solid designs done up front and solid testing done on the back end — and on making great games, of course. ■

by Scott Draeker

Making a Case for Linux Games

Great ideas seem obvious in retrospect. Nearly eight months after founding Loki Entertainment Software, putting games onto Linux is starting to seem obvious, too.

For those of you recently returning from the outer rings, Linux is a free, open-source operating system. It's fast, stable, reliable, and responsive — technically equivalent and often superior to commercial operating systems because Linux development is driven by technology, not marketing. Think of the Linux development community as the world's only functioning meritocracy. Only the best code survives. A solid estimate of Linux users is difficult to come by — it's perfectly acceptable to download the OS or copy it for a friend — but the most reliable figures put the 1998 Linux installed base somewhere between 12 and 15 million.

Illustration by Pamela Hobbs

Still, you might ask, is Linux ready for the desktop? Isn't it just a server OS? Well, International Data Corp. estimates that Linux held about two percent of the worldwide *desktop* market in 1998. Quite remarkable for an OS which has only recently begun to see desktop applications. The trend is familiar. New technologies often trickle down from high-end applications, such as servers, to the consumer desktop.

It's true that most of the applications available for Linux today are server applications. But consumer applications are beginning to appear. There are two very good graphical user interfaces for Linux already available: KDE (http://www.kde.org) and GNOME (http://www.gnome.org). There are also several good Linux office applications, including WordPerfect 8 for Linux. Corel's WordPerfect was recently released over 250,000 times within two weeks of its release. Who's to say games aren't next?

Not only will Linux become an increasingly viable desktop OS — I believe that it's also going to be the gaming OS of choice. Because Linux is open source, it's possible to make changes to the OS itself to enhance game performance. By developing on Linux, the game industry will further Linux development — and build in superior game play.

That's why why we chose to port Activision's CIVILIZATION: CALL TO POWER to the Linux platform. At Loki, we license the rights to port successful game titles to Linux. The original developer provides us with source code (which we do not release). We then port the game. Loki tests, publishes, and supports the Linux port — and pays the original developer royalties. This way, we're able to deliver the best titles the PC game world has to offer to our customers.

All developers could potentially benefit. Game-related software libraries are also open source in the Linux world and thus gain all the benefits of the open-source model. In our own company, we are currently using the Simple DirectMedia Layer (SDL) (http://www.devolution.com/~slouken/SDL/) to support input, graphics, and sound. The changes we make to SDL in our porting work will be publicly available — source code and all. Eyes will begin combing the code. With thousands of developers scrutinizing SDL code, bugs will be found and fixed faster than they would in any single company's product.

Open source also encourages open standards. And open standards translate into lower costs for developers and fewer headaches for users. Linux is far more likely to standardize on a particular 3D API, for example. By contrast, Windows developers struggle to support competing 3D, sound, and other proprietary formats.

The combined benefits of an open source OS, open source libraries, and open standards add up to a superior gaming environment. In the near future, the same game running on the same hardware will be faster, more stable, and more responsive on Linux. Hardcore gamers will pick up on this quickly. What about game developers? ■

Scott Draeker was a practicing attorney specializing in software and technology licensing issues when legal research (surfing) on the Internet led him to the Linux community. An avid game player, he became a vocal proponent of Linux gaming and eventually formed Loki Entertainment Software in August 1998.