gd

GAME DEVELOPER MAGAZINE

SEPTEMBER 1998

UBM Tech

Where's Our Sundance?

Some years back, certain forces decided that a marriage of Hollywood and Silicon Valley should be arranged. Much was written about this nuptial, and many prognosticated that this joining would forever change both industries. This vision of the future never materialized, and many game developers have been happy to distance themselves from their movie-making brethren, but lately I've been thinking that one aspect of the film industry does deserve a closer look by game developers — the concept of the indie film festival. The Sundance Film Festival is an event we could do well to imitate.

The Sundance Festival came out of nowhere in 1978 (at the time it was called the "U.S. Film Festival") as the country's first film festival. The original idea for the festival was simple enough: to bring filmmakers, authors, and actors to Utah to screen classic indies and talk about current social themes in films. ("Indies" are films made outside of the massive, often formulaic Hollywood system, and are typically characterized by their miniscule budgets.) The late Arthur Knight, then a professor of film at USC, suggested turning the festival into a national competition to foster the emerging market of American-made independent films. As the festival grew, the premieres began to dominate the program, and showings of older films were scaled back.

Today the Sundance Festival is one of the film industry's biggest events, and its highlight is the American Independent Dramatic and Documentary Competition, where new American indies are premiered. It's hard to overstate the exposure that this competition gives to these films. Most of these indies wouldn't be seen by distributors and studios otherwise. As a result, many indie filmmakers look to Sundance as their opportunity to present their films before an influential audience. Distributors see what's available and often sign on indies for wide distribution. Film industry execs go to the festival to unearth undiscovered talent and see what themes cutting-edge films are exploring. Audiences come in droves to preview the movies and, hopefully, rub elbows with the movie-making crowd.

The Sundance idea has since been replicated all over the world, and chances are that there's a city near you that has its own film festival now, but it's the Sundance festival that most filmmakers pine for.

Is it just me, or is the game development community missing out on a golden opportunity here? I think the time is ripe to create an event like this to highlight outstanding new games, particularly those by smaller game development companies. Right now, the big launch (preview, really) event for game developers is E3. While E3 definitely serves its purpose in hooking up established publishers with the likes of Wal-Mart purchasing agents, it doesn't cut it for the less-established developer looking to sign a game to a publisher — or even just break into the market. Between the 30,000 people streaming though the cavernous tradeshow floor, the pulsating beat of high-decibel techno-rock, and the cost of exhibiting at this tradeshow, the little guy doesn't stand much of a chance to get noticed.

What the game development industry needs is another, more relaxed venue where "indie" games can be "screened" in a comfortable setting, and where the gaming public (and other aspiring developers) can see what kinds of titles are being developed on a shoestring. Like Sundance, there ought to be eligibility requirements, a jury to select and judge entries, and awards for the winners. While you might scoff at the idea of a festival of indie games, I know some damn talented developers who could use this kind of event to show off their projects and skills. If those who dismissed the idea of an indie film festival were taken seriously a few decades ago, great indie films like *Hoop Dreams; sex, lies, and videotape; Crumb; Clerks; The Brothers McMullen; Paris is Burning*; and other excellent Sundance premieres might never have caught on (or even have been made).

If the Sundance idea intrigues you too, let me know — I'd like to hear from you. Send your thoughts and ideas to me at adunne@sirius.com. We'll see where this takes us. ■

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

EDITOR IN CHIEF Alex Dunne adunne@sirius.com
MANAGING EDITOR Tor D. Berg tdberg@sirius.com
DEPARTMENTS EDITOR Wesley Hall whall@mfi.com
ART DIRECTOR Laura Pool lpool@mfi.com
EDITOR-AT-LARGE Chris Hecker checker@d6.com
CONTRIBUTING EDITORS Jeff Lander jeffl@darwin3d.com
Josh White josh@vectorg.com
Omid Rahmat omid@compuserve.com
ADVISORY BOARD Hal Barwood
Noah Falstein
Brian Hook
Susan Lee-Merrow
Mark Miller

COVER IMAGE dub Media Inc.

PUBLISHER Cynthia A. Blair cblair@mfi.com
WESTERN REGIONAL SALES MANAGER Alicia Langer (415) 905-2156 alanger@mfi.com
EASTERN REGIONAL SALES MANAGER Kim Love (415) 905-2175 klove@mfi.com
SALES ASSOCIATE Ayrien Houchin (415) 905-2788 ahouchin@mfi.com

MARKETING MANAGER Susan McDonald
AD. PRODUCTION COORDINATOR Dave Perrotti
DIRECTOR OF PRODUCTION Andrew A. Mickus
VICE PRESIDENT/CIRCULATION Jerry M. Okabe
ASST. CIRCULATION DIRECTOR Mike Poplardo
CIRCULATION MANAGER Stephanie Blake
CIRCULATION ASSISTANT Kausha Jackson-Craine
NEWSSTAND ANALYST Joyce Gorsuch
REPRINTS Stella Valdez (916) 983-6971

un Miller Freeman
A United News & Media publication

CEO-MILLER FREEMAN GLOBAL Tony Tillin
CHAIRMAN-MILLER FREEMAN INC. Marshall W. Freeman
PRESIDENT/COO Donald A. Pazour
SENIOR VICE PRESIDENT/CFO Warren "Andy" Ambrose
SENIOR VICE PRESIDENTS H. Ted Bahr
Darrell Denny
David Nussbaum
Galen A. Poss
Wini D. Ragus
Regina Starr Ridley
VICE PRESIDENT/PRODUCTION Andrew A. Mickus
VICE PRESIDENT/CIRCULATION Jerry M. Okabe
VICE PRESIDENT/GROUP DIRECTOR KoAnn Vikören
SENIOR VICE PRESIDENT/SYSTEMS AND SOFTWARE DIVISION Regina Starr Ridley

www.gdmag.com

In Defense of Curriculum Designers

I was outraged by Seymour Papert's irresponsible, self-serving Soapbox in the June 1998 issue of *Game Developer*. As an instructional designer, I believe he paints a very misleading picture of what motivates us. I'm offended by his accusation that it's in a curriculum designer's best interest for a student not to learn (so that a new curriculum is needed and brings more business). The instructional designers I've known are dedicated to educating others. Teaching is not a lucrative profession. Clearly, financial gain isn't our primary motivator.

Papert seals his hypocrisy by using his article simply to promote his latest book. This cripples his argument and makes his column a self-serving advertisement that panders to a target audience of programmers and others in the game industry who might read it and think "I've always said we didn't need instructional designers!"

As an instructional editor and project manager of computer-based training programs at Total Learning Concepts, I made sure that our customers learned key concepts. Any perpetuation of business came not from doing a poor job, but rather from always doing the best job that I could. If our clients didn't learn from our material, they certainly wouldn't come back to us for help in the future.

Another serious flaw in Papert's argument is his comparison of learning how to play a videogame with learning math or reading. A "Professor of Learning" must be aware that the brain does not process all information in exactly the same way. How can Papert justify clumping all these different types of information processing into one category? I suggest he, or anyone interested in how people learn different skills, read any work by the foremost expert in the field of learning, Howard Gardner. Professor Gardner's pioneering theories have led to the identification of eight different types of learning. I'd like to know exactly which games Papert thinks teach players how to learn, and what specific learning skills he thinks they are developing by playing these games.

Papert's column might lead a reader to assume that all types of learning can be grouped into one category, or that a designer who is great at teaching one set of skills may be an expert in every field. It's true that teaching a person how to learn is considered the ultimate goal of academia, but when it comes to educating students in specific subject matter, the medium cannot afford to become the message.

A panel discussion at this year's E3 on educational software presented another important reason for a curriculum designer to lend input on an educational product: promoting the product in the classroom market. In order for a school system to adopt educational software, the software must comply with the school's curriculum. Great instructional designers are not only familiar with a given school system's curriculum; they also have experience implementing the curriculum in the classroom.

Got some issues? E-mail us at gdmag@mfi.com. Or write to *Game Developer*, 600 Harrison Street, San Francisco, CA 94107.

I have the utmost respect everyone involved with project development, and recognize the important contributions of each member of the team. I've learned much about game design from incredibly talented, insightful programmers. I hope game developers recognize the advantages and benefits that a good instructional designer can bring to an educational product. Instead of simply replacing instructional designers with programmers, I recommend that the two work closely together to create better educational games.

Zareh MacPherson Artinian
via e-mail

PAPERT RESPONDS: *I'll refrain from devoting more than one sentence to Artinian's personal flames. What Artinian sees as venal book-selling commercialism, someone more familiar with academic practices would see as standard scholarly responsibility. But the issues at stake are deeper than a ping-pong debate between Artinian and Papert. Our conflict of opinion is an incident in a worldwide confrontation between two opposed perspectives on learning.*

I am not cowed by the fact that eminent members of the education establishment would support Artinian's position. Of course they would. The battle is about a challenge from new technologies and from new theories of learning that threaten to overthrow the accepted structure of school, the idea of curriculum, the segregation of children by age, and pretty well everything that the education establishment will defend to the bitter end.

Artinian throws out a challenge that highlights one key position in this battle: "I'd like to know exactly which games Papert thinks teach players how to learn, and what specific learning skills he thinks they are developing by playing these games." The most important learning skills that I see children getting from games are those that support the empowering sense of taking charge of their own learning. And the learner taking charge of learning is antithetical to the dominant ideology of curriculum design. By definition, curriculum design means assigning to experts the job of deciding the best way for each individual to learn each subject. The power of the idea of taking responsibility for one's own learning applies to all learning. It is sheer mystification to suggest that no principles can be shared by all forms of learning. Saying that the learner is in charge does not mean that everyone has to re-invent every wheel. Good learners will recognize the limits of their inventiveness and seek help. In the past, the opportunities for school-aged people to do this effectively were extremely limited. They still are today. But the presence of digital technologies is rapidly moving us into a period where learners can learn what they need to know on their own agenda rather than on the predetermined agenda of a curriculum. We will soon be able to give up the assembly line model of grade after grade, exercise after exercise.

It would be naive to believe this could happen without resistance from the education establishment — which includes several multibillion dollar sectors of the education industry as well as huge bureaucracies with a vested interest in maintaining the status quo. I grant that most people who make and apply curriculums are underpaid and motivated by the welfare of children. But this does not alter the fact that present-day schools, to which (as Artinian actually boasts) they have to cater in order to sell their products, are relics from an earlier period of knowledge technology.

BIT BL

NEWS FROM THE WORLD

INDUSTRY WATCH

by Alex Dunne

WE'LL MISS YOU. Game designer Dani Bunten Berry, who developed such classics as SEVEN CITIES OF GOLD, M.U.L.E., ROBOT RASCALS, COMMAND H.Q., and MODEM WARS, passed away on July 3rd, following a battle with lung cancer. Dani was a pioneer in the industry, and a truly remarkable person that many will miss. She was 49.

ANOTHER TLC SPREE. The Learning Company is purchasing Brøderbund Software in a stock swap valued at $420 million. TLC has been on an intense shopping spree this year, netting Mindscape in March and PF. Magic in May. Following this acquisition, The Learning Company will control about 40 percent of the educational software market, according to analysts.

BRING BACK SPROCKETS? In his keynote address at MacWorld in New York, interim CEO Steve Jobs explained the importance of the consumer market to Apple and singled out game development as a priority. Jobs took a swipe at his predecessors, jokingly commenting that, "For some reason, Apple's previous management didn't like games. We do now, though." A video segment showed that there are indeed some A-titles coming for the MacOS, including Eidos' TOMB RAIDER series, two MicroProse Star Trek titles, GT Interactive's UNREAL, Microsoft's AGE OF EMPIRES and CLOSE COMBAT, and an undisclosed LucasArts' Star Wars game.

ACCLAIM REPORTED EXCELLENT RESULTS for its third fiscal quarter (ending May 31). The company's $73.2 million in revenue represents a 76 percent increase over the same period last year. Profits came in at $5.7 million, compared with a net loss of $69.7 million for the same period last year. Acclaim cited Nintendo's price reduction on N64 cartridges to third-party licensees, as well as strong demand for its Acclaim Sports branded titles such as ALL-STAR BASEBALL 99 and JEREMY MCGRATH

Sim Solutions

Two MIG 29s flying down Yosemite canyon, taken from a run-time .EXE.

NDIMENSION recently released SimStudio, a development solution for simulations that enables you to automate time-consuming, complex tasks and easily add real-world physics, accurate dynamics, true collision detection, and behavior modeling to your game world.

SimStudio uses object-oriented tools and is best suited to applications that require real-time dynamics and a very high degree of interactivity — such as vehicular, defense, architecture, marine, and flight sims, among others. The suite contains three core products: the IDE, a C++ API, and the real-time simulation executive (RTSX). It features dynamics and AI editing, predictive collision detection, the ability to create custom plug-in artificial intelligence and dynamics modules, intelligent line-of-sight and height above terrain, and the creation of unlimited scenario databases. SimStudio also imports 3D Studio and MultiGen models.

SimStudio runs on Windows 95 or Windows NT 4.0 with an Intel Pentium-class CPU of 166MHz or better. You'll also need a 3D accelerator/IG with Windows support. SimStudio supports any OpenGL or Direct3D capable hardware, including the 3Dfx Voodoo and Voodoo2 based hardware. The suite sells for $3,995. Plug-in modules and device extensions are also available.

- Ndimension Simulations Pty. Ltd.
  Santa Clara, Calif.
  (408) 986-0900
  http://www.ndimension.com

Character Studio R2

KINETIX is now shipping the latest incarnation of Character Studio software, its the character animation plug-in software extension to 3D Studio MAX R2 and R2.5.

Character Studio R2 combines motion capture, editing, and blending technology with traditional keyframe animation, new skin deformation tools, and Character Studio's footstep-driven technique. Character Studio R2 ships with a library of more than 500 ready-to-use motion capture samples representing a variety of sequences, such as punching, kicking, jumping, and running. Custom motion capture and hand animations can be integrated with the packaged sequences. The tool is comprised of Biped, a hybrid footstep-driven motion capture/keyframe animation system, and Physique, an interactive skinning system.

Character Studio R2 software is available worldwide as a plug-in application for 3D Studio MAX R2 and R2.5, and is U.S. list priced at $1,495. Upgrades from Characters Studio R1 are priced at $495.

- Kinetix
  San Francisco, Calif.
  (800) 879-4233 / (415) 547-2000
  http://www.ktx.com

trueSpace 4

CALIGARI unveiled trueSpace4 — the

new version of its authoring tool for interactive Web, graphic, and game design — at SIGGRAPH in July.

The latest version of trueSpace introduces three major new features: a photorealistic renderer with hybrid radiosity, a true 3D user interface, and game-quality hardware acceleration. "Hybrid" radiosity means that you can combine traditional phong shading with ray tracing and radiosity both in the same image. The new renderer also supports features such as atmospheric rendering, volumetric shadows, reflectance shaders (including true transparency and anisotropic reflectors), and lens flares, among others. Caligari has altered their 3D user interface by replacing traditional control elements such as dialogue boxes, buttons, and sliders with 3D widgets — thus moving even the controls into the 3D environment, which means the interface is fully accelerated by hardware. This allows you to perform all editing operations in real time. Other new features include NURBS surfaces, bones (the ability to create and manipulate the skeleton of a 3D character), polygonal editing and per-face texturing, and function curves.

Caligari's trueSpace 4 runs smoothly on current generation 3D chips, and is optimized for next generation chips such as Permedia3 or the RIVA TNT. trueSpace4 has a street price of $595.

■ Caligari Corp.
Mountain View, Calif.
(800) 351-7620 / (415) 390-9600
http://www.caligari.com

SurfaceSuite Stand-alone

SVEN TECHNOLOGIES has just released a stand-alone version of SurfaceSuite Pro, an adaptive texture mapping application for 3D artists, animators, and designers.

Adaptive texture mapping is a technique developed by Sven Technologies to facilitate efficient, photorealistic texture mapping by treating texture maps as sheets of rubber that can be placed, warped, and blended on a 3D model in real time. The tool's control-point-based mapping system allows you to take photographic source images (pictures from a digital camera, scanned photos, and so on) and map them onto complex 3D models. You can apply control points onto a 3D model and then apply corresponding points to a 2D texture. The stand-alone version of SurfaceSuite Pro offers all the adaptive texture mapping, blending, and compositing capabilities of the earlier plug-in version and contains several new features. New features include: wide compatibility with most 3D modeling applications, real-time texture blending, NURBS and patch texturing, an enhanced graphical user interface, and the "Relaxer" object-dependent texture mapping. The Relaxer allows you to generate a texture-coordinate mapping that is optimized for even surface distribution of textures over an object — this eliminates texture pinching, streaking, and stretching.

As a plug-in, SurfaceSuite Pro is compatible with 3D Studio MAX, and sells for $495. As a stand-alone, SurfaceSuite Pro is available for $595, and is compatible with a variety of 3D modeling tools.

■ Sven Technologies Inc.
Palo Alto, Calif.
(650) 852-9242
http://www.sven-tech.com

Using SurfaceSuite Pro's control-point-based mapping system to map a photo onto a 3D model.

SUPERCROSS 98, as major factors. Gross revenues by platform for the quarter consisted of 50 percent for N64, 30 percent for PlayStation, 18 percent for PC, and the balance in portables and all other. North American operations generated 58 percent of net revenues and international operations 42 percent for the quarter.

KESMAI SIGNS ON HENDRICK. Kesmai Studios recently nabbed veteran game designer Arnold Hendrick from Interactive Magic. Hendrick will take the position of senior producer for Kesmai's next generation of Air Warrior combat flight sims. While at IM, Hendrick was the senior designer for PANZER '44 and IM1A2 ABRAMS military sims. His game design career spans over 25 years, and includes classics such as DONKEY KONG JR. for the Atari 2600 and MicroProse's GUNSHIP.

ACTIVISION ACQUIRED HEAD GAMES PUBLISHING, a developer of "outdoor sports and lifestyle" games. Head Games was the company behind the recent hit BIG GAME HUNTER. Activision explained that the purchase follows its strategy to expand its product offerings into new genres at affordable price points (BGH lists for $20), and target a broader consumer audience. Other Head Game titles under development or recently released include ZEBCO PRO FISHING 3D, AMF PROBOWL 3D, DUCKHUNTER PRO, CROSMAN EXTREME PAINTBRAWL, BRUNSWICK PROPOOL 3D, and MASTERCRAFT EXTREME WATERSPORTS. Amazing.

DIAMOND MULTIMEDIA announced worse than expected results for its second quarter (ending June 30), caused largely by flat sales of its Voodoo2-based Monster 3D II card. William Schroeder, Diamond's president and CEO, stated that "…sales of our Monster 3D II product, while quite strong, fell short of our expectations toward the end of the quarter by approximately $20 million. Since Monster II was one of our best margin products in the quarter, this had a significant impact on our total margins for the quarter." Diamond posted a loss of $8.3 million for the quarter. Diamond was one of the first companies with a Voodoo2 card, so this may not be good news to second- and third-tier card manufacturers using the Voodoo2.

Oh My God, I Inverted Kine!

We have all heard about inverse kinematics. It has become a buzzword in computer graphics. High-end 3D animation packages brag about how effectively they handle IK. So IK clearly has something to do with animation, right?

Well, look at it piece by piece. There are two methods for studying motion: kinematics and kinetics. Kinematics is the science of motion without regard to the forces that cause it. If I were interested in how forces and torques act upon an object to create motion, I would be looking into the kinetics side of dynamics. But I don't want to open that can of worms. So for now, let's just stick to kinematics.

Kinematics is really about the geometry of motion. If you read my columns in March through May 1998, you know that when animating a character, it's often convenient to build a skeletal hierarchy that represents the different parts of the character. When animating this character, I keep track of the position and orientation of each of these parts. For example, to move a character's hand into a desired position, I may rotate the upper arm, then the lower arm, and finally lower the hand, until I am happy (see Kine in Figure 1). This form of animation is known as forward, or direct, kinematics (it's forward because you manipulate each joint forward throughout the hierarchy).

But, what if I wanted just to position the hand and let the software calculate a set of joint orientations for the other bones to generate the final position? That's the goal of inverse kinematics. Given a desired position and orientation for a final link in a chain, establish the transformations required for the rest of the chain.

You can see how this is a big plus for animators. By simply dragging around the hands and feet, they can position the entire character. That's why any 3D graphics software that's interested in competing in the animation market must have IK. But, how does this apply to real-time games?

Inverse Kinematics and Gaming

Interactivity is very important in 3D games. Players want the ability to truly interact with their environments. However, this level of interaction is difficult to create. If some of the goals in the game include picking up objects or manipulating switches and levers, then the character needs the ability to visually interact with these objects. To make the problem easier, many game titles create one canned animation for each action. Then, when the character encounters an object that it needs to pick up, there are two ways to handle the action: either the player must line up the character manually to perform the interaction, or the game must align the character with the object automatically. The former technique can lead to frustration on the players' parts as they try to align the character manually. The latter can lead to visual problems if the character is allowed to correct too far. Anyone who has played games such as TOMB RAIDER is very familiar with the issues involved.

Now, these methods are perfectly reasonable cheats that game designers use to avoid difficult problems from either a programming or production perspective. However, if you have the desire and computational bandwidth to spare, it would be good to solve this problem. By implementing an IK system in a real-time game, you can enable the character to reach out interactively for any object within its reach.

Inverse kinematics allows you to create complex characters that face the player. How about a serpent that whips its head around to confront the character, no matter from what direction the character approaches? Inverse kinematics opens up many similar possibilities to game designers. So, now that you're all convinced that you need inverse kinematics in your game, how do you go about doing it?

Taking Animation to the Sixth Degree

I need to take a minute to discuss degrees of freedom. You see statements such as, "A complete six-degree-of-freedom engine," in ad copy all the

FIGURE 1. *Kine application in action.*

Jeff can be freely manipulated about an arbitrary axis at Darwin 3D, for a fee of course. To impose your own restrictions on him, e-mail jeffl@darwin3d.com.

time. But what does that really mean? In my March 1998 column, "Better 3D: The Writing Is on the Wall," I discussed degrees of freedom and how they were affected by rotations. To recap loosely, an articulated figure is connected by a series of joints. Each joint forms the number of degrees of freedom in the next object of the hierarchy. Figure 2A depicts a simple sliding joint like you may see in a shock absorber. This joint, called a prismatic joint, exhibits one degree of translational freedom. Moving the joint only moves the end position in one dimension. Figure 2B depicts a basic rotational, or revolute joint. It allows rotation around one axis defining one degree of rotational freedom.

To solve these problems, you need to be pretty comfortable with trigonometry. If you're like me, your trig is a little rusty. I recommend you get your hands on a good trig book...

In actuality, most joints in a character have more then one degree of freedom. For example, a wrist joint usually allows rotation to some extent in the x, y, and z axes. This represents three full degrees of freedom for the wrist alone. However, when a game engine is described as having six degrees of freedom, this refers to the player's point of view. The player is able to move the camera in all three directions and has rotational freedom about all three axes.

When you're trying to solve the inverse kinematics of a system, you are solving a system of nonlinear equations. Each added degree of freedom makes the problem more complex. This means that each way you can limit the system will make the calculations easier later.

So, How Do You Do It?

In general, there are two forms of solutions for an inverse kinematic system: closed form solutions and numerical solutions. Closed form solutions are found analytically by using noniterative calculations. John Craig has shown that all systems with only revolute and prismatic joints having a total of six DOF in a single series chain are solvable closed form systems (see For further info). To solve a closed form system, one can take algebraic and geometric approaches. The benefit of the closed form solution is that it can be calculated quickly and exactly.

One uses numerical solutions when the system is too complicated for closed form methods. They use iterative calculations to approach an actual solution as closely as possible. Because of the iterative method used, a numerical solution can take much more time to calculate. But the approach solves very complex kinematic systems.

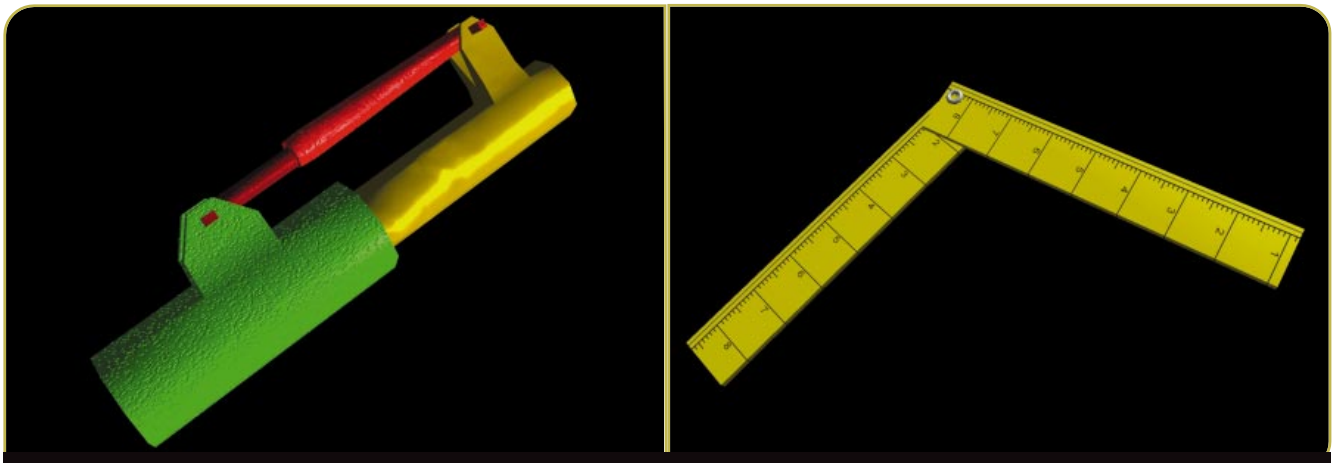
Once More into the Trig

To solve these problems, you need to be pretty comfortable with trigonometry. If you're like me, your trig is a little rusty. I recommend that you get your hands on a good trig book and go through the basic identities and conversion formulas. It will make your descent into the wild world of kinematics a lot less painful. You'll be surprised how much it will help out your 3D programming skills, too.

Let me start by taking a look at the closed form solutions. They're much easier to understand and provide a strong basis for the iterative methods. Take a look at the system in Figure 3. This represents a two-joint articulated arm in a single plane. By restricting the motion to the x,y plane, the calculations are much easier. That doesn't mean it's not an interesting case. A character reaching for an object can be calculated in a single 2D plane and still maintain a lot of flexibility.

The first bone is of length L_1 and is rotated about the origin by θ_1 degrees. The second bone is of length L_2 and is rotated about the local axis by θ_2 degrees. This puts the end position of this system at P. By applying basic trigonometry I know that the position of the origin of the second bone is:

$$\theta_2 = (L_1 * \cos(\theta_1), L_1 * \sin(\theta_1)) \quad \text{(Eq. 1)}$$



FIGURES 2A AND 2B. *Figure 2A represents one translational degree of freedom, and Figure 2B represents one rotational degree of freedom.*

If I then add in the second bone, I get a final position for P.

$$P_X = (L_1 * \cos(\theta_1)) + (L_2 * \cos(\theta_1+\theta_2))$$
$$P_Y = (L_1 * \sin(\theta_1)) + (L_2 * \sin(\theta_1+\theta_2)) \quad \text{(Eq. 2)}$$

This is the formula for the forward kinematics for the system in Figure 3. It represents the two degrees of freedom in the figure. Because of the few degrees of freedom and the restriction to 2D, the formula is not that bad. But what I really want to know is, given a position P, what values for θ_1 and θ_2 do I need to solve the equation?

One key piece of math that I'm going to pull out of my rusty mind is a couple of basic trig identities.

$$\cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b)$$
$$\sin(a+b) = \cos(a)\sin(b) + \sin(a)\cos(b)$$

In order to finish it up, I need to square both parts of Equation 2 and add them together, applying my trig identities along the way. This gives me the following:

$$x^2 + y^2 = L_1^2 + L_2^2 + 2L_1L_2\cos(\theta_2). \quad \text{(Eq. 3)}$$

I can now easily solve for θ_2.

$$\cos(\theta_2) = \frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2} \quad \text{(Eq. 4)}$$

The angle is obtained by inverting the cosine operation.

$$\theta_2 = \text{Acos}\frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2} \quad \text{(Eq. 5)}$$

By solving for θ_1 using Equation 2 and the identities, you get the final piece of the puzzle.

$$\theta_1 = \frac{-(L_1\sin(\theta_2))x + (L_1 + L_2\cos(\theta_2))y}{2L_1L_2} \quad \text{(Eq. 6)}$$

That's all there is to it. It's clear that if there were many more degrees of freedom, this technique would be impossible. But for this problem, I'm off and running. Equations 5 and 6 give me all I need to code a solution to the system in Figure 3.

LISTING 1. *Compute an IK solution to an end effector position.*

```
////////////////////////////////////////////////////////////////////////
// Procedure:   Compute IK
// Purpose:     Compute an IK Solution to an end effector position
// Arguments:   End Target (x,y)
// Returns:     TRUE if a solution exists, FALSE if the position isn´t in reach
////////////////////////////////////////////////////////////////////////
BOOL COGLView::ComputeIK(CPoint endPos)
{
/// Local Variables ///////////////////////////////////////////////////
  float l1,l2;          // BONE LENGTH FOR BONE 1 AND 2
  float ex,ey;          // ADJUSTED TARGET POSITION
  float sin2,cos2;      // SINE AND COSINE OF ANGLE 2
  float angle1,angle2;  // ANGLE 1 AND 2 IN RADIANS
////////////////////////////////////////////////////////////////////////

  // SUBTRACT THE INITIAL OFFSET FROM THE TARGET POS
  ex = endPos.x - (m_UpArm.trans.x * m_ModelScale);
  ey = endPos.y - (m_UpArm.trans.y * m_ModelScale);

  // MULTIPLY THE BONE LENGTHS BY THE WINDOW SCALE
  l1 = m_LowArm.trans.x * m_ModelScale;
  l2 = m_Effector.trans.x * m_ModelScale;

  // CALCULATE THE COSINE OF ANGLE 2
  cos2 = ((ex * ex) + (ey * ey) - (l1 * l1) - (l2 * l2)) / (2 * l1 * l2);

  // IF IT IS NOT IN THIS RANGE, IT IS UNREACHABLE
  if (cos2 >= -1.0 && cos2 <= 1.0)
  {
    angle2 = (float)acos(cos2);          // GET THE ANGLE WITH AN ARCCOSINE
    m_LowArm.rot.z = RADTODEG(angle2);   // CONVERT IT TO DEGREES
    sin2 = (float)sin(angle2);           // CALC THE SINE OF ANGLE 2

    // COMPUTE ANGLE 1
    angle1 = (-(l2 * sin2 * ex) + ((l1 + (l2 * cos2)) * ey)) /
             ((l2 * sin2 * ey) + ((l1 + (l2 * cos2)) * ex));
    m_UpArm.rot.z = RADTODEG(angle1);    // CONVERT IT TO DEGREES
    return TRUE;
  }
  else
    return FALSE;
}
```

I Can't Reach that Far

Another important issue in an inverse kinematic system is the idea of reachability. Given a position P, is it possible for the figure to reach that spot? A nice side effect comes out of Equation 4. If the value of the division is not in the range of -1 to 1, then the point is not reachable by the figure. At this point, I can bail out and avoid the rest of the calculations.

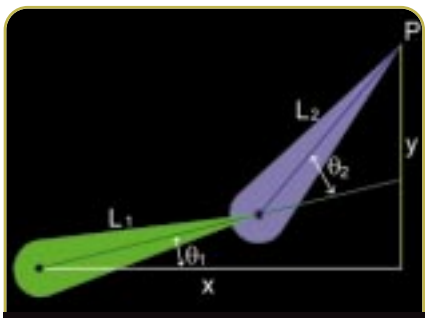


FIGURE 3. *Closed form solution 1.*

Another method for checking whether the goal is reachable is to see if the distance to the goal point is less than or equal to the sum of the lengths of the joints. This illustrates an important point. When solving a kinematic problem with analytical methods, it's not always possible to find a solution that's close enough. Sometimes you don't want close. You only want a solution if it's correct. But if you would prefer your system to be as close as possible, an iterative numerical solution is probably better.

When solving a kinematic problem with analytical methods, it's not always possible to find a solution that's close enough.

Bring on the Code

Using these formulas in an application is pretty easy. There are a couple of things to remember. The formulas assume that the base of the figure is at (0,0). In the case of a character, this may not be true. In my application, I subtract the base offset from the desired end position. This makes things work out quite nicely. The other issue is that the trig functions in C require radians. If your animation system or API requires degrees, an extra conversion step is required. By using lookup tables for the trig functions, or an animation system that handles radians, this conversion can be eliminated. However, on current PC systems, this is probably not an issue because the calculations are relatively minor.

You can see the algebraic solution to my inverse kinematic problem in Listing 1. The routine sets the rotation values of each joint in degrees if the target position is in reach. You will notice the reachability test right in the middle of the listing.

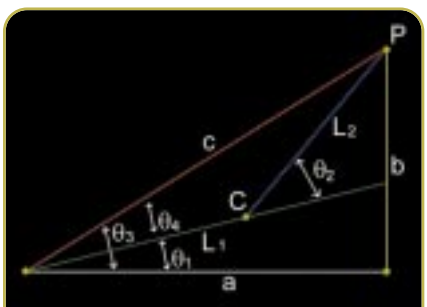


FIGURE 4. *Closed form solution 2.*

Another Closed Form Solution

What I just went through is known as an algebraic strategy for the closed form manipulator. Another strategy for solving the closed form is the geometric solution. The strategy is to break the problem down into a couple of plane geometry problems. The problem is framed in Figure 4. The strategy is to create the line C that extends between the origin and the target position. We can then make use of the law of cosines to solve for angle θ_2.

The law of cosines states

$$c^2 = L_1^2 + L_2^2 - 2\,L_1L_2\cos(C).$$

I can substitute $180 - \theta_2$ for C, leaving

$$c^2 = L_1^2 + L_2^2 - 2\,L_1L_2\cos(180 - \theta_2).$$

Applying the trig identity of the sum of cosines and the facts that $\cos(180) = -1$ and $\cos(-\theta) = \cos(\theta)$, I can substitute $\cos(180 - \theta_2)$ with $-\cos(\theta_2)$. This yields

$$c^2 = L_1^2 + L_2^2 + 2\,L_1L_2\cos(\theta_2).$$

You will notice that this is the same as Equation 3. The same algebra is applied to get the value for θ_2. To solve for θ_1, I need to find the angles θ_3 and θ_4. θ_3 is easy.

$$\theta_3 = \text{Atan2}(b,a)$$

By applying the law of cosines again, I can solve for θ_4.

$$\cos(\theta_4) = \frac{a^2 + b^2 + L_1^2 - L_2^2}{2L_1c}$$

The inverse cosine is calculated so that θ_4 is between 0 and 180 degrees. Then the angles are combined. They are added if θ_2 is less than 0 and subtracted if θ_2 is greater than 0.

$$\theta_1 = \theta_3 + \theta_4 \quad // \ \theta_2 \text{ is less than } 0$$
$$\theta_1 = \theta_3 - \theta_4 \quad // \ \theta_2 \text{ is more than } 0$$

I didn't provide code for the geometric solution, but feel free to try it out for yourself.

The Application

The sample application this month is a 2D inverse kinematic solver for a two link manipulator. If you click anywhere on the screen, Kine will try and reach it. If the point is in his reach, the solution is displayed. If the point is not in reach, a message is displayed. The application uses much of the same framework as my previous articles. One difference is that the display is an orthogonal view. This works well for 2D displays.

We have the basics of inverse kinematics out of the way. Next month, I'll attack the more difficult problem of solving arbitrary hierarchies using an iterative numerical strategy. Until then, check out the source code and application on the *Game Developer* web site. ■

FOR FURTHER INFO

Craig, John J., *Introduction to Robotics: Mechanics and Control.* Second Edition. Reading, Mass.: Addison-Wesley, 1989. This is a very good book on robotics. It provides analytical solutions for many different types of robotic manipulators.

McKerrow, Phillip John. *Introduction to Robotics.* Reading, Mass.: Addison-Wesley, 1991.

Watt, Alan and Mark Watt. *Advanced Animation and Rendering Techniques.* New York, New York: ACM Press, 1992. Yes, I used it again. Get the hint and get the book.

Heineman, E. Richard. *Plane Trigonometry with Tables.* McGraw Hill, 1956. An older trigonometry book that I picked up a while ago. If you are working on 3D graphics, you need a book like it.

The Chip Industry

At last May's CGDC, industry analyst Rob Glidden delivered an informative presentation on how chip technology and digital television standards are converging to create media processors that may just lay the foundation for future gaming platforms. Or, maybe not.

It depends on you, the kings of content. No, really. You are the kings of content. The killer app for interactive television is gaming, it's just that no one knows what that means in practice. We just know that we can cram more features and functions onto smaller pieces of silicon, thereby delivering more bang for the buck, and we know that once you have digital television in place, you have a big stream of digital data going into every home. Which leads one to think: isn't the Internet just one big stream of digital data? That's more grist for the convergence mill. So, over the course of the next couple of years, consumer electronics companies, PC makers, and anyone else with a yen to make a set-top box, are going to spew media rich processors and devices. Television content is going to change dramatically as a result (sorry). So, who is going to make the digital entertainment of the future? Is it you? Perhaps it should be. This latest incarnation of the convergence bug is as inevitable as Moore's Law.

Moore's Law and the Darwinian Process

What Moore's Law successfully predicted, and what has engaged the minds of chip designers since, is that every 18 to 24 months we can expect to cram twice as many transistors onto the same area of silicon. The amount of silicon real estate that you use determines the cost of a chip, and an increase in transistors actually correlates to an increase in chip performance and throughput, which also results in the mistaken assumption that Moore's Law just says that chips get twice as fast every two years. It's not that simple. Gordon Moore was really concerned with the way chips are manufactured, and his pronouncements helped Intel to create a simple and highly effective business model: build bigger and better chip fabrication plants, or fabs, to build bigger and better processors. As long as you sell the bigger and better processors, you can invest in another set of new fabs, and start the cycle again.

This fact has not gone unnoticed by the other chip companies. It's just that none of them had the foresight, or luck, to be in the PC business. So, maybe they're not as big and successful as Intel right now, but these other chip companies are looking for their own monopolies — and a golden platform to call their own. They have to get their turn, and as the millennium draws to a close, they just might have the opportunity. One of the main reasons for this ray of competitive hope is that chip makers are reaching a plateau in terms of what can be done with new chip fabrication processes and how much functionality they can cram onto a piece of silicon before losing touch with reality. After all, it isn't easy to keep track of millions of transistors. So, problem one with new chip technologies is going to be design. Chip designers have to find new ways in which to handle the enormous complexity of connecting millions of transistors, maybe tens of millions of transistors. No one company has the answer.

Problem two has to do with the physical problems of creating denser chips. As a result of the need for atomic level machinations, chip manufacturing costs billions of dollars to implement. In the seventies, an Intel fab cost somewhere between five to ten million.

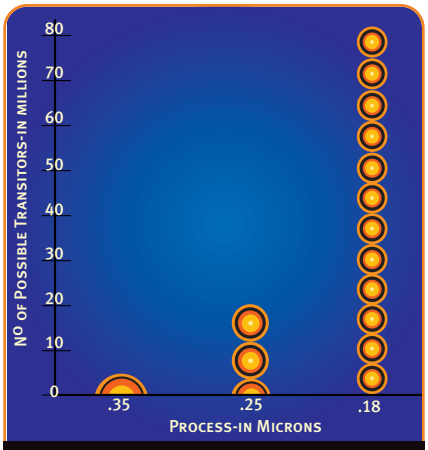


FIGURE 1. *The evolution of processors.*

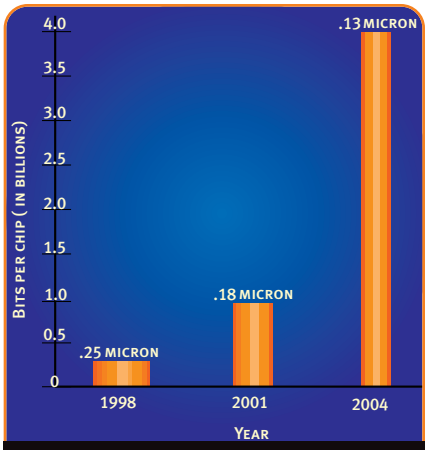


FIGURE 2. *The projected evolution of DRAM.*

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.

That figure is now in the billions of dollars (it seems that Moore's Law can also be applied to the cost of keeping pace with the technology).

Problem three is the man-years of effort that have to go into solving problems one and two. More complex parts and more complex processes require more people to make production happen. If that wasn't enough, not only does playing keep-up with Moore's Law get more expensive and difficult as time goes by, but so does finding ways of using the extra horsepower it brings to chips. It's one of the problems that the PC industry has right now. Outside of real-time 3D games, there isn't much else of mass market appeal to drive consumer adoption of Intel's high-end desktop processors.

THE EVOLUTION OF PROCESSORS. Hand in hand with evolutions in processors, we also find that we get commensurate increases in memory speed and capacity. Not only do we get bigger, better processors, but we can stick a lot more memory around them and run bigger, better programs. One obvious beneficiary of this situation has been 3D graphics. For instance, Nvidia's RIVA TNT and 3Dlabs' Permedia 3 will both be manufactured in 0.25 micron processes by the end of 1998.

As a result, graphics board vendors are pretty much resigned to having their sweet-spot products based on these chipsets supporting 8MB of memory as standard, with a small price delta to move up to 16MB versions. Expect street prices for these boards to be in the $150 range. Not bad for the consumer, although the cut-throat pricing just makes you wonder how these chips can get better, and cost less, and support so many innovators. Well, the fact that Nvidia and 3Dlabs are hitting the same point in the performance curve as 3Dfx, and doing so at extremely competitive price points, is not only because of the expertise of these companies, but also because they have positioned themselves to take advantage of the extra transistors available to them this year. The same is true of Matrox, S3, and ATI. But 3Dfx won't get left behind — they will be in 0.25 micron mode by the first half of 1999. This leapfrogging in the graphics industry is a simple example of how chip companies can parlay their knowledge of Moore's Law into a competitive catch-up dance.

The other implications of transistor cramming are already appearing in Intel's MMX. In the near future,this technology will be part of AMD's 3DNow and Cyrix's integrated graphics. Furthermore, Intel's Whitney, which will put rasterization functionality into core logic (thereby binding some of the 3D pipeline with the processor), also blurs the line between computers and low-cost multimedia set-top boxes. Obviously, Intel and AMD are attempting to extend their processors' reach and reduce the reliance on expensive external peripheral components (such as graphics and audio chips). Some cynics might suggest that with the lack of compelling content that requires high-end processors, these chip companies have no choice but to eat the lunch of other components on the motherboard, and compete on price. The reality is that chip manufacturers have no choice but to take over the real estate of the system from other parties in order to drive the platform, and also reduce costs. Is it any wonder that Rendition was bought by Micron, who had just announced its purchase of Texas Instruments' memory business? Chip

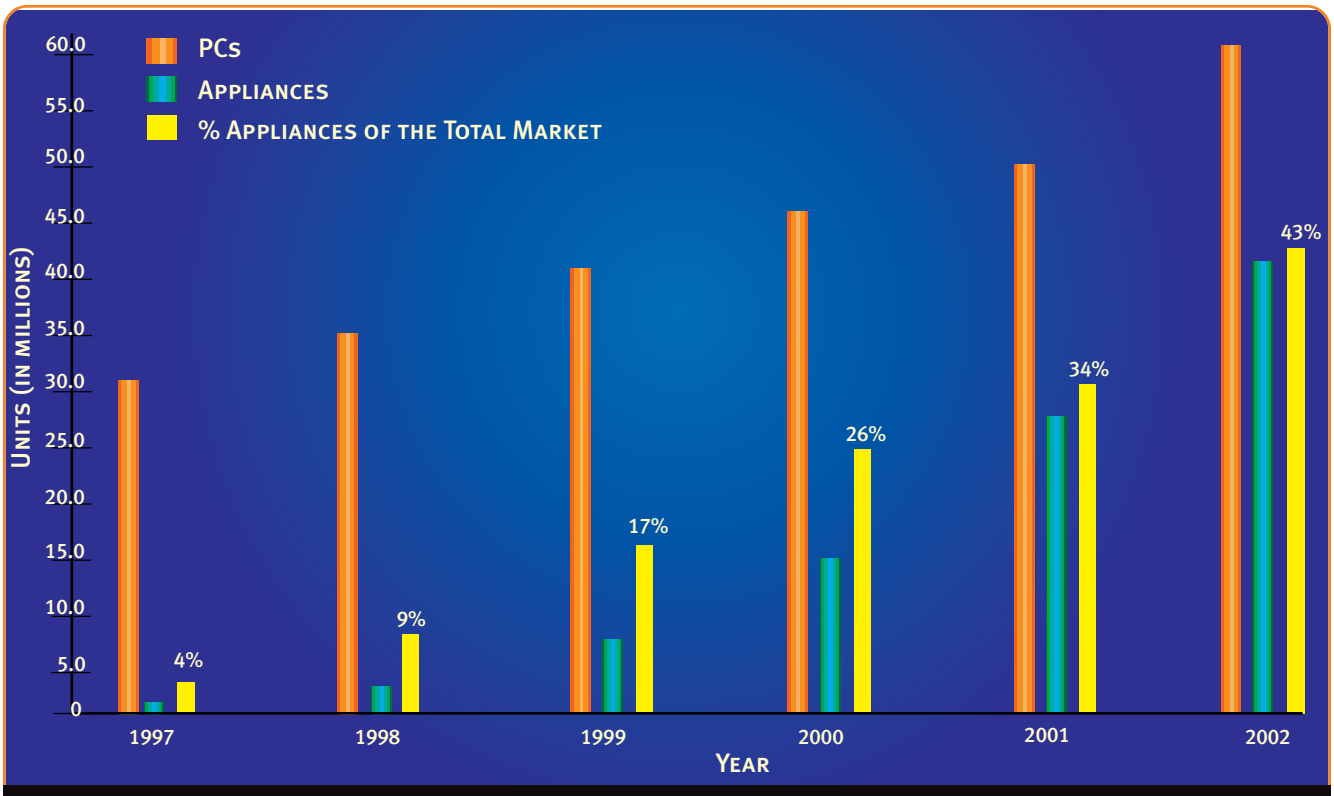


FIGURE 3. *Forecast of U.S. unit shipments of Internet appliances versus PC shipments (Source: IDC Research).*

makers are looking at integration as one way of making the most out of the chip real estate they own.

THE EVOLUTION OF DRAM. In the last issue of this column, I briefly mentioned VM Labs. VM Labs' business plan is simple — add 3D games capability to DVD players, and eventually DSS receivers, or anywhere there is hardware for MPEG-2. Console makers have noticed that the number of transistors that will do MPEG-2 decoding on set-top boxes is enough to do a good job on real-time 3D. Nintendo, Sony, or Sega can just as easily add MPEG-2 decode support to their boxes. In fact, it's rumored that PlayStation 2 will be just such a device, although Sony doesn't seem convinced that such a convergence is necessary or practical for its gaming audience. However, VM Labs provides some compelling (although not unique) arguments — and these arguments will only become more compelling as we move towards 0.18 micron technology, and tens of millions of transistors per chip at little cost. Yet, the question remains for these alternative set-top boxes. Where is the content going to come from?

Nothing but Internet

Despite the fact that semiconductor technology issues point towards a future of low-cost, media-rich platforms that can handle 3D and MPEG-2 with equal aplomb, neither would be as interesting to the game development community as these potential platforms' impact on the Internet. At present, the PC is the dominant access point to the Internet. However, market research indicates that the future is not so PC-centric. The Internet may provide the answer to the content needs of these media rich platforms. It's a standard that everyone can adhere to and understand, and there's enough content out there to keep most consumers happy right now, especially those who currently have no PC access.

But the novelty of surfing the Web will wear thin very quickly unless the entertainment quotient rises a lot more. That's where the game development community has a role — although that may not sit well with the existing power structures. At present, online gaming is pretty much a hardcore, or enthusiasts', activity. As it gains in popularity, it seems to get even more hardcore — just take a look at some of the QUAKE clans on the Web. These online gaming communities are just one or two channels of interactive entertainment. The raw power of the chipsets within set-top boxes is going to be sitting in people's living rooms, where the biggest audience is not hardcore gamers, but television-watchers. Electrifying the couch potato should be the mantra of game developers who want a piece of this action.

In effect, digital television and the Internet are the wires into people's homes, and the new breeds of set-top boxes are enabling technologies. It's somewhat akin to the emergence of cable programming as a viable alternative to the big three networks. This new cable revolution won't be just about gaming (maybe there's room for the Quicken Channel), but the opportunities are going to be there. Did I say, as inevitable as Moore's Law? I did. You can't stand in the way of progress, but you can certainly catch a ride on it as it rolls by. ■

Back

DirectMusic

DirectMusic for the Masses

by Tom Hays

DirectMusic is a complete overhaul of the way that Windows plays music. It replaces the basic code that Windows applications use to get MIDI data out of a file, through the computer, and to the output device. It's a completely rewritten and rethought system, all the way down to what noises come out and how.

Has it arrived too late? Now that so many games are using Red Book CD audio and other streaming mechanisms, is DirectMusic enough to make MIDI relevant again to game developers? I think so — no matter how you currently handle music in your games, DirectMusic is definitely worth checking out.

Illustration by Ben Fishman

Tom Hays serves as audio director for a mid-sized suburban game company. He loves his kids and likes his beer. For more scintillating details, write tomhays@dnai.com

DirectMusic starts out by addressing the major problems of Windows' old `MidiOut` API, such as shaky timing and limited real-time control. It offers consistent playback of custom sound sets using an open standard, Downloadable Sounds Level 1 (DLS1). On top of that, DirectMusic opens more than one door to achieving adaptive musical scores in games.

Like other SDKs from Microsoft, DirectMusic will try to cover many bases, not all of them related to games. Most observers agree that it has some great solutions for background music on web sites. Are DirectMusic's approaches relevant to game development? Its depth makes for a massive API, with hundreds of pages of documentation. Is it too complex to use on a project with a deadline?

This article is an overview of a big piece of work that is still in alpha, so don't look at it as a review. It's more of a look at what DirectMusic is and what it isn't, to help you get some idea of whether it fits your needs. The SDK should be in beta as you read this, and will be released as part of DirectX 6.1 late in the year.

What is DLS?

Downloadable Sounds Level 1, or DLS1, is a nonproprietary specification made by the Interactive Audio SIG of the MIDI Manufacturers' Association. It lays out an architecture and file system for a really basic sample-based synthesizer.

DLS was motivated by developers' desire to be freed from the constraints of the fixed palette of sounds found in General MIDI wavetable ROMs. Another problem with General MIDI synthesizers is the fact that their musical response is inconsistent — the General MIDI specification failed to carefully specify things such as exact envelope responses and volumes. What's more, the synthesizers often just plain sound bad. This experience made game sound developers want a system by which their music would sound the same on every user's system. And of course, the hardware folks wanted to make MIDI relevant again in a new way, so that they could develop and sell a new generation of hardware.

DLS was written with acceleration in mind. Since the working group that hashed out the specification was primarily made up of hardware vendors, no existing sound cards could get shut out by setting the standards too high. The idea was to make something that would run on Creative Labs' AWE32 and other early 1990s sound cards and then get their manufacturers to write DLS1-compatible drivers. Unfortunately, developing and ratifying this took from 1994 until 1997, and the drivers for existing hardware never materialized.

Where DLS1 actually did emerge was in 1997's Microsoft Software Synthesizer, the Miles Sound System API, and in drivers for PCI chipsets such as S3's SonicVibes. Now that DirectMusic is in alpha, several makers of PCI-bus sound hardware are making Windows driver model (WDM) DirectMusic-compatible DLS drivers.

While the world waits for everyone to get these new sound cards and drivers, DLS will largely rely upon the new version of the Microsoft Software Synthesizer supplied with DirectMusic. It's bare bones DLS1 (no filtering), with reverberation added. This simplicity lets it claim only 0.35 percent of the CPU per voice in a Pentium 166MHz MMX, meaning that a song with 10 voices playing at any given time would take up 3.5 percent of the CPU's cycles. On a Pentium II 266MHz, these figures drop to 0.12 percent per voice. (These figures are for the reverberationless current release, running at 22KHz 16-bit stereo output.)

Of course, these figures will drop to almost nil on systems that have DirectMusic-compatible acceleration of DLS. There are already sound chipsets on millions of computers, from ESS, S3, Analog Devices, and others, that already use DLS via proprietary drivers. Standardized WDM drivers for these will put DLS acceleration into many existing machines with just a driver update.

A common argument against software synthesizers is that they require memory for holding instrument samples. Unlike the AWE32 and AWE64, which had on-card RAM, these new PCI accelerators use system RAM and move everything across the PCI bus. Whether or not a developer uses custom samples, the same overhead is there. They have no ROM-based wavetables. With modern hardware and WDM drivers, if you're going to use MIDI at all, it might as well be custom DLS.

CONTINUED ON P. 23.

Way Back When Down South...

Back in the late 1980s and early 1990s, there lived in Atlanta, Ga., a team of imaginative, talented music programmers called the Blue Ribbon Soundworks. They made a MIDI sequencer for the Amiga called Bars & Pipes which was so innovative that some people still keep an Amiga around just to run it.

By 1994, Blue Ribbon's main focus was a technology called AudioActive, part of which saw the light of day in music-generating programs such as SuperJam and Audiotracks Pro. AudioActive was an API and toolset that generated MIDI music performances on the fly by using data types called styles and personalities. At AudioActive's heart was a toolset for breaking compositions into their component parts and an engine for putting them back together.

To design this system, Blue Ribbon examined the way that real performers in various musical genres make the decisions that affect the progress of a piece. The system bore some conceptual similarity to musical Markov chains, in which each note has a weighted probability of going to each other note. But AudioActive was quite a bit more complex, subtle, and, in true musician form, more subjective. In some cases, it was able to create very convincing performances.

From Atlanta to Redmond

This pedigree was the first thing that I, and many other developers, heard about Microsoft's new music system. It made us skeptical from the outset. Microsoft's developer-hype literature still emphasizes DirectMusic's real-time music generation aspects to such a degree that it looks like AudioActive: The Sequel. Despite automatic music's enormous gee-whiz factor for us computer music types, I couldn't help but feel that its real-world use would simply be one more

What is DLS?(Cont.)

DLS2 is being written for a much more advanced generation of hardware. Nothing has been announced, but common sense would dictate that it will plug the obvious holes of DLS1. The main new features will likely include dynamic filtering attached to envelopes and low frequency oscillators (LFOs), reverberation, and better layering abilities. It will remain a nonproprietary standard, letting a DLS2 collection play on any hardware — Windows or not — that has the right drivers. It's not unrealistic to expect to find DLS1 or 2 showing up attached to specifications such as QuickTime, .AVI, or MPEG, and in various web and console applications.

In the meantime, DLS1 is very much worth checking out for most game development teams that use in-game music. As of this writing, the alpha version of the Microsoft Software Synthesizer was missing some key features such as reverberation, decompression (through Windows' Audio Compression Manger), and the ability to play large samples. Therefore, I can't assess what it can really do for a number of real-world game applications.

If it works as advertised, it will support many interesting techniques for delivering music in a standardized format. Except for quality variance in users' speakers, amplifiers, and digital-to-analog converters, DLS files will sound exactly the same wherever they play.

Alternatives

In CPU-intensive applications (read: most games) that wish to use MIDI and can tolerate ROM-based General MIDI, installation or startup code can check whether or not the system has DLS acceleration or an old-fashioned ISA card with fixed General MIDI. Then it can choose an output device, striking its own balance of speed vs. sound quality.

Developers who want a richer synthesis model than DLS Level 1, and are willing to take an additional CPU hit, will find other vendors' software synthesizers making themselves available as replacements for the Microsoft synthesizer.

Shark Food?

There is one red flag waving over widespread use of DLS: sample copyrights. Some providers of musical instrument samples are getting nervous about having versions of their samples floating around in DLS collections.

This makes sense in the context of making DLS collections to publish on the 'net and share with your buddies. For game developers, though, common sense would dictate that this is a non-issue. Just fold your DLS collections into a resource file along with the rest of your copyrighted art and sound, so that only dedicated hackers have a chance of pulling out the data. To be even safer, encrypt the data. This way, the data presents itself in exactly the same way that it would if it was recorded on a music CD: mixed in with other instruments in a stereo output stream.

Beyond this, the revenue models for selling samples to developers and to hobbyists are completely different. When a vendor sells a sample to a developer, it usually costs a great deal of money and comes with full rights to use the sample in a piece of music or audio-visual product for sale to the masses. If one of these samples leaks out to the public and is used as a Windows startup sound, the loss to the creator is commensurate with the cost of one sample on one of those "1,001 Whacky Windows System Noises" CDs — maybe ten cents per user, as opposed to two bucks or more for a sample off of a professional library.

However, lawyers aren't known for their common sense. They are known for going to great lengths to maximize the revenue of their employers, period. Hopefully, sample providers will come up with a licensing scheme or an industry agreement for acceptable use. Also, companies might start coming out with DLS collections, which will force them to figure out licensing. Until this happens, if developers start including samples straight off of commercial sources such as CDs — or even ROM-based professional synthesizers — without getting DLS-specific permission, someone might sue. Then, whoever has the most money for lawyers will make the rules.

This is no reason not to use DLS. Create new samples and get permission for any commercial samples you use. If you're feeling daring, don't follow these guidelines, but encrypt your data (neither the author nor *Game Developer* magazine takes any responsibility for any legal consequences of actually following my advice, however).

way for a skinflint producer to avoid paying for professional composition.

But over the past three years, plenty has happened. The trademark "AudioActive" is now used for an MPEG-2 audio player from Germany. Blue Ribbon Soundworks was purchased by Microsoft in late 1995, and its principals moved from Atlanta to Redmond. Its development lead, Todor Fay, continues to spend many days in trade group meetings small and large, listening to what people who make and support music for interactive products want and discussing his team's ideas.

Fay apparently doesn't like to say no. DirectMusic incorporates a truly frightening number of features, including some of the features that developers have been asking for in a music API. It includes an evolved, 100-percent rewritten version of what used to be AudioActive. It also includes hooks for replacing, adding, or modifying any component in the entire system with whatever music generator or filter you or a third party might come up with on your own. It gives applications access to MIDI and other control data in real time.

The fact remains that this open architecture was written with a certain approach to adaptive music at its core. It's an odd thing to find in a Microsoft API: a highly involved music recombiner and regenerator — neat thing, but not the right solution for everybody. Sometimes, in poking through the SDK with a different need in mind, a developer will be mystified and frustrated at some of the approaches and some of the omissions. However, DirectMusic does try to offer solutions for those who don't wish to use the System Formerly Known as AudioActive. Thus far, Microsoft's publicists have done themselves and developers a disservice by giving the impression that the interactive music engine is the core reason to look at DirectMusic. This is definitely not true.

In its alpha stage, DirectMusic is such a big package that many people's first impression is that it's just too complex to use on a typical project. One of the things I set out to do in researching this article was to see if there were reasonably simple paths to solutions for common problems buried in the over 300 pages of API documentation. Microsoft needs to do this if they want to sell

DirectMusic to the game development community; as DirectMusic approaches beta (early July), the company is rewriting the documentation with this approach in mind.

DirectMusic's Innards

DirectMusic's headlines for most people who make games are DLS support for hardware acceleration and MIDI with over a million channels and rock-solid timing. It's a big package, consisting of several major parts that operate on different levels. It's not necessary to use or even understand all of the components to make good use of parts you need.

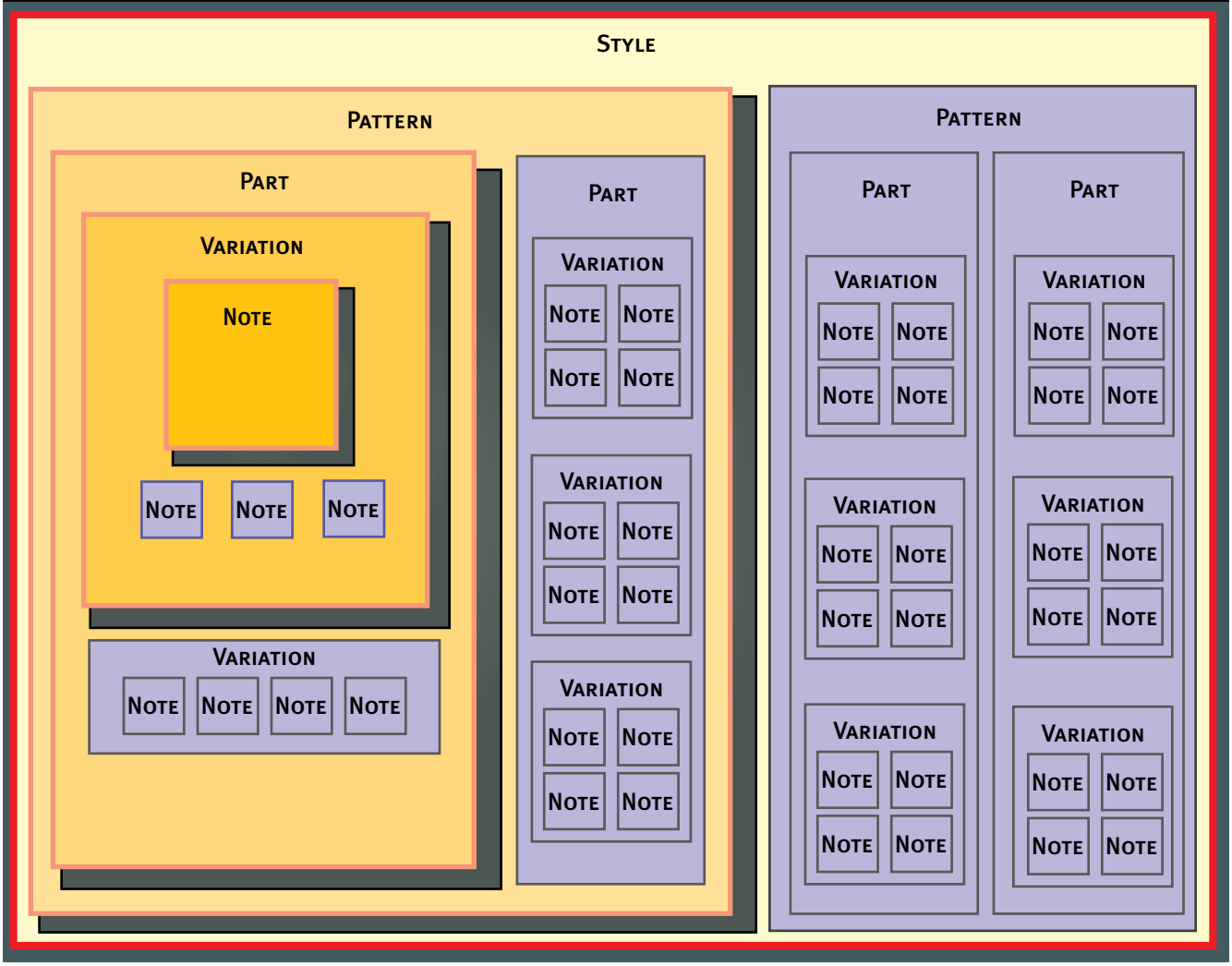
For starters, DirectMusic replaces Windows' MidiOut technology with a new model. DirectMusic's MIDI support has subsample timing accuracy, allows flexible selection of output ports (including third-party creations), and lets applications inspect, filter, and modify MIDI data as it comes out. The release version will also multiply MIDI 1.0's 16 channels by a healthy 65,536, for a total of 1,048,576 discreet channels (called `pChannels` within DirectMusic).

The biggest single claim that DirectMusic has to making MIDI relevant again is its support for DLS. According to its developers, the bundled Microsoft Software Synthesizer was using 0.12 percent of the CPU per voice on a Pentium II 266MHz MMX as of late June. These numbers will get a bit worse when reverberation is added (reverberation wasn't included in the API as of this writing, but is scheduled to happen before final release). Under the Win32 Driver Model in Windows 98 and Windows NT, this is open to hardware acceleration by PCI-bus sound cards.

DirectMusic includes a Roland-made General MIDI/GS sound set. However, the really great thing about DLS is that it opens up MIDI in games to a variety of techniques for using samplers that electronic musicians have built up over the years. These range from basic wavetable-style techniques (but with any choice of sample data) to sampling entire musical phrases and triggering them via MIDI commands.

If MIDI is a dirty word for many game developers, it's not because of MIDI itself, which is simply a control mechanism and has no intrinsic sonic quality, good or bad. It's because of the inconsistent, usually low quality, fixed sample

FIGURE 1. *Elements of Style: Notes within Variations within Parts within Patterns within Styles. Many Properties can be set at each level or allowed to use to value of its parent.*



sets in the built-in synthesizer ROMs on most sound cards. DLS lets MIDI go back to being a timing, control, and note-triggering mechanism, as opposed to being a synonym for crappy-sounding game music . When MIDI is freed up to do its stuff, it can provide the granularity, malleability, and reaction time needed to make music react to what goes on in an interactive world.

As I mentioned, DirectMusic was conceptually built up from its specialized music-digesting system, the most controversial and confusing part of the SDK. This system is good at its original purpose, but that's not the whole story. It has a big side-benefit: a way to play and control segments of MIDI data, apply tempo maps and data filters, and concatenate them into other segments at musically-appropriate junctures (Figure 1). APIs such as the Miles Sound System (see Andrew Boyd's review on page 51), HMI's Sound Operating System, and DiamondWare's STK have already been doing this sort of thing (and more) under Windows despite MidiOut's limitations. All of these SDKs' developers are likely to be able to do more interesting things more reliably under DirectMusic.

Segments, Tracks and Tools

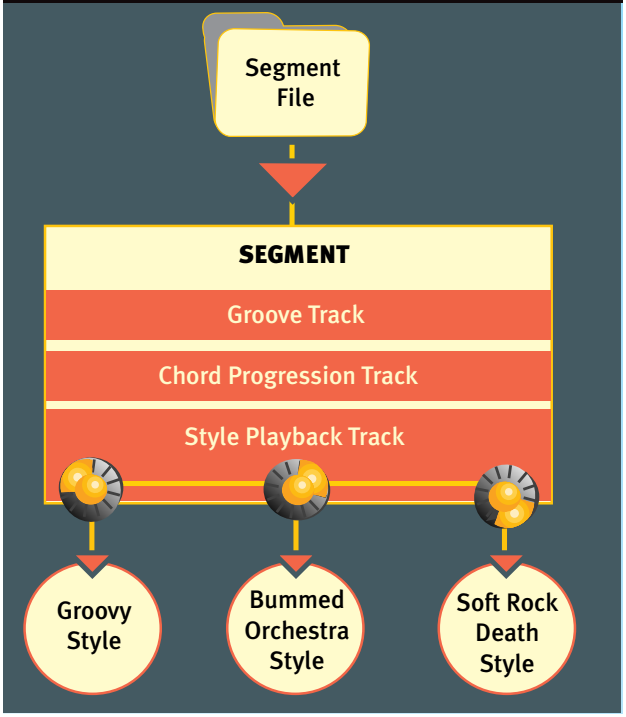
DirectMusic's essential playback unit is the track. Tracks are contained inside segments (Figure 2). Typical examples of tracks and segments would include:

- an imported MIDI file, which the `segment` object splits into three tracks containing notes, tempo, and time signature information (Figure 3);
- a style playback segment that points to one or more styles, which are compositions that have been abstracted to some level and can change based on real-time input;
- groove, chord map, and signpost segments that the interactive music engine can use to generate style playback segments;
- special-purpose segments, such as a mute segment, that can automate playback by turning channels on and off.

For those who wish to do complex things with music that can't be done with the built-in generation system, DirectMusic is built to be extended. For starters, tracks and segments are an extensible data type. Because they are the core playback unit, they will let Microsoft and third-party vendors address any fundamental complaints from developers.

DirectMusic also incorporates objects called Tools, which are intended to be easy for developers or third parties to write. These sit in what's called a tool graph, which makes all tools present cooperate with one another. A tool can operate on just one logical chunk of music (a segment) or can process the entire output. If DirectMusic catches on, expect to see scads of tools written to plug its holes, such as a MIDI channel and note mute mask, a MIDI echo, a velocity modifier, a quantizer/dequantizer, and so on.

FIGURE 2. *Basic style playback using a prebuilt segment file. The Style Playback Track can point to different Styles as playback progresses.*



For hardware vendors who want to extend the API to include new capabilities, DirectMusic provides a mechanism called the property set. Each of these is tied to a Global Unique ID (GUID), and each gets its own index of individual properties, indexed from 0. A given attribute index for a given GUID is always the same. For example, let's say that a developer has built an interface and drivers to hook a real siren to the parallel port. In order to integrate the device's API into DirectMusic, the developer would publish the GUID of the "DirectSiren," along with its indexed property set. An application supporting DirectSiren could then use DirectMusic's `IKsPropertySet` interface to see whether or not the DirectSiren's `DeafeningAirRaid` property is available.

Programming with DirectMusic: A Smorgasbord of COM objects

DirectMusic consists of 24 distinct COM objects. This lets developers use only the portions they need. For example, if you just want MIDI output, you don't need to incur the overhead of DLS or the learning curve of any interactive music code.

It also means that developers can replace entire sections of the system with ones that meet their needs. The idea is to make an architecture robust enough that third-party vendors of related products and tools will have a much easier time, and won't need to reinvent the wheel in order to support the code they really want to provide. For example, Headspace is making a version of its web-based music player/generator Beatnik that integrates the DirectMusic API.

The DirectMusic Loader

At DirectMusic's technological core lies the Loader, responsible for locating, loading, and registering objects. It was designed with low-bandwidth applications in mind, so it strives for efficiency.

To use the Loader, generally the first step is to set a search directory. This isn't required; an object can be referenced by full path name. URLs are not yet supported. Once a search directory is set, the Loader can search for objects

using its **ScanDirectory** method and enumerate them in a database of their names and GUIDs.

The Loader's caching system relies upon this database: if an application asks for the same object twice (even in different locations), and if that object is in the database, it doesn't need to be loaded a second time. Caching is enabled for all objects by default, but can be turned off and on with the Loader's **EnableCache** method. For a balance between conserving RAM and avoiding repeated loads of the same object, an application must make smart use of the **CacheObject**, **ReleaseObject**, and **EnableCache** methods. Of course, there will be cases where caching is not good — browsing through tons of instruments in a DLS editing application, for example.

Once this database exists, an application can use the Loader's **EnumObject** method to show all objects of any class or classes in the database and then make an instance of the object (without duplicating data) using the **GetObject** method.

Output API: Instruments and Ports

The basic means by which DirectMusic makes actual sounds come out of the digital-to-analog converters of a game player's PC is DLS. The API represents DLS instruments with the **DirectMusicInstrument** object, and sets of instruments (Collections and Bands) with the **DirectMusicCollection** and **DirectMusicBand** objects. The way any of this gets out of the box is via a **Port** object.

To use DLS in an application, first you must have one or more files full of DLS instruments, comprising both sample data and associated control (articulation) data. This industry-standard (not just Microsoft's) file type is known as a DLS Collection. As a simple enhancement to General MIDI, you can use the General MIDI/GS DLS collection bundled with DirectMusic. This way, all of your users will hear the same sounds, and you won't play sound-card roulette.

Of course, using the stock GS set sort of misses the point of using custom sounds. The better way is to have your crack team of composers and sound designers deliver custom DLS collections comprising instruments specifically developed to go along with your game's music. Sound designers can also supply DirectMusic bands, which are detailed references to DLS data in one or more collections (Figure 4).

Basic Playback with the Performance API

DirectMusic's playback objects include **Port**, **Performance**, **Track**, and **Segment**. The **Performance** object is the music playback überobject. It adds and removes **Port**s, downloads **Instrument**s, attaches graphs of **Tool**s, deals with event notification, and plays **Segment**s. **Segment** objects contain data in one or more **Track**s, which is where the actual music data resides.

A DirectMusic **Track** is not the same thing as a track within a type 1 MIDI file. In fact, a DirectMusic **Segment** can contain all of the data from an entire imported MIDI file.

About the simplest thing an application can do with DirectMusic is to create a **Performance**, create a **Loader**, and tell it to load a single MIDI file. The **Loader** returns the MIDI file in the form of a **Segment**. To play the **Segment**, call the **Performance**'s **PlaySegment** method.

FIGURE 3. *An imported MIDI file, which the* **segment** *object splits into three tracks containing notes, tempo, and time signature information*

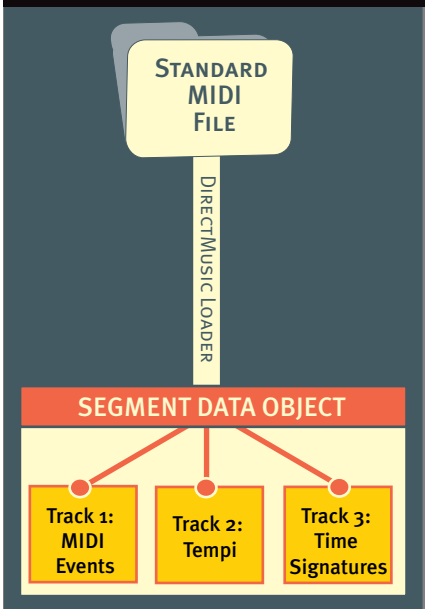
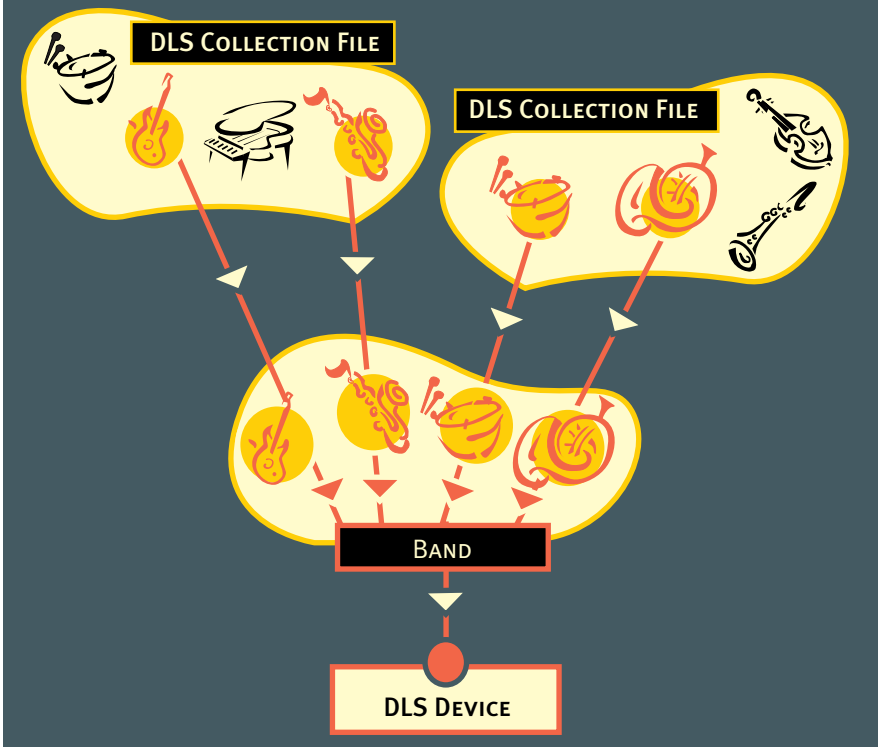


FIGURE 4. *Bare-bones MIDI playback with custom instruments: load a Standard MIDI File, load a DLS Collection or Band, and play.*



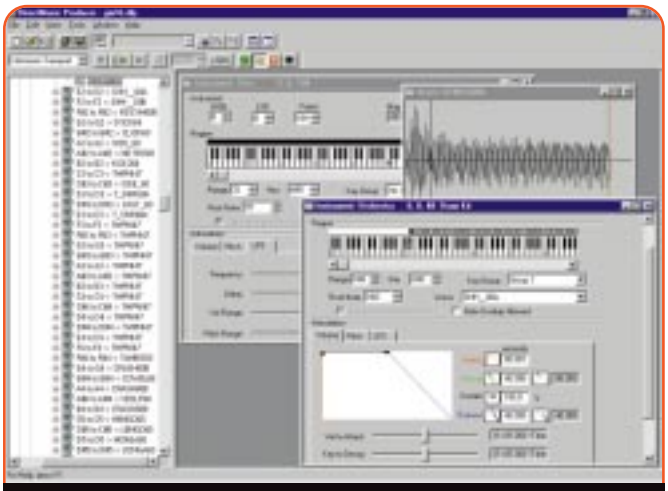


FIGURE 5. *DirectMusic Producer's Downloadable Sounds (DLS) editing windows.*

While it's possible simply to play a MIDI file without invoking any more of the Performance API than I just described, an application can easily bring to bear more of DirectMusic's control mechanisms. For example, you could create a Performance containing Segments, each comprising a single, complete MIDI file. Using the Performance API, an application can then queue, layer, and modify these Segments. Each Segment can get a delay value when queued for playback; this value can then adjust itself to match tempo changes. The `PlaySegment` method also accepts a parameter telling it what type of rhythmic juncture — beat, bar, note, and so on — to jump in on. By playing multiple Segments simultaneously, an application can add and subtract musical elements.

The next logical item on many game music composers' current wish lists is a way to manage segment playback based on game state inputs using an authoring-level scripting scheme or something similar. Using a set of variables shared between the music engine and the host application, this type of system could emulate what a music editor does for a film: watch what's going on and select, mix, and match existing musical elements accordingly.

High-level scripting isn't part of DirectMusic. Some developers think this should have been the fundamental thrust of any music system from Microsoft, and that DirectMusic misses the point. Its authoring-level logic doesn't go beyond the single-segment level; to do more requires application code.

On the other hand, I can't see anything in DirectMusic's architecture that would preclude a higher-level system for real-time rendered music editing. Its low-level code should make this sort of thing easier and more reliable than it was under the old MidiOut system.

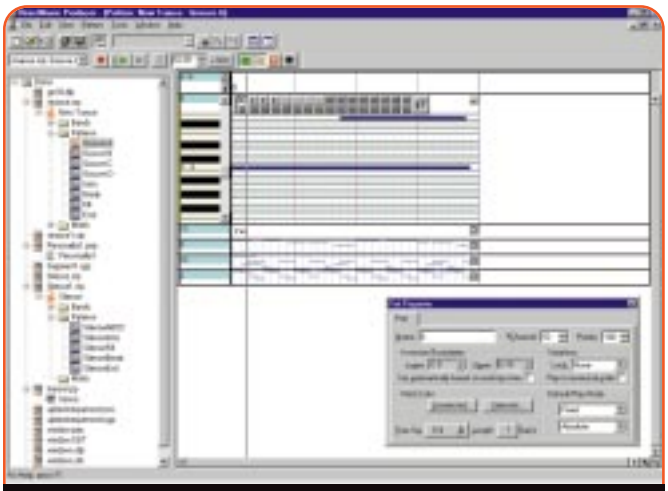


FIGURE 6. *A Pattern screen, with one Part expanded for editing, the Part's Properties dialog, and the notes of one of its Variations on a piano-roll timeline.*

Layers

DirectMusic's sharp timing comes courtesy of its core layer. This layer also supports the software synthesizer and other DLS-related services. It supports buffered, time-stamped MIDI input and output, letting the system do things such as play multiple sequences with completely independent timing. Normally, DirectMusic itself sequences the MIDI data, but others can write their own sequencers and plug them into DirectMusic. All of the higher-level stuff — loading and playing files and the interactive music engine — is part of the performance layer.

Composing with DirectMusic Producer

DirectMusic has two audiences that it must please: audio creative types and programmers. The face of DirectMusic for a musician or sound designer is an application called DirectMusic Producer. A tutorial, or even a decently comprehensive review, should be the subject of another full feature article once the tool is complete.

Producer's nature reveals itself with the "Insert File into Project" dialog, when it shows a list of the sorts of things it can open. These include all of DirectMusic's editable data types: Bands, DLS Collections, Chord Maps, Templates, Segments, and Styles. Each of these existing data types has its own interface within Producer (Figure 5). These almost behave as their own applications, except that they can pass data back and forth and can be built into unified projects.

One great thing about Producer is that it comes with an API that lets developers build new editing tools into the application. This means that if, for example, a vendor wanted to make its algorithmic music generator available as a DirectMusic component, it could build the editor right into the Producer, allowing it to talk with other components such as the DLS editor.

How Deep Do You Want to Go Today?

DirectMusic's Interactive Music engine can be used to varying degrees. The deepest levels are only going to be of interest to a few developers, as they get into rather specialized solutions. The simplest level should be of interest to plenty of developers: just import one or more MIDI files, each as a segment, and thus make them available to the API for queuing and scheduling.

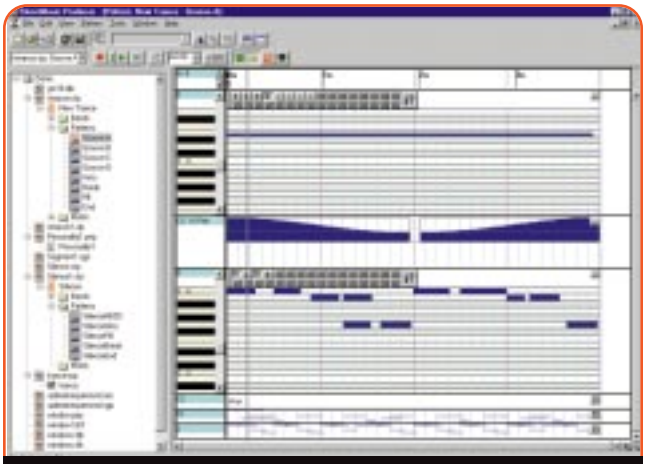


FIGURE 7. *A Pattern screen, with parts showing controller data and multiple variations. The button marked with a chord and a question mark opens a dialog for each variation, letting a composer choose its harmonic situations.*

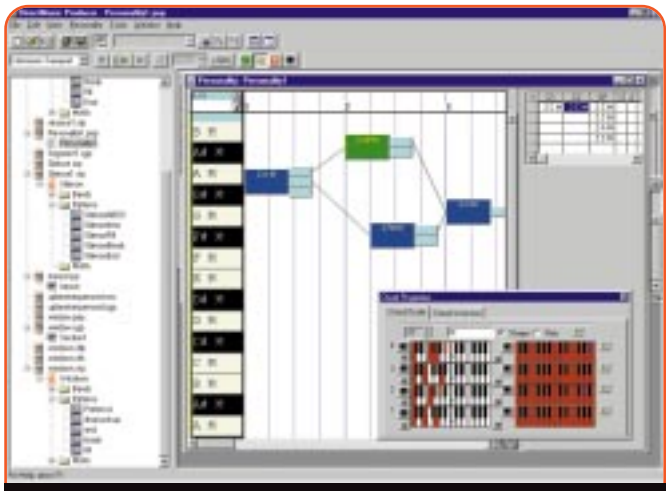


FIGURE 8. *Basic style playback using a prebuilt segment file. The Style Playback Track can point to different Styles as playback progresses.*

The next level gets into a data type called a Style. Styles contain patterns (Figure 6), which are like MIDI sequence files in that they contain one or more parts, each with a single instrument, that can be set to play with no randomness or variation. So, conceptually, the simplest style is just like a MIDI file.

Going one step deeper, you can add variations to individual parts (Figure 7). Typically, these are made by copying the contents of a part and adding or subtracting notes to change its feel and density. You can tie together variations in different parts within a style, so that they encompass more than one instrument. These variations can then be chosen either by the game's code or by parameters set within the music engine.

A more basic parameter for selecting patterns is a number ranging from 1 to 100 called Groove Level. Groove Level can derive from a Groove Track, but it can also be set by your game based upon state variables. The more intense the state of things, the higher the Groove Level. This lets DirectMusic choose patterns based upon the groove ranges assigned to them by the composer.

More than one pattern can play at once. Unless a secondary pattern has tempo data associated with it, it will take its timing from the main pattern. A specialized type of pattern, called a motif, is intended to be triggered by events. Motifs generally consist of only one or two instruments and are short. The simplest example might be a single drum hit.

Up to this level, no actual notes are being generated or even bent by the music engine. It's simply been storing, playing, and combining musical elements that were fully composed by a human being. The most extensive use of this engine in a game to date, Monolith's SHOGO - MOBILE ARMOR DIVISION, went no deeper than this.

The next step, if you take it, starts automatically transposing some notes. This involves a segment track type called a chord progression (Figure 8). To use one, abstract the chord changes from your piece of music and use DirectMusic Producer to place them in a chord progression track within the segment. On playback, the style engine recreates the proper notes by mapping the notes in the style to the harmonic information within the chords. Each chord supports up to four subchords, called levels.

Using chord progressions requires building in a bit more information into your parts and variations. Various attributes can be set at levels ranging down to the individual note to determine whether or not a musical element can be transposed by a chord, and if so how (Figure 9). Variations can be set to play only at certain scale positions or junctures. Parts can be assigned to different levels within the chords.

Beyond this, DirectMusic includes templates and chord maps (formerly called personalities), which the composition engine can use to automatically generate segments (Figure 10). A template is a segment that has everything for style playback except the chord progression track. Instead, the template includes a sign post track, which defines a road map for how to place chords in the chord progression. A separate file known as a chord map defines the actual chords as well as rules for mapping them to the sign post track. These include sign post chord definitions and a tree graph of chord connections. The composer who creates the template can build in weights for the probability of choosing one chord over another at a given juncture in the music.

The composition engine combines the chord map and template to create a style playback segment with the resulting chord progression, along with the template's groove track and style playback track. By combining different chord maps and styles with a single template, an application freshly com-

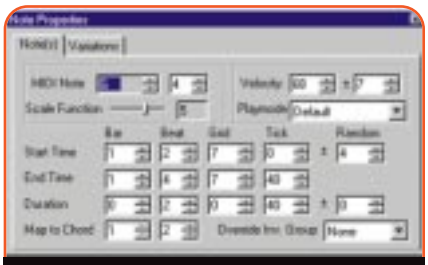


FIGURE 9. *Properties available for each individual note in a style. The Variations tab opens a set of checkboxes attaching the note any of the variations in the pattern.*

poses musical variations for each scene.

And then there are shapes, which actually generate templates: "Give me forty bars of music that rise, and then a snappy 12-bar coda." It's a bit more involved than that, but you get the idea.

By the time you're using shapes, the chord progression is truly generative, but the original composition work that went into the style still peeks through. For more information on templates and shapes, see the documentation on Microsoft's DirectX web site.

Templates and shapes can create style playback segments offline, for example during a game level load. This won't interrupt whatever playback DirectMusic is up to at the time.

The Future

A common thought about DirectMusic is that it should merge with DirectSound, combining the APIs' strengths and addressing their weaknesses. According to Kevin Bachus, DirectMusic's product manager, "DirectSound and DirectMusic are really siblings in the same audio organization. Seamless integration of DirectSound and DirectMusic is very important. Expect to see more and more convergence in future versions of DirectX."

DirectMusic's DLS output can benefit from DirectSound-specific features such as spatial positioning (a.k.a. 3D sound). In turn, DLS offers control over sound effects from within authoring tools, letting sound designers take some of the sound effects implementation out of the hands of programmers. It can also offer features of which DirectSound has no inkling — such as envelopes and midsample looping — which can be completely set up by a sound designer and triggered using standard MIDI commands.

As I mentioned, this is a big package. It attempts a great many things, and based on the alpha version, it seems to do many of them well. It doesn't include some things that many developers wanted, especially a track-based, scriptable system for controlling adaptive music playback. However, what it offers is closer to solving some of the same problems than might seem evident at first glance.

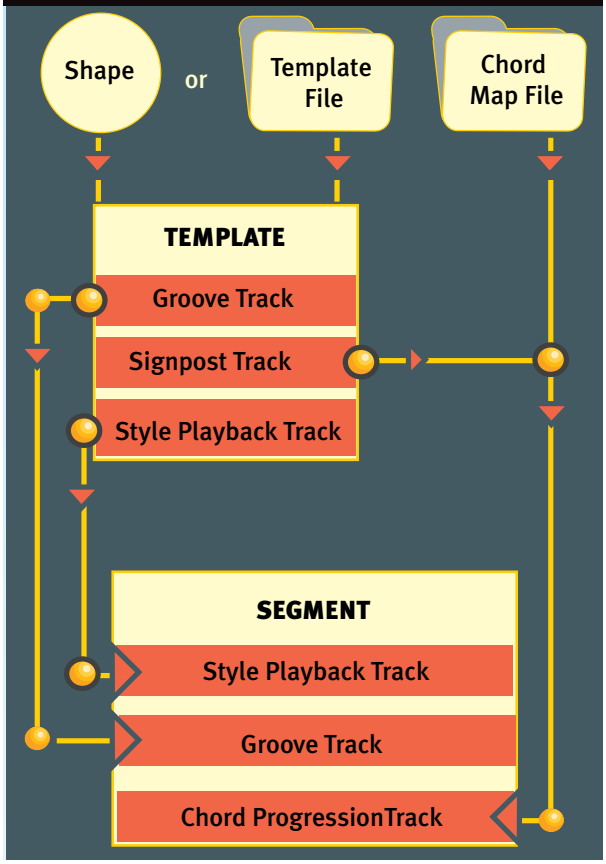
So, what is DirectMusic good for? Who is it good for? For those game developers who continue to use MIDI as their games' music output, the basic architectural improvements are long overdue. For those who can realize their music well using custom DLS1 sound sets and can afford some CPU hit on unaccelerated user machines, DLS will be a relief from being stuck with inconsistent playback and General MIDI's limited palette of sounds. If you have plenty of disc space, don't need run-time access to the CD drive, want dirt-simple programming, and don't care about adaptivity in your music, Red Book remains the way to go, even though DLS can do full CD quality (44KHz 16-bit stereo).

If you want to do adaptive music, first analyze what you want to happen. Perhaps videotape some game play and score music to this using traditional techniques. Once you've figured out how music should ideally operate in your game, look at both the controls your code needs to spit out and what the music engine needs to do in response. Once you've defined the task to this degree, you may or may not find your solution within DirectMusic. Take a look at third-party APIs such as Miles or DiamondWare; you're likely to find the codebase you need without writing it all yourself.

If you want an adaptive digital audio streaming engine, try doing this with large samples under DLS. With the software Microsoft has supplied as of this writing, I can't judge whether this will work well or not.

Will DirectMusic make MIDI relevant again for you? What am I, your mother? Set aside a day and check it out. If nothing else, it's brain candy. Enjoy. ■

FIGURE 10. *The Composer engine can create unique Style Playback Segments by combining Templates with Chord Maps. Templates can be pre-built, or generated using Shapes.*



FOR FURTHER INFO

Microsoft DirectX Documentation
http://www.microsoft.com/directx
To get information and offer feedback about DirectMusic, e-mail dxmusic@microsoft.com

General Reference
Kientzle, Tim. *A Programmer's Guide to Sound*. Reading, Mass.: Addison-Wesley Developers Press, 1998.

Interactive Audio Special Interest Group (IA-SIG)
http://www.iasig.org

MIDI Manufacturers' Association
http://www.midi.org

Acknowledgements

The author would like to thank Microsoft's Todor Fay and Dan Teven of Teven Consulting for their assistance with this article.

M l i e i g i Di ec X 6

by Jason L. Mitchell, Michael Tatro and Ian Bullard

One of the most interesting features introduced to Direct3D in the recent release of DirectX 6 is multiple texturing. Unfortunately, it's also one of the more confusing new features. This article will introduce multiple texture mapping into the context of the traditional pixel pipeline. We will describe the multitexture programming model, provide programming examples, and spend some time addressing the issues involved in robustly taking advantage of multitexturing hardware while maintaining fallback paths for application-level multipass methods. We have also created MulTex, a simulator that interactively illustrates this potentially puzzling new feature of Direct3D. Experimenting with MulTex is a good way to gain some familiarity with the texture blending abstraction. MulTex is available from the *Game Developer* web site and is definitely useful to have by your side as you read this article. (MFCTEX, a similar tool written by Microsoft, ships with the Microsoft DirectX 6 SDK.)

The Traditional Pixel Pipeline

In previous versions of DirectX, the texture mapping phase of the Direct3D pixel pipeline has only involved fetching texels from a single texture. The two gray pipeline segments in Figure 1 are the stages in the traditional pipeline that deal with determining texel color and blending that color with the color of the primitive interpolated from the vertices. These two stages of the pipeline are replaced by the new multitexturing abstraction. The rest of the pipeline remains untouched.

Texture Operation Units

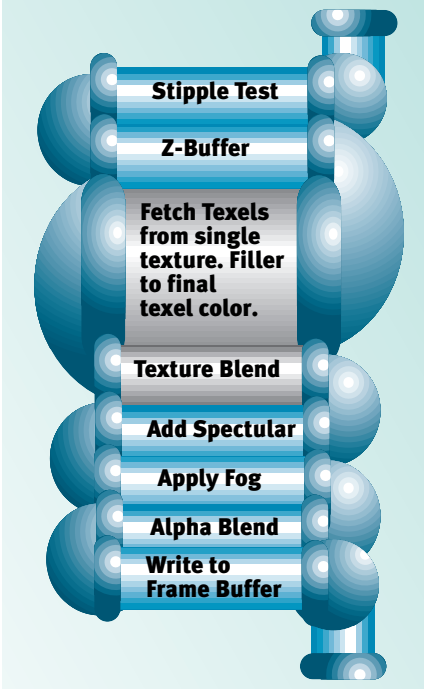
DirectX 6 introduces the concept of a texture operation unit. Each unit may have a single texture associated with it, and up to eight texture operation units can be cascaded together to apply multiple textures to a common primitive.

Each texture operation unit has six associated render states, which control the flow of pixels through the unit, as well as additional render states associated with filtering, clamping, and so on. Figure 2 shows two texture operation units cascaded together. We'll limit our discussion here to the dual texture case to keep things simple and because most of the near-term 3D hardware will support only two textures.

Three of the render states in each texture operation unit are associated with RGB (color), and another three are associated with alpha. For RGB color, the render states **D3DTSS_COLORARG1** and **D3DTSS_COLORARG2** control arguments, while **D3DTSS_COLOROP** controls the operation on the arguments. Likewise, **D3DTSS_ALPHAARG1** and **D3DTSS_ALPHAARG2** control arguments to **D3DTSS_ALPHAOP**. Essentially, the **D3DTSS_COLOR***x* render states control the flow of an RGB vector, while the **D3DTSS_ALPHA***x* render states govern the flow of the scalar alpha through parallel segments of the pixel pipeline, as shown in Figure 2.

Jason L. Mitchell (JasonM@atitech.com) is a software engineer in the 3D Application Research Group at ATI Research Inc. Michael J. Tatro (mike@stainlesssteelstudios.com) is a software engineer at Stainless Steel Studios in Cambridge, Mass. Ian Bullard (ianb@technologist.com) is a software engineer at New World Computing.

FIGURE 1. *The Direct3D pixel pipeline.*



Arguments

Using the argument states, you can direct input, such as interpolated diffuse color or texel color, into the texturing operations. Table 1 shows a complete list.

Additionally, you can invert the arguments or replicate their alpha channel across the RGB channels. In the API, you can bitwise OR in the constants D3DTA_COMPLEMENT and D3DTA_ALPHAREPLICATE with any of these render states to achieve the desired effect. D3DTA_COMPLEMENT simply inverts each of the color channels, while D3DTA_ALPHAREPLICATE replicates the alpha from the argument across the R, G, and B channels. Naturally, the D3DTA_ALPHAREPLICATE flag isn't meaningful if it's used with D3DTSS_ALPHAARGx. Also, D3DTA_CURRENT doesn't make sense for the 0 texture operation unit because there is no previous texture operation unit.

Operators

The operators in each unit can operate on one or both of the corresponding arguments. The operator render states can be set to any of the values in Table 2.

This long list of operations may seem a bit daunting at first, but with some experimentation, the abstraction is actually quite approachable. To get you started with the model, the next section illustrates some common multitexture techniques and how they can be programmed in DirectX 6. We suggest that you follow along with MulTex.

FIGURE 2. *This screenshot, taken from the MulTex utility, shows two cascaded texture operation units.*

Each texture operation unit also has states for texture addressing and filtering associated with it. The application programmer can set these render states independently for each texture operation unit. A common example would be to set the base texture of an object, such as the brick texture in the following dark mapping example (Figure 3), to use D3DTADDRESS_WRAP texture addressing, while the texture operation unit for the dark map uses D3DTADDRESS_CLAMP.

Multitexture Examples

DARK MAPPING. Naturally, our first example of multiple texture mapping is the dark map described by Brian Hook in the August 1997 issue of *Game Developer* ("Multipass Rendering and the Magic of Alpha Rendering"). Dark mapping is commonly used in lieu of vertex lighting, where one of the two textures contains an unlit base texture and the other contains a lighting texture (the dark map). Using the new multiple texturing API, one might implement this technique as shown in Figure 2. In the figure, the two large blue boxes represent texture operation units, and the red lines show the flow of data through the pipeline. The first texture operation unit merely passes data from texture 0 to the next stage. The second texture operation unit receives these texels via Arg2 and also fetches texels from texture 1 via Arg1. The results are modulated, giving the final texel color as shown on the right-hand side of Figure 3. Nothing interesting is being done with the alpha channel of the pipeline in this case. Code (generated by MulTex) for dark mapping is shown in Listing 1.

MODULATE2X. The preceding technique is called dark mapping rather than light mapping because the resulting texel can only be a darker version of the unlit texel from the primary map. For this reason, some applications use a variant of the modulate technique, where the resulting texel is brightened by a factor of two using the D3DTOP_MODULATE2X operation instead of D3DTOP_MODULATE. (This can also be done in two passes using the alpha blending operation of Src*Dest + Dest*Src.) One notable engine that uses this technique is AnyChannel's AnyWorld engine, used in the upcoming Postlinear/SegaSoft game, VIGILANCE (Figure 4). The AnyWorld engine uses a radiosity lighting model that requires precomputed light maps from LightScape, converted for use in the engine. The advantage to using multitexture rendering in this case is that the base texture maps can be tiled and reused, while the light maps, which are unique to each polygon and very low resolution, are used with clamping. This trades the high polygon count associated with radiosity rendering for a fairly large texture footprint.

SPECULAR HIGHLIGHTS AND ENVIRONMENT MAPPING. In order to incorporate view-dependent reflection of light sources into the lighting model, a specular term

TABLE 1. *DirectX 6 texturing operations.*

D3DTA_TFACTOR	Take pixel data from API-level factor. This factor is set with RENDERSTATE_TEXTUREFACTOR.
D3DTA_DIFFUSE	Use interpolated diffuse color (Gouraud shading).
D3DTA_CURRENT	Use color from previous texture operation unit.
D3DTA_TEXTURE	Use color from texture associated with this unit.

TABLE 2. *Operator render states.*

D3DTOP_DISABLE	Disable this and any later texture operation units
D3DTOP_SELECTARG1	Pass argument 1 untouched
D3DTOP_SELECTARG2	Pass argument 2 untouched
D3DTOP_MODULATE	Multiply both arguments together
D3DTOP_MODULATE2X	Multiply both arguments and shift 1 bit
D3DTOP_MODULATE4X	Multiply both arguments and shift 2 bits
D3DTOP_ADD	Add Arguments together
D3DTOP_ADDSIGNED	Add Arguments with -0.5 bias
D3DTOP_ADDSIGNED2X	Add Arguments with -0.5 bias and shift 1 bit
D3DTOP_SUBTRACT	Subtract Arg2 from Arg1, with no saturation
D3DTOP_ADDSMOOTH	Add arguments and subtract product
D3DTOP_BLENDDIFFUSEALPHA	Blend arguments based on interpolated alpha
D3DTOP_BLENDTEXTUREALPHA	Blend arguments based on texture alpha
D3DTOP_BLENDFACTORALPHA	Blend arguments based on factor alpha
D3DTOP_BLENDTEXTUREALPHAPM	Linear alpha blend with premultiplied Arg1 Arg1 + Arg2*(1-Alpha)
D3DTOP_BLENDCURRENTALPHA	Blend arguments based on current alpha
D3DTOP_PREMODULATE	Modulate with next texture before use
D3DTOP_MODULATEALPHA_ADDCOLOR	Arg1.RGB + Arg1.A*Arg2.RGB
D3DTOP_MODULATECOLOR_ADDALPHA	Arg1.RGB*Arg2.RGB + Arg1.A
D3DTOP_MODULATEINVALPHA_ADDCOLOR	(1-Arg1.A)*Arg2.RGB + Arg1.RGB
D3DTOP_MODULATEINVCOLOR_ADDALPHA	(1-Arg1.RGB)*Arg2.RGB + Arg1.A
D3DTOP_BUMPENVMAP	Per pixel environment map perturbation
D3DTOP_BUMPENVMAPLUMINANCE	Environment map perturbation w/luminance channel
D3DTOP_DOTPRODUCT3	A per-pixel dot product that could be used for specification of surface normal vector data in texture maps. The result is (Arg1.R*Arg2.R + Arg1.G*Arg2.G + Arg1.B*Arg2.B) where each component is scaled and offset to make it signed.

is added to the view-independent (diffuse) term. In Direct3D, an application can provide the renderer with specular values at polygon vertices by passing them in the vertex structures. The interpolated specular colors are then added to the lighted texture color as shown in Figure 1. The problem with this method is that specular reflections tend to be fairly localized on an object, and their appearance can vary wildly depending on the tessellation of the object in the area of the specular reflection. Theoretically, the peak brightness of a traditional specular highlight can fall within a polygon (as in, not at a vertex), but this interpolation scheme doesn't reproduce such behavior. Figure 5A shows the artifacts caused by using vertex specular lighting. These artifacts are even more pronounced when the object and/or viewer are in motion.

One solution to this problem is to use a secondary texture as a specular light map, as shown in Figure 5B. The secondary texture coordinates of these polygons are generated by projecting into the light map (perhaps using a hemispherical environment mapping technique). The same effect, with a full environment map, is shown in Figure 5C. In Figures 5B and 5C, the teapot is rendered using traditional diffuse lighting at the vertices, which is modulated with the primary (woodgrain) texture. To this product, the second blending unit adds the specular map. Direct3D multiple texturing syntax is shown in Listing 2.

LINEAR BLENDING. Multitexturing also lets you linearly blend between two textures for morphing effects. You can use any of the D3DTOP_BLEND*x*ALPHA operations, but D3DTOP_BLENDFACTORALPHA or D3DTOP_BLENDDIFFUSEALPHA are most efficient for frame-to-frame variation of the linear blend factor. A good example of using a linear blend for morphing is the ATI Knight Demo, first shown at the CGDC in 1997 and illustrated in Figure 6. In this example, the stone texture map on the knight statue is the primary texture, and the "living" texture map is secondary. From frame to frame during the morph, varying the blend factor causes the living texture to fade in until it is the only texture visible. For efficiency, before and after the morph, traditional single texture mapping is used with the appropriate texture (Figures 6A and 6D).

WHAT ABOUT PLAIN OLD DIFFUSE VERTEX LIGHTING? As show in Figure 1, DirectX 6 has rolled the whole texture blending phase of the traditional pixel pipeline into the multiple texture mapping model. As a result, there is no separate texture blending render state like the one that existed in previous versions of Direct3D. Instead, the program needs to use a single texture operation unit to perform common vertex lighting. In order to use common vertex lighting on a single texture, program texture blending unit 0 as shown in Listing 2, but disable texture blending unit 1.

OTHER TEXTURE OPERATIONS. What we've illustrated here are techniques that are likely to be of immediate use to developers, given the effects popular today and the capabilities of available hardware. Microsoft has provided illustrations of a variety of multitexturing effects and their multipass equivalents in the DirectX 6 SDK, though some of the single-pass versions of the techniques may not be supported by current or even next-generation hardware. Of course, users of Direct3D applications are likely to have cards with varying coverage of multitexture features (including no multitexturing features at all). How can an application determine the feature coverage of the hardware it is running on and use the most optimal multitexturing technique? Fortunately,

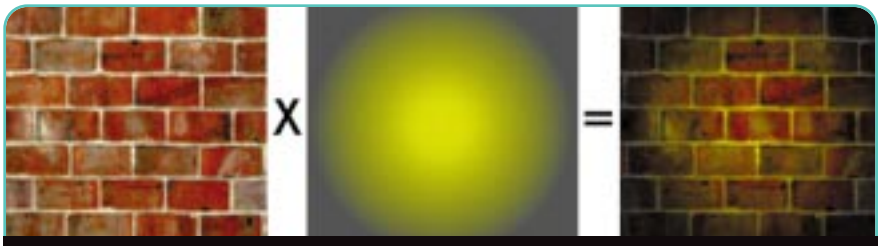


FIGURE 3. *Dark mapping in action.*

FIGURE 4. *The AnyWorld engine. The walls and floor are tiled with smaller textures while large light maps add highly detailed lighting to the scene.*

the new API includes a means for validating the multitexturing techniques an application will use. In the next section, we cover this validation scheme as well as an architecture for incorporating fallback techniques so that an application will be able to take advantage of multitexturing hardware when it's available and robustly fall back to multipass techniques when it isn't.

The Reality of Hardware Support

You'll note that throughout this article, we've consistently referred to the new texture abstraction as just that, an abstraction. The model, or abstraction, that's illustrated here and documented in Microsoft's DirectX 6 SDK doesn't necessarily map directly to the silicon designed by 3D video card makers. In fact, in some cases, the silicon predates the API. Additionally, the full feature set defined in the model isn't implemented on any 3D cards available to date. As a result, programmers may initially find support of multiple texture mapping modes somewhat sparse relative to the extreme flexibility of the API. Microsoft has written a reference rasterizer for DirectX 6 that implements the full multiple texture mapping model; software developers can use these to experiment with new effects. Also under DirectX 6, the software rasterizer (not the same as the reference rasterizer) has support for two texture operation units and a reasonable subset of the operations defined by the API. MulTex can also serve as a tool for experimenting with new techniques. For the foreseeable future, we recommend that developers plan to implement both multipass and single-pass versions of techniques in their titles, where the multipass code is executed when the application is running on boards that cannot provide the desired functionality in a single pass.

Typically, we expect developers to define a set of materials that they will use in their applications. These materials are defined by the texture operation units' arguments and operations that will be used when rendering a polygon of that material type, as well as a few other criteria, which we'll touch on in a moment. Each multitextured material will have multiple ways that it can be rendered: the ideal single-pass case, the multipass fallback case, and any intermediate cases. For example, a wall with a static diffuse light map and a dynamic light map might have three rendering methods (Table 3).

At application initialization time, the 3D application should run through its

LISTING 1. *Dark mapping. In this code snippet, unspecified render states are left in their default states for brevity. In practice, an application should be more defensive than this.*

```
// Program Stage 0:
lpDev->SetTexture(0, pTex0 );
lpDev->SetTextureStageState(0, D3DTSS_COLORARG1, D3DTA_TEXTURE);
lpDev->SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_SELECTARG1);

// Program Stage 1:
lpDev->SetTexture(1, pTex1 );
lpDev->SetTextureStageState(1, D3DTSS_COLORARG1, D3DTA_TEXTURE);
lpDev->SetTextureStageState(1, D3DTSS_COLORARG2, D3DTA_CURRENT);
lpDev->SetTextureStageState(1, D3DTSS_COLOROP, D3DTOP_MODULATE);
```

LISTING 2. *Specular mapping as shown in Figures 5B and 5C.*

```
// Program Stage 0:
lpDev->SetTexture(0, pTex0 );
lpDev->SetTextureStageState(0, D3DTSS_COLORARG1, D3DTA_TEXTURE);
lpDev->SetTextureStageState(0, D3DTSS_COLORARG2, D3DTA_DIFFUSE);
lpDev->SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_MODULATE);

// Program Stage 1:
lpDev->SetTexture(1, pTex1 );
lpDev->SetTextureStageState(1, D3DTSS_COLORARG1, D3DTA_TEXTURE);
lpDev->SetTextureStageState(1, D3DTSS_COLORARG2, D3DTA_CURRENT);
lpDev->SetTextureStageState(1, D3DTSS_COLOROP, D3DTOP_ADD);
```

FIGURE 5. *Vertex specular (A) versus specular mapping (B) and environment mapping (C).*

TABLE 3. *A three-tiered approach to multitexture rendering a wall with a static light map and a dynamic light map.*

Top Tier	Single-pass	(Static Light Map + Dynamic Light Map) * Wall Texture
Middle Tier	Two-pass	Frame buffer = Static Light Map + Dynamic Light Map Alpha blend to get Wall Texture * Frame buffer
Bottom Tier	Three-pass	Frame buffer = Static Light Map Alpha blend to get Frame buffer = Static Light Map + Dynamic Light Map Alpha blend to get Frame buffer = (Static Light Map + Dynamic Light Map) * Wall Texture

materials to determine which rendering method it can use for each material on the given hardware. The application would program the texture operation units for the top tier and validate this set of states by calling the new function `IDirect3DDevice3::ValidateDevice()`. If the function passes, the application can use this tier when this material is being rendered. If `ValidateDevice()` fails, the application should keep moving downward until a tier passes validation. When `ValidateDevice()` fails, it returns an error code that indicates why it failed; applications should be prepared to handle this successfully. The error codes are shown in Table 4.

These return codes are the additional parameters that make up what we are considering a material. For example, an application that plans to use dark mapping techniques, sometimes using trilinear filtering and sometimes using bilinear filtering, should consider each of these cases separately and validate accordingly. This distinction will probably become less critical in a year or two, but with the first crop of multitexture hardware, it's very important to pay attention to validation.

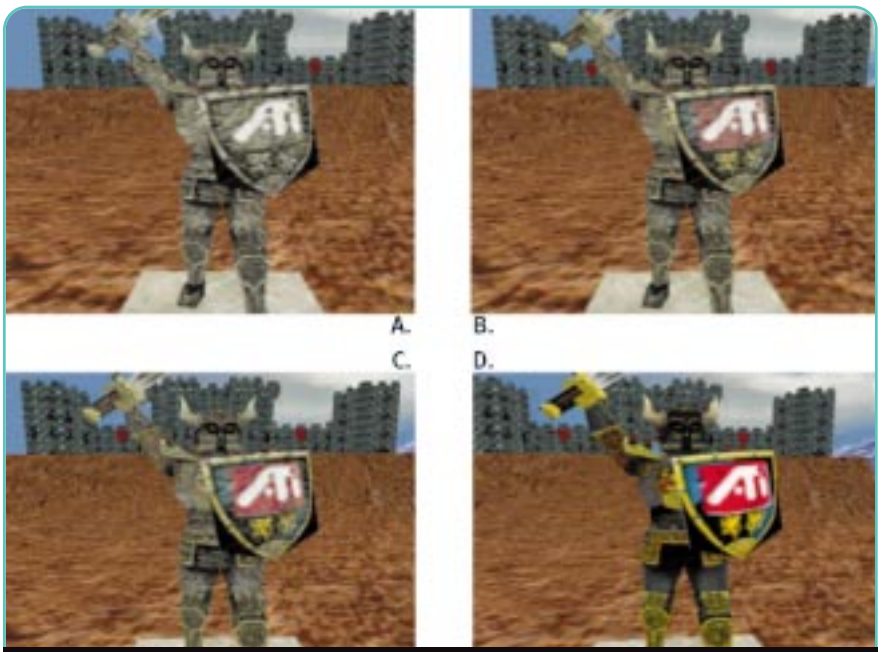


FIGURE 6. *Four frames of the ATI Knight Demo from CGDC 97 showing a texture morph using a linear blend factor. The camera is moving slightly during this series of shots. The full .AVI is available from the* Game Developer *web site.*

TABLE 4. `IDirect3DDevice3::ValidateDevice()` *return codes.*

WRONGTEXTUREFORMAT	The hardware cannot support the current state in the selected texture format.
UNSUPPORTEDCOLOROPERATION	The specified color operation is unsupported.
UNSUPPORTEDCOLORARG	The specified color argument is unsupported.
UNSUPPORTEDALPHAOPERATION	The specified alpha operation is unsupported.
UNSUPPORTEDALPHAARG	The specified alpha argument is unsupported.
TOOMANYOPERATIONS	The hardware can't handle the specified number of operations.
CONFLICTINGTEXTUREFILTER	The hardware can't do both trilinear filtering and multitexture at the same time.
UNSUPPORTEDFACTORVALUE	The hardware can't support `D3DTA_TFACTOR` greater than 1.0.

Chipsets with Multiple Texture Support

At press time, the ATI Rage Pro and the 3Dfx Voodoo2 are the only shipping cards with multitexture support. So developers already have access to the first crop of available hardware. For up-to-date information, DirectX 6 drivers, and a list of which texture operations are supported, keep an eye on the developer support material on the ATI and 3Dfx web sites. ■

Acknowledgements

Thanks to AnyChannel, John Pallett-Plowright, and our colleagues at ATI for their input.

FOR FURTHER INFO

3Dfx Inc.
http://www.3dfx.com

ATI Research Inc.
http://www.ati.com

AnyChannel's AnyWorld engine
http://www.anychannel.com/anyworld.html

Pla i g AM i Ca e Sh

by Melianthe Kines

DAY 1 OF YOUR MOTION CAPTURE SHOOT, 3:30PM. So far, you've only captured four moves on your list. The talent (who arrived late because he didn't have directions to the studio) is complaining that his sensored-up shoes are killing him (the shoes are a size too small). The game's lead animator looks glum and keeps asking you to reshoot moves. He can't explain why, but what the talent is doing is "all wrong." The performer is perplexed and asks the animator for feedback instead of talking to you, the director. Your lead programmer, on the other hand, is happy with the talent's performance, glimpsing the action as she shuffles through your shot list for the first time. In fact, she's already asked for ten additional variations, plus transitions for each move. The studio manager informs you that you have an hour left to shoot because the crew needs time to recalibrate the studio before they're scheduled to go home at six o'clock. Just then, your executive producer pops in to let you know that the product manager is bringing in a photographer from a game magazine. "Everything going all right?" he asks you. "Good, because we've only got a few months to get that animation into the game." It slowly dawns on you that maybe some planning would have been a good idea.

Motion capture is an incredible technology that can be used to create breathtaking game animation. Still, it's a tool, not a magical solution. Learn how to use motion capture properly, and it can produce great results that will make your life easier. Show up to your shoot without preparation, and you'll almost certainly waste time and money; worse, you might not get any useable animation. If you're a typical game developer, you're watching your budget closely and guarding your schedule even more fiercely. In the long run, planning your shoot and visualizing the end result is worth the time spent up front.

Are you considering motion capture for your next game? First, determine if it's really right for your project. Motion capture is most useful for 3D games with tons of character animation. Consider the game engine, the style of anima-

Melianthe Kines is a freelance interactive director and producer. She has directed motion capture and Ultimatte shoots for Acclaim Entertainment and Electronic Arts. Her past motion capture projects include NBA JAM EXTREME, NBA JAM '99, THE CROW: CITY OF ANGELS, and FIFA: ROAD TO WORLD CUP '98; her Ultimatte production credits include WWF: IN YOUR HOUSE and FRANK THOMAS BIG HURT BASEBALL. She can be contacted via e-mail at mkines@escape.com.

tion you want, and of course, your budget and schedule. You may want to combine motion capture data with Ultimatte video elements and other types of animation. Motion capture is by no means appropriate for every team and every game; on the other hand, it's hardly "Satan's rotoscope" or a threat to animators in any way. In fact, animators are critical to planning the shoot and then turning the data into something useful and attractive.

In this article, I'll explain how to plan the shoot; next month, I'll discuss how to direct the talent and run the motion capture session. These are general guidelines for planning any use of motion capture for games, whatever the specific technical profile of the project may be. Your shoot will have its own set of technical considerations, including the game engine, programming and animation tools, and the motion capture system to be used. I'll be referring primarily to optical motion capture, but the same ideas should apply to other methods of motion capture production (magnetic or tethered systems, for example).

If your team decides to use motion capture, someone will have to take on the responsibility for organizing and producing that aspect of the project. I strongly recommend that your team designate one person as "the director" — in other words, the person in charge of the motion capture production. The director may be the team's producer, programmer, or animator, or a freelance director with motion capture experience. This individual bears the responsibility for making the shoot a success and will have an overview of all project issues related to motion capture. The director will be coordinating information from the entire team, from the game designer to the marketing manager. Whoever is in charge must have a thorough understanding of the game and enough time in his or her schedule to plan the shoot properly. Ideally, this person should be able to communicate clearly and diplomatically with the talent and all members of the team.

Planning a motion capture shoot for a game is very different from planning a shoot for a film or any other linear end product. What's the difference? Your goal is to end up with hundreds of individual moves that connect perfectly to one another. If you've planned other kinds of game animation, you should already have a good understanding of this process. Of course, in a motion capture project, the actual creation of the data involves many additional considerations. Full-motion video sequences, since they are linear, should be handled separately from the in-game character moves.

Starting Out: Reviewing the Animation List and Flowchart

In order to begin planning your shoot, you need the game specification, including an animation list and flowchart. You'll be working with the rest of the team to revise the animation list and flowchart, eventually coming up with a shot list. Let's look at the components in this revision process.

ANIMATION LIST. Take a look at the specification. Who are the characters in the game? These characters should be defined in the animation list, along with the moves that have been planned so far. Has every character been accounted for? In what environments will these characters be operating? With what sorts of objects will they interact? Sports equipment and weapons are obvious props that you'll need, but don't forget about boxes, goal posts, ladders, keys, and so on.

Let's say you're producing an action game called SUPERGUY. I'm going to keep this example fairly simple. Table 1 lists some of the information that might be provided in the SUPERGUY specification.

FLOWCHART. You should have a separate flowchart for each character. There may be one in the specification, but often you'll have to create a flowchart that is more geared toward planning motion capture (Figure 1). In fact, the process of creating a flowchart is a great way to test the depth of the animation list.

Look at your motion capture flowchart and see if the character can easily transition from each move to any other. You'd do this no matter how the animation was being created, but with motion capture, you can't have an animator create the missing move without a reshoot.

ADDING MOVES. The example animation list and flowchart would lead to many questions. For example,

- Does the character have to stand up from a crouch before he can walk or run?
- Does the character have to stop walking or running to fight?
- If Superguy is hit hard enough to fall, is that all one move (**Fall**) or is it be a combination of motions?
- If the character is walking, can he transition to a stop or to

TABLE 1. *Animation list for SUPERGUY example.*

GAME CHARACTERS IN SUPERGUY:
- Superguy (user-controlled)
- Evil Villainess
- Big Bad Boss
- Henchman Type A (small)
- Henchman Type B (medium)
- Henchman Type C (large)

SUPERGUY'S MOVES:
- Stand (rest frame)
- Walk
- Run
- Punch forward/right/left (from stand)
- Kick forward/right/left (from stand)
- Shoot forward/right/left (from stand)
- Crouch
- Punch forward/right/left (from crouch)
- Kick forward/right/left (from crouch)
- Shoot forward/right/left (from crouch)
- Special Move #1
- Special Move #2
- Defend (from stand)
- Get hit forward/right/left (from stand)
- Get hit forward/right/left (from crouch)
- Fall down (from stand)
- Fall down (from crouch)
- Lying down (not dead)
- Get up (from fall to stand)
- Dead

a run from either foot? Or does one walk cycle have to be completed before the transition?

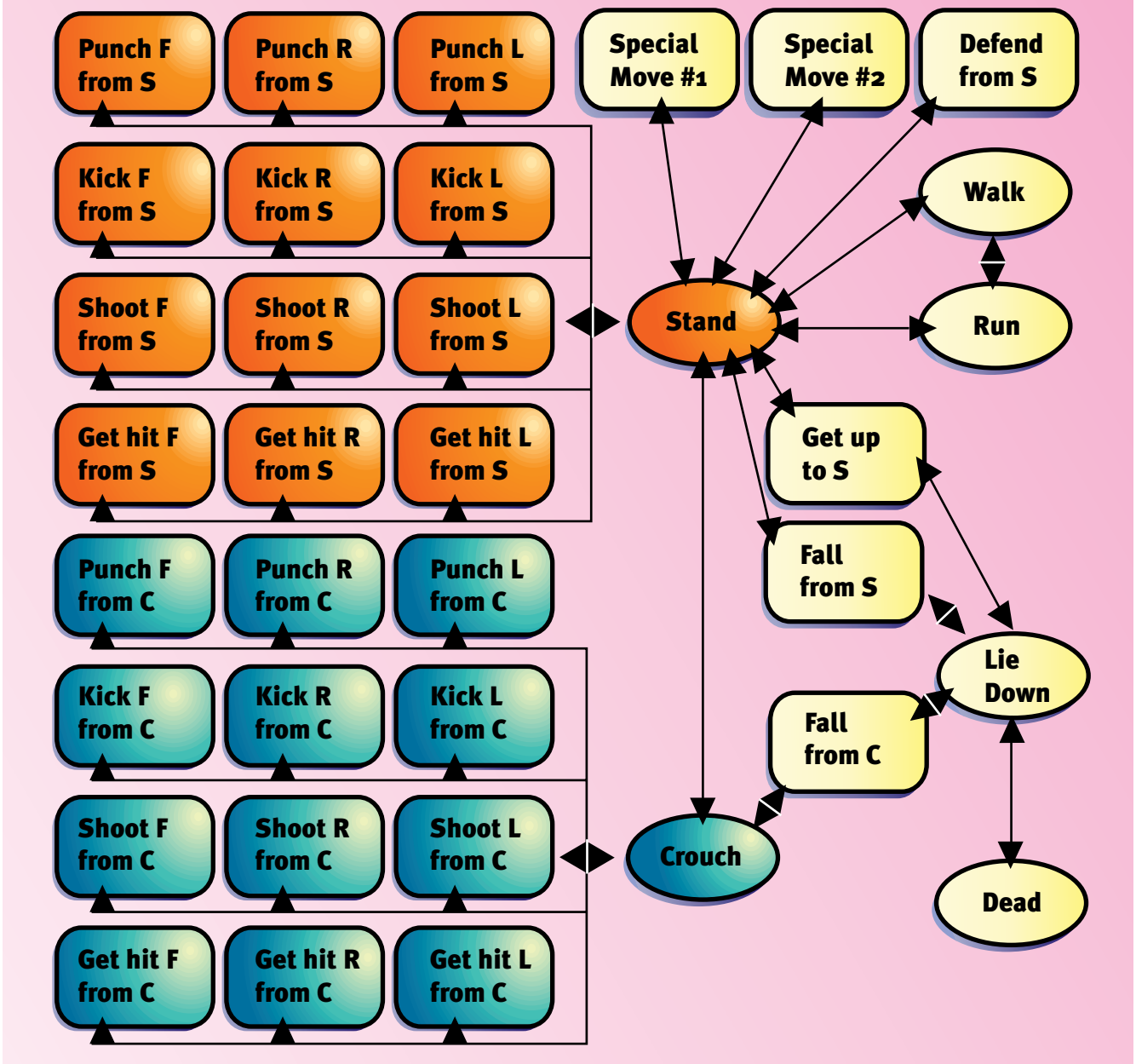
- If the enemy characters are different sizes, where do the punch and kick attacks connect on each of them?
- Do each of the enemy attacks (which would be listed on those characters' move lists) cause the same **Get hit** reaction on Superguy?
- Does Superguy have a gun at all times, or does he pick it up or pull it from a holster?
- What are the parameters for the "special moves?"

The answers to these questions depend on the game's engine and design. Some animation blending tools will handle transitions, so you don't have to capture transition moves. While it's generally safer to capture more data than you need, naturally you don't want to spend extra time and money shooting totally superfluous moves.

Review the list with the designer and producer to see whether additional moves are required to improve game play. Does the player's character have all moves necessary to confront or avoid danger and enemies? Can the character reasonably travel through every area of the game's environment? While this is a design issue and, therefore, the designer and producer's responsibility, it's up to you, the director, to avoid additional motion capture shoots late in the production schedule.

You'll most likely have to add some transitions to any animation list. For this example, we'll say that your team has some great blending tools, but you're going to change the following animations because you need realistic human motion for the transitions:

FIGURE 1. *SUPERGUY's move flowchart (F = forward, R = right, L = left, S = stand, C = crouch).*



Walk to run (starting with left foot)
Walk to run (starting with right foot)
Run to walk (starting with left foot)
Run to walk (starting with right foot)
Crouch to walk
Crouch to run
Get hit from stand (forward/right/left) high
Get hit from stand (forward/right/left) low
Fall down (from stand) forwards
Fall down (from stand) backwards
Fall down (from crouch) forwards
Fall down (from crouch) backwards

Once you've reviewed and amended the animation lists and flowcharts for each character, you can start planning the shot list. You'll also need an estimated frame count for each move. Work closely with the animation team that's going to reduce the motion data to the target size so you understand their procedures. No human being can throw a punch in precisely fifteen frames, but you can determine a relative timing strategy.

Creating a Shot List

So far, an animation list for motion capture seems similar to that for any other type of character animation. Defining the shot list, however, is where you will account for how to shoot the moves in a motion capture studio.

CREATE A DATABASE. It's helpful to use a database program, such as File Maker Pro, to organize the motion capture information. You can generate a shot list from the database, and later you and the other team members will be able to produce customized lists necessary for post-production. You should have separate fields for character names, talent, move names, move descriptions, file names, frame counts, size of capture space, props, and special set-ups (Table 2). Also, note whether the animation is looping or transitional. Any move that repeats is a loop — standing, crouching, or walking, for example. Any move that has a defined start and finish is a transition — a punch, a fall, or a special move. List the starting and ending positions for all transition moves.

FILE NAMES. Establish file naming conventions with your team early in the process. This makes sorting the data easier and will allow you to name additional moves later in the process. Why bother with conventions? Moves are almost always added to the shot list as the team reviews it. You'll likely throw in extra shots when you're in the studio, whether due to a brilliant inspiration or a moment of paranoia. Allowing for last-minute decisions doesn't mean you haven't planned carefully — if you're prepared with a logical naming and reference system in advance.

For example, you could use the following format for the Superguy file names:

SGSA001A: SG (Superguy) S (standing) A (attack) 001 (punch forward) A (version)
EVWT001A: EV (Evil Villainess) W (walking) T (transition) 001 (to run, left foot) A (version)
HBCL001A: HB (Henchman B) C (crouching) L (loop) 001 (crouching) A (version)

PRELIMINARY SHOT LIST. Your preliminary shot list should have as much information as possible. I've created an example of a shot list for Superguy (Table 3). Don't worry about the sequence in which the moves are listed; later on, you'll be arranging them in the order in which you plan to shoot them.

All key team members should thoroughly review and agree upon the completed shot list. Although you'll provide detailed written descriptions of each move, on their own these descriptions can still be subject to interpretation. To clarify, create storyboards that everyone can examine. Videotape someone acting out the moves, even if that person isn't the actual talent for the shoot. You should also have an appendix to the shot list with agreed-upon scene and prop measurements that correspond to the game environment.

Also include approved sketches of the game characters in costume. Anything that's flowing, such as long hair or a coat or a cape, is going to need special attention from both the animators and the studio personnel. You may be able to create a special motion capture costume or prop to track the motion of tricky costume elements. Do a test shoot to see if it works properly.

Too often, team members individually approve a written shot list without realizing that each of them still imagines the moves in a different way. These discrepancies may only become apparent the day of the shoot or worse, afterwards, when team members see the moves for the first time. Naturally, these people have other responsibilities and may not want to give the shot list their full attention during the planning stage. It's up to the director to hold meetings and discuss the moves in detail. Emphasize to the team members that their participation in these sessions will save everyone a great deal of work and frustration later on.

Planning the FMV Shoot

Usually, the production of full-motion video sequences (cut scenes) is left until the last stages of the game's development. Therefore, there's a tendency to put off the planning for the FMV shoot until the last minute. That's a shame, because you can create incredible FMV sequences with motion capture. You may be able to capture more detailed motion data for an FMV sequence because the whole thing will be prerendered and won't be competing for memory with other game elements.

The specification should provide information about all cinematic sequences needed for the game. You should have final voice-over scripts and detailed storyboards, including camera cuts, in order to create separate shot lists for FMV production.

The timing of these sequences will probably be closer to real time, and you'll have to match any planned voice-over dialogue. Depending on the type of motion capture studio you're

TABLE 2. *What to include in your motion capture shot list.*

Character name
Talent name
File name
Move name
Move description
Frame count
Loop or transition
Start position
End position
Size of capture space
Props
Special set-ups

using, you may also be able to capture the moves of two or more people at once (for example, you can do amazing fight sequences). When casting your talent, remember that these scenes also tend to require more acting ability than the in-game moves.

Getting Ready for the Shoot

WHEN TO SHOOT. At what point in the development cycle should you start shooting? Of course, your specification, animation list, and shot list must be completed and approved first. But don't let the schedule slip. As soon as you've compiled the information for your shot list and your talent is available, do a test shoot. Then start shooting, since months of post-production will be needed to get the moves into the game. Roughly, in a 14-month development cycle, principal shooting should take place from months 4 through 8, with pickup shoots possible in later months.

Try to split up your principal motion capture production into a minimum of two sessions spaced several weeks apart. Before the second shoot takes place, the team should have a chance to process most of the data from the first shoot and test it in the game. You'll want to have the team's in-depth feedback on the first shoot's results before you go back into the studio, so you can learn from any mistakes, as well as reshoot them. You may have other concerns about the way the first shoot went: maybe the talent was awful and has to be replaced; maybe the rig that you had the studio build didn't work the way that you intended; or perhaps a prop broke and you need time to have it repaired. Besides, a break in shooting will allow the talent and the team (including you) to re-energize and get a fresh perspective on the project.

FIND A STUDIO. You should make tentative reservations for the studio as far in advance as possible, whether you're using your company's motion capture studio or booking time at an out-of-house facility. If you're not sure where to find a studio, try to get recommendations from colleagues in the game business. The Web has some resources, as well; for example, the SIGGRAPH site has motion capture information, and most studios have their own sites. Some of the better known optical motion capture studios available to game developers are House of Moves in Venice, Calif.; BioVision in San Francisco; and Acclaim Entertainment's studio in New York.

When you book time for your principal shooting, arrange for a test shoot, too. Hold this session as soon as you can — it should only take a day or two. If your talent isn't available or hasn't been chosen yet, get someone else who's reasonably capable to perform the moves. (Doesn't your assistant have a secret desire to wear that black lycra suit?) Make sure that the same team members that will attend the "real" shoot are present at the test shoot. You want their participation, of

TABLE 3. *Sample entries for SUPERGUY's shot list.*

File Name	Character/ Cap. Space	Move Name	Start	End	Loop	Fr Ct	Props/Special	Description
SGSL001A	Superguy SMALL	Standing loop (rest frame)	—	—	Y	6	N	Standing, arms at sides. Rock side to side slightly, looking tough. No gun.
SGWL001A	Superguy SMALL	Walk	—	—	Y	10	N	Walk cycle, left foot first.
SGRL001A	Superguy LARGE	Run	—	—	Y	15	N	Run cycle, left foot first.
SGWT001A	Superguy LARGE	Walk to run (starting with left foot)	W	R	N	5	N	Transition from walk to run, start on left foot and end on right.
SGWT002A	Superguy LARGE	Walk to run (starting with right foot)	W	R	N	5	N	Transition from walk to run, start on right foot and end on left.
SGRT001A	Superguy LARGE	Run to walk (starting with left foot)	R	W	N	5	N	Transition from run to walk, start on left foot and end on right.
SGRT002A	Superguy LARGE	Run to walk (starting with right foot)	R	W	N	5	N	Transition from run to walk, start on right foot and end on left.
SGSA001A	Superguy SMALL	Punch forward (from stand)	S	S	N	15	N	Throws quick jab forward from standing position.
SGSA002A	Superguy SMALL	Punch right (from stand)	S	S	N	15	N	Throws right cross to right from standing position.
SGSA003A	Superguy SMALL	Punch left (from stand)	S	S	N	15	N	Throws upper-cut to left from standing position.
SGSA004A	Superguy SMALL	Kick forward (from stand)	S	S	N	15	N	Streetfighting style kick forward from standing.
SGSA005A	Superguy SMALL	Kick right (from stand)	S	S	N	15	N	Karate kick to right from standing.
SGSA006A	Superguy SMALL	Kick left (from stand)	S	S	N	15	N	Knee to opponent's stomach to left from standing.
SGSA013A	Superguy SMALL	Shoot forward (from stand)	S	S	N	15	Gun	Pull out revolver and shoot once forward. Replace gun.

course; but there's another reason to include them. You can set up procedures for communication and establish your role as the director. You'll direct the talent, call action and cut, and run the session while responding to the team's feedback.

Casting

Once you've figured out what moves you need to shoot for the project, you'll need some extraordinary performers that can fluidly and accurately play your game characters. Assuming you've decided not to hire celebrity talent, it's time to hunt down some "motion specialists" as talent for the shoot.

Noncelebrity talent. For an action game, stunt men and women are an excellent choice. (If you're planning to use a stunt coordinator, he or she should be able to bring in people for auditions.) Experienced stunt performers know how to throw a punch, hit their marks, fall on a mat, and repeat the whole thing over and over exactly the same way. Martial artists and gymnasts are other possibilities for fighting games. Check your local clubs and academies. Does your game have a medieval or sword-fighting theme? Contact fencing teams and performers from Renaissance fairs. For sports talent, try local colleges and training centers with a good reputation in your sport. If you can afford it, look into the possibility of hiring athletes from minor league teams.

Hold auditions and record them on video (this will help you get all necessary approvals on casting decisions later). Look for performers who move fluidly and have no unwanted idiosyncrasies in their motion (such as a bad knee). Motion capture technology will very accurately reflect any unusual human movement, and this will seem magnified in a game where the same ten frames are repeated often. For example, if you motion capture someone with a bad knee, the character may appear to limp when running. You should also make sure that the performer has roughly the same proportions as the game character. Height and frame are more important than weight because the actor's "skeleton" (measured by the distance between joints) will be used to calculate the character's motion. Discuss these issues with the animators, the motion capture studio manager, and the team that will acquire the motion data.

Another important casting consideration is the performer's attitude. Tell the prospective talent the bad news up front: body-clinging motion capture suit, unwieldy sensored props, repetitive moves, the need to hit precise rest frames and marks, and so on. In exchange for these somewhat bizarre conditions, plan to pay your performers fairly and treat them like royalty. Make sure that you choose people who are not only enthusiastic, but also intelligent enough to understand what they're getting into. They should also have flexible schedules and be available for pickup shoots later on. And of course, you should always have additional choices available as backup.

Remember that you can cast one person to play several characters. For our imaginary SUPERGUY shoot, I'd probably

cast three performers: a medium-build man as Superguy and Medium Henchman, a bigger guy as Big Bad Boss and Large Henchman, and a medium-build woman as Evil Villainess and Small Henchman. Also, find out from your technical team if motion from one performer can be applied to characters played primarily by other people.

CELEBRITY TALENT. One of the most common perceptions of motion capture hell is that of a shoot with a prima donna celebrity who couldn't care less about your silly little videogame. After suffering through an agonizing and embarrassing capture session, the data turns out to be useless. Don't let this happen to you. Your first step should be to make sure that the performer is right for your project, regardless of whether he or she is a famous athlete that your marketing department is hot to sign up.

Obviously, the marketing and public relations value that a star performer brings to your project is a major consideration. But there really are other concrete advantages; after all, celebrities are famous for a reason. Professional athletes will astound you with their abilities when you get them in the studio. They're fast, strong, and know their sport inside and out. Sure, you might get lucky and find an amazing local athlete for your shoot. But it's extremely unlikely that an amateur will ever match the sheer talent of a celebrity player. Think about it — these athletes wouldn't have made it to the big leagues without incredible natural ability, and on top of that, they train and play the game every day. The same idea applies to nonsports projects; for example, a celebrity actor would bring his or her own unique abilities to the shoot. Remember that motion capture will accurately represent the physical mannerisms of the performer, so your audience will recognize the moves of someone famous. (For example, check out the motion-captured Michael Jackson in his video directed by Stan Winston.) If you're creating a game based on a feature film, consider using the star's stunt double as talent for your shoot.

Still, there's no point hiring a celebrity if his or her abilities aren't appropriate for your game. Ideally, you would give a short list of acceptable celebrity performers for your project to whomever is going to arrange the deal. You can't bring these people in for auditions, but you can study footage of them doing the kind of motion you need (such as basketball or fighting). If the talent is going to play more than one game character, try to choose someone who can perform generic moves in addition to those of his or her own distinctive style. (Would you want to end up with a basketball videogame where every character — on every team — plays like Patrick Ewing?)

Before any contracts are signed, provide the talent's agent with a clear explanation of what the talent will have to do — the same "bad news" you would tell noncelebrity performers. If possible, send a copy of the shot list and a videotape of past motion capture shoots, plus exciting videogame footage showing the kind of results for which you're aiming.

Preparing a Shooting Schedule

ORGANIZE THE SHOT LIST. Before you can schedule your session properly, you should put the shot list in the order in which you plan to shoot — logically and efficiently. Group the shots by talent, size of capture space to be set up, special set-ups (such as stunt rigs), and the logical progression of your moves. Simply put, establish your rest frame positions and loops before capturing moves that branch off from those positions (Table 4).

TABLE 4. *Sample schedule for SUPERGUY shoot (main character).*

DAY 1: Small Capture Space	
Rehearsal	
SGSL001A	Stand (rest frame)
SGWL001A	Walk
SGSA001A	Punch forward (from stand)
SGSA002A	Punch right (from stand)
SGSA003A	Punch left (from stand)
SGSA004A	Kick forward (from stand)
SGSA005A	Kick right (from stand)
SGSA006A	Kick left (from stand)
SGCL004A	Crouch
SGCT001A	Crouch to walk
SGCA001A	Punch forward (from crouch)
SGCA002A	Punch right (from crouch)
SGCA003A	Punch left (from crouch)
SGCA004A	Kick forward (from crouch)
SGCA005A	Kick right (from crouch)
SGCA006A	Kick left (from crouch)
SGSA007A	Shoot forward (from stand)
SGSA008A	Shoot right (from stand)
SGSA009A	Shoot left (from stand)
SGCA007A	Shoot forward (from crouch)
SGCA008A	Shoot right (from crouch)
SGCA009A	Shoot left (from crouch)

Day 2: Small Capture Space	
Rehearsal	
SGSD005A	Defend (from stand)
SGSD001A	Get hit forward (from stand) high
SGSD002A	Get hit right (from stand) high
SGSD003A	Get hit left (from stand) high
SGSD004A	Get hit forward (from stand) low
SGSD005A	Get hit right (from stand) low
SGSD006A	Get hit left (from stand) low
SGCD001A	Get hit forward (from crouch)
SGCD002A	Get hit right (from crouch)
SGCD003A	Get hit left (from crouch)
SGLLOO1A	Lying down (not dead)
SGLT001A	Get up from fall to stand
SGXL007A	Dead
Leave time for idle moves/improvisation	

Day 3: Large Capture Space	
Rehearsal	
SGRL001A	Run
SGWT001A	Walk to run (starting with left foot)
SGWT002A	Walk to run (starting with right foot)
SGRT001A	Run to walk (starting with left foot)
SGRT002A	Run to walk (starting with right foot)
SGCT002A	Crouch to run
Stunt Coordinator: Mats & Stunt Rig	
SGST001A	Fall down (from stand) forwards
SGST002A	Fall down (from crouch) forwards
SGST001A	Fall down (from stand) back
SGST002A	Fall down (from crouch)back
SGZA001A	Special Move #1
SGZA002A	Special Move #2

Try not to schedule all of the boring moves together. If you need to shoot an entire series of simple transitions (and they don't require special props or rigs), place a few at a time throughout the shot list. If you group them together and spend over an hour having the performer act like a robot, you're likely to lose any momentum you've built up in the shoot. On the other hand, every now and then, the talent will be glad to have some easy moves — especially after some particularly difficult sequences.

DAILY SCHEDULE. Talk to the motion capture studio manager about the studio's scheduling procedures. For how many hours a day can you shoot? How much time does the crew need to prepare and wrap the studio? How long should you break for lunch, and who will order it? Is overtime a possibility? Usually, a good day's schedule consists of approximately six hours of capture time. Longer sessions may wear your talent out too early in the shoot, and won't allow reasonable time for the crew to prepare and wrap the studio.

DO YOU NEED A REHEARSAL DAY? Some people bring in the talent a day or a week before the shoot to run through the moves. You may be surprised, but I don't believe in rehearsing the performer ahead of time. I've found that it's best simply to schedule some rehearsal time at the beginning of each shoot day, or even to do a quick rehearsal before each move. Of course, it's still important to send a copy of the shot list to performers a few weeks before the shoot so they know what to expect and can make their own preparations, if they like.

Once you've figured out the schedule, don't keep it to yourself. Explain it to the team and distribute a detailed copy to all involved. It's also a good idea to write up "call sheets" for each day, indicating the time that you plan to start and finish shooting, as well as what talent, what type of moves, and which props and special set-ups are required (Table 5). A call sheet allows the motion capture studio manager and crew to have everything you'll need standing by (make sure all props are created in advance and have been approved by the studio, as well as the team). Your goal is to have as little down-time between set-ups as possible. Delays not only cost you studio time, they chip away at that all-important momentum that you've built up. Maintaining a steady pace in the shoot keeps everyone at their best.

TABLE 5. *Sample call sheet for SUPERGUY shoot.*

Day #	Of	Date	Project	Director	Crew call
3	12	4/27/98	SUPERGUY	Melianthe Kines	9:00AM

CAPTURE SPACE Small **SPECIAL NOTES** Lunch delivered 1:00PM

Move Description	Shot Numbers	Props	Rigging/Special
Rehearsal			
Run	SGRL001A		
Walk/Run/Crouch transitions	SGWT001A-2A SGRT001A-2A SGCT002A		
Falls forward	SGST002A- SGCT002A		Mats/Box
Falls backward	SGST003A- SGCT003A		Mats/Box
Special moves	SGZA001A- SGZA002A		Flying Rig/Mats

Talent	Character	Suit/Scale	On Set	Remarks
Manny Macho	Superguy	9:30AM	10:00AM	Needs trainer

Take Care of Your Performer

I mentioned before that you should treat the talent — celebrities or not — like royalty. Make sure that you've accounted for the talent's schedule, breaks needed, meal requirements, and transportation to and from the shoot. Make sure you have the proper size information for the talent's motion capture suit and footwear, and ask the talent what type of shoes he or she prefers. If the performer has a broken-in pair he or she doesn't mind selling you for the shoot, get those shoes in advance so the studio can prepare them for motion capture.

Assign the talent a gofer for the shoot day; remember that the talent may feel imprisoned in a motion capture suit. You certainly don't want him or her to try taking the suit off in the middle of the day — you'd have to recalibrate before continuing.

If the talent lives far away, arrange for a hotel room close to the studio. It doesn't have to be the Four Seasons, but make sure the performer can get a good meal, a proper shower, and a comfortable night's sleep. Arrange a car or limousine service for pick up every morning (from home or a hotel) and return every night. You'll have a better chance of getting the performer there on time and less pressure to send him or her home at a certain hour. Have the meals, drinks, and snacks that he or she prefers; but if the talent tells you that morning about a craving for Chinese food, just order it. It will be money well spent.

In other words, spoil performers and provide for almost their every need. Why? It will make them feel special and they'll like you better (that may sound trite, but it really does help). If the performer isn't worried about being hungry or how to get home, there's a better chance that his or her full attention will be on the shoot. But a more important benefit is that you, the director, will have more control over the session. Fewer surprise requests and delays equals a more productive shoot.

Maintaining control over the production process is the name of the game. You'll see how your brilliant planning can pay off when it's time to run the session and direct the shoot. I'll discuss that next month in "Directing Motion Capture for Games." ■

FOR FURTHER INFO

Acclaim Entertainment
http://www.acclaim.net/

BioVision
http://www.biovision.com/

House of Moves
http://www.moves.com/

SIGGRAPH
http://www.siggraph.org/

D eli g A di E gi e

by Andrew Boyd

Interactive music is a hot-button item for game audio these days. It could well be the most important development in game soundtracks since, well, games began having soundtracks. The promise is that the soundtrack will be as responsive to the player's actions as the visuals or the game play, and more immersive than linear music could ever be. The potential impact on the gaming experience is enormous, to say the least. But so is the potential impact on game development, as new tools and technology become available. Much of the excitement around this idea is being generated by Microsoft's forthcoming DirectMusic technology.

However, there is a chance that DirectMusic might not be the great panacea that it's touted to be, and betting a development project on largely untested technology is unwise. When DirectMusic ships, will it deliver on its great promise? If this technology doesn't live up to the hype (you can read more about it Tom Hays' article on page 20), don't forget that there are other audio engines with proven track records. In this review, I look at two of them: the Miles Sound System 4.0 (MSS) and the Headspace Audio Engine 1.0 (HAE).

Both of these engines are mature, have been used in an incredibly diverse collection of products, and were built by programmers who have listened to user feedback and have maintained an interest in keeping their products vital and relevant. That said, each product comes with its own set of particular strengths and weaknesses, and I'll explain those as I go along.

I approached this review as a composer and sound designer working in games — I'm not a programmer. So my perspective of these products is from the standpoint of someone who creates content to be submitted to the systems. As such, I'm in a position to judge the quality of the audio produced by these engines. However, as these are both fundamentally programmers' tools, I teamed up with a game programmer who put together some demo applications, analyzed the documentation, and generally gave me a feel for working within these environments.

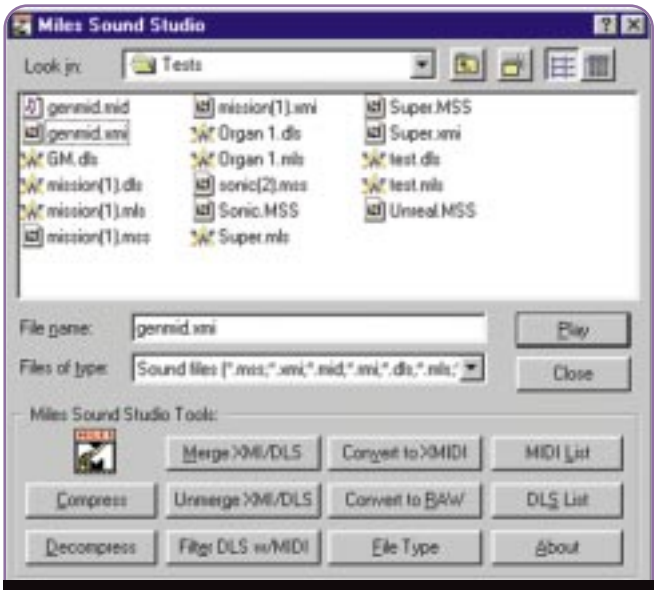
The Miles Sound System 4.0

John Miles introduced the Miles Sound System in 1991 and RAD Game Tools now distributes and supports it. The MSS technology is everywhere; RAD claims that over 800 games have shipped using the MSS, and even DirectSound uses MSS technology, as the DirectSound audio mixer was licensed from RAD. What this means is that the technology has been beaten on and abused by huge numbers of programmers, and the tech support people have heard nearly every kind of problem imaginable.

In MSS 4.0, RAD provides the programmer with a consistent, easy-to-use, and functionally transparent front end for sound and music programming. MSS provides a consolidated interface for just about everything you could

Andrew Boyd has been creating sound for games since 1993. He now runs Audible Images, a music and sound design house in San Francisco, Calif. He can be reached at andrew@audibleimages.com.

Headspace's Beatnik Editor



The Miles Sound Studio

want to do with digital audio, MIDI, and Red Book sound formats. It contains a set of prebuilt authoring tools as well as functions for creating custom tools. Not only that, MSS further reduces the complexity barrier by including the Quick Integration Services, which provides basic sound playback functionality with almost no programming work at all (startup, load and play, unload, and shutdown calls are all that is necessary to get going). It's not particularly feature laden, but it doesn't get much easier. Additionally, RAD ships the full MSS source code along with the API, which is a very nice touch. Most important to my purposes here, though, is that MSS now includes a full-featured software synthesizer based around the MIDI Manufacturers Association's DLS standard, the same kind of synthesizer that DirectMusic will use.

AUTHORING CONTENT FOR MSS. In its most basic form, you can just hand MSS a .WAV file and play it with amazing simplicity. Similarly, you can create a Red Book format audio CD, and MSS can start playing that, too. In fact, MSS can handle existing content with no problem at all and, in fact, can do some very interesting things with it. For instance, playing multiple sounds simultaneously is easy, and MSS handles all resampling chores for you at run time, should your sounds happen to be in different resolutions. But to take advantage of the sophisticated features of the MSS software synthesizer, you need to provide your content in some unique formats. The most important formats are the downloadable sound (DLS) collections used by the software synthesizer.

MSS doesn't provide any tools for creating the DLS collection files needed to feed the software synthesizer (though RAD is considering it for future versions). Currently, the most commonly used tool is the one released by the MIDI Manufacturers Association when they put out the final DLS specification. I didn't have a chance to look at that tool for this review, but I understand it's hardly the kind of mainstream application that will be on every musician's computer. In any event, RAD's implicit assumption seems to be that because DLS is an open standard being rapidly adapted by the industry, it shouldn't be too hard to find good tools to make DLS content in the near future. I think this is a perfectly reasonable assumption — we're not quite there yet, though. For my tests, I used a shareware product called Awave 4.5 (available at http://hem.passagen.se/fmj/fmjsoft.html), which did a fine job of editing DLS content. In fact, it can actually read directly from MSS format files, compressed or uncompressed, and RAD speaks well of it as a companion to their software synthesizer.

The tools that MSS provides aren't pretty or particularly easy to use, nor do they feel as polished as Headspace's Beatnik Editor. Nonetheless, the Miles Sound Studio, the Miles Sound Player, and MIDI Echo have some powerful features. The Sound Studio is a convenient, consolidated set of tools for compressing, decompressing, converting between, and generating information about many of the various file formats that MSS uses. With Sound Studio, you convert standard MIDI to XMIDI, merge XMIDI and DLS files, compress DLS instruments, and compress .WAVs or convert them to .RAWs. You can strip DLS collections back out of merged files, or generate a list of all the DLS parameters in a collection.

One great tool is the "Filter DLS with MIDI" function that analyzes a selected DLS collection against a MIDI file (or files) and throws out all instruments not actually used in that file or files. This can provide a huge savings in file size, especially if you're using the Microsoft's General MIDI set. Combined with the fact that the instrument sound data can be ADPCM (Adaptive Differential Pulse Code Modulation) compressed, some very small files are possible indeed. This ADPCM compression can also be done in the Sound Studio and applied to any .WAV file or DLS collection.

The Sound Player application is a very simple, straightforward program that does just what its name implies. It's convenient because it allows you to load different DLS collections, compressed or uncompressed, and play sequences with them. It also allows you to turn reverb and antialias filtering on and off, and switch between mono and stereo; 16-bit and 8-bit; and 11KHz, 22KHz, and 44KHz rendering, all while a song is playing back. It monitors

processor usage during all this, too, so you can see very clearly the resource hits that the various options take.

MIDI Echo is the least visually attractive of the tools as it's a console application — it runs in a DOS box and uses character graphics to display information. On the other hand, it provides functionality that can be very useful. MIDI Echo lets you audition the software synthesizer from a MIDI controller or pipe a sequencer directly through the software synthesizer, so you can make changes before going through all the conversion processes necessary to prepare a file for final delivery. Unfortunately, I was unable to use this tool to any real extent because of the way my Turtle Beach sound card is set up. Apparently, MIDI Echo tries to allocate some sound card resources that the Turtle Beach card doesn't allow to be allocated all at once. Be forewarned: if you use a Turtle Beach card in your sound development machine and you want to run this tool, you'll probably want get a Sound Blaster to go along with it. (Turtle Beach says the problem can be worked around by adjusting the card's IRQ settings.)

The complete set of MSS tools is available from the RAD web site, along with some nice sample music files and a demo showing the system's ability to play and stream audio and play its software synthesizer. It's worth a look.

SUPPORT AND DOCUMENTATION. The MSS documentation is vast, thorough, and meticulously detailed. It gets out of your way and tells you what you need to know quickly and concisely. The tools' documentation could be more engaging, perhaps, because the users of the tools are less likely the bedraggled programmer than the sound designer; still, the information you need is there.

Technical support is one of RAD's real strengths. The company is extremely helpful, responsive, friendly, and accessible. When I called to inquire about MIDI Echo's resource allocation problem, I was put right through to a person with specific knowledge of my problem. Though the problem yet remains, I'm now confident that the fault lies with the Turtle Beach sound card, not the software.

By the time you read this, RAD should be shipping a new release of MSS. The new version adds native support for MPEG Layer-3 compressed audio, a new effects architecture that can apply effects such as reverberation and chorus to all audio, and more.

The Headspace Audio Engine

HAE began life in 1991 as a way to play standard MIDI files on a Macintosh by creating a wavetable synthesizer entirely in software. The resolution was poor, the playback was limited, but it worked. Soon its author, Steve Hales, had grown it into a product called SoundMusicSys, which could handle all of a game's sound implementation. SoundMusicSys was marketed under a couple of different brands and was used in a number of very successful games. After the tool went through several revisions and was ported to Windows and some set-top boxes, Headspace bought and renamed it. HAE now forms the technological underpinnings of Headspace's various products.

HAE is a unified, full-featured synthesis, sampling, and streaming audio environment that runs entirely in software and with support for multiple platforms. Its software synthesizer is somewhat more sophisticated than the DLS synthesizers in MSS or DirectMusic, especially in terms of modulation and effects parameters, and it defaults to 64 available voices. Because of its cross-platform heritage, it supports a large number of file formats. It also uses very robust filtering mechanisms to create very high overall output quality, even when doing wide pitch bending or other forms of drastic resampling (which, of course, it can do on the fly at run time).

The Headspace Audio Engine's API is very deep and flexible, but is also rather complex. It has a different feel than MSS — for instance, when you want to stream a file from disk, you must start a thread in your application to service the stream very often to keep its buffers from under-running; MSS will, by default, handle all of that service for you. (Of course, MSS allows the default to be overridden in order give control over to the application.) The difference is in the approach to development. Both products provide very powerful toolsets, but HAE is more focused on raw flexibility than simplicity of implementation. Many programmers will find this approach acceptable, even preferable, because it places the onus on the programmer to code the low-level sections of the audio system. To give you an indication of its depth, HAE provides functions to stream audio from any source — and the source doesn't have to be an audio file. You can generate sound-like data from anywhere, and it will generate its own MIDI information from these triggers.

AUTHORING CONTENT FOR HAE. HAE can work with a wide variety of standard sound formats, including .WAV, .AIFF,

Comparative file format support.

	Miles Sound System 4.0	Headspace Audio Engine 1.0
Audio Formats	.WAV, .RAW, .VOC, native support for IMA ADPCM compression	.WAV, .AIFF, .AU, .MOD, and Sound Designer II in Beatnik Editor
Software Synthesizer	DLS-1 compatible, per-voice low-pass filter, global synthesizer reverberation	Proprietary wavetable/sample playback synthesizer, resonant filter, reverberation
MIDI Formats	Standard MIDI, XMIDI	Standard MIDI, RMF, Direct MIDI Messages
Mixing Features	Run-time resolution-independent mixing of streamed or triggered sounds, with volume, pan/balance, and pitch	Run-time resolution-independent mixing of streamed, triggered, or generated sounds, with volume, pan/balance, pitch, and reverberation

.AU, MIDI, and .MOD files. The support for .MOD files is intriguing, because in some ways the .MOD file is the granddaddy of this current round of software synthesis. While .MOD is much more popular in Europe than in the U.S., there is nevertheless an amazing amount of content out there in .MOD format. It is apparently difficult to author content for, however, and Headspace doesn't provide any .MOD authoring tools. Still, this discrepancy is an indication of HAE's cross-platform roots. HAE also has its own proprietary format, .RMF (Rich Music Format), and you can author .RMF files using a free-standing application called the Beatnik Editor (see "The Beatnik System").

SUPPORT AND DOCUMENTATION. The HAE documentation is extensive and very thorough, and it comes as a conveniently indexed .PDF file. The documentation's tone, however, is a bit breezy. I guess it's cute for a bit, but it does get tiresome — and I know more than a few programmers who would be seriously unamused by reading the little side jokes while looking for some feature's documentation (at 3:00AM… the day before final…).

I received courteous and thoughtful technical support, and Headspace's support personnel did fix the problem I brought to them. But my first e-mailed request for support went unanswered until I followed up with a phone call. As with MSS, if you need programming support, you can actually talk to the author of the system, and that certainly inspires confidence and usually gets quick, correct answers.

Sound Quality

I have little to say about the sound quality of either system's straight digital audio or Red Book playback. MSS and HAE can just pipe out the audio exactly as it sounded when it went in. Their mixers also do a fine job of combining multiple sounds without creating audible artifacts (neither has an intrinsic limit to the number of sounds it can play, but each provides a function to limit voices so as to not overtax the hardware). Once you begin to alter sounds, though, differences appear. Both systems can take sounds of different sample rates simultaneously and resample them on the fly to into one stream, but HAE was capable of slightly better sounding output. In one simple but revealing test, I switched the output rate of the playback engine between 22KHz and 44KHz while listening to different combinations of three streams — one at 22KHz, one at 32KHz, and one at 44KHz. This forced the systems to perform different degrees of simultaneous up- or downsampling of the streams in order to mix them together. I also compared the output of the up- and downsampled files against a reference file resampled offline using Sound Forge. With identical source files, the two systems produced slightly different audible results.

HAE's upsampling sounded especially impressive — smooth and fairly natural — but it came with a processor hit of an additional two to three percent per stream (naturally, it takes more processor time to render a 44KHz output stream than a 22KHz stream). HAE downsampled the 44KHz to 22KHz quite well, and with little or no additional processing time; the resultant sound was somewhat duller than the reference downsampled file, but was otherwise artifact-free. The 32KHz file didn't downsample all that well, sounding soft and muddy, but performance remained consistent. MSS had more trouble with the upsampling,

The Beatnik System

Headspace, as a company, has credibility to spare when it comes to the games industry. Steve Hales, HAE's architect and the author of SoundMusicSys before it, has been involved with audio for games for a long time. His technology has appeared in some major games, such as LEMMINGS, HEXEN, a couple of WING COMMANDERS, SIMCITY 2000, and more. Plus, soundtracks composed and designed by Headspace's creative types have been featured in high-profile games such as CYBERIA and OBSIDIAN. So why is HAE not on the tip of every game developer's tongue? Because the company currently is focusing its promotional and evangelical efforts on the Internet community. The Beatnik System consists of a web browser plug-in, an editor application, and a set of web-based resources for users. Headspace is making good progress in the web industry, which could draw some focus away from the games side of its business. However, the company claims to remain committed to the games industry, saying it does have some big game development clients using its technology, and that it's interested in working with more developers. Also, because the whole Beatnik System is built on HAE technology, there are some happy side effects for game sound development. One of these is the Beatnik Editor.

The Beatnik Editor is Headspace's proprietary tool for creating .RMF documents. It's a MIDI processor, a wavetable creation device, a real-time controller, a compressor, and an encryption tool wrapped up in a fairly musician-friendly package. Because it's aimed at a much more general audience than just hardcore developers, the Beatnik Editor has the look and feel of a mainstream application. As of this writing, it's available as a beta release, and only as a Macintosh application. On the plus side, it's free, and it's useful for both HAE and Beatnik-specific content. Headspace says it's working on a Windows version of the Beatnik Editor, but the company isn't ready to announce a release date for it yet. It has released a Windows product called the Beatnik Converter, but it's a comparatively simple format converter, not a complete production tool like the Beatnik Editor.

The Beatnik Editor is fairly easy to use. You load in sound files (in Sound Designer II, .WAV, .AIFF, or .AU format) and turn them into instruments. The editor provides a very useable graphical interface for defining keymaps, envelope, filter, pan, and so forth. You can audition these instruments from the onscreen keyboard or link to your sequencer and play them directly from a sequence. You can create a bank of these user instruments to go along with the supplied General MIDI bank, and you can also copy instruments from the GM bank, edit them, and save them in the user bank. You can then load in standard MIDI files that call these instruments and save the whole thing out as a compressed .RMF file suitable for use inside HAE or with the Beatnik plug-in on any web page. If you only use instruments from the GM bank, the resulting file is very small, as the GM bank is always resident in the playback systems.

generally adding just a bit of buzz to high-frequency content. This was very slight in most cases, but audible in an exposed sound — in a mix, it was much less noticeable. The process incurred no noticeable performance hit. Downsampling was fine with MSS; the 44KHz to 22KHz downsample sounded a little less dull than the same process performed in HAE, but still reasonably smooth. The 32KHz to 22KHz process was actually superior in MSS, but still didn't sound too good.

While both audio engines can apply a reverberation effect to audio streams during playback from their software synthesizers, don't expect much in terms of sound quality. Processing something as complex as reverberation in real time on a PC, without using up all available resources, is just too much to ask. Of the two products, HAE has an edge in quality, with a decay that at least hides the individual echoes — a small consolation. Both produce gritty and artificial sound, with a metallic quality that's not at all pleasant and really does little to provide any kind of space around sounds. However, in some cases, it's better than nothing; it's awfully nice to have a reverberation option if you need it. For example, because you can make completely custom sounds, it's possible to create your most important instruments (or even vocals) as loops, phrases, or hits with high-quality reverberation during recording, then create all the other instruments from the wavetable to save space. Applying global reverberation at playback will help to mix all these elements much more satisfactorily. Check out the demo song SUPER.MSS on the RAD web site for an example of this process. Using the Miles Player application, turn reverberation on and off and listen to the way the mix works. If you have a DLS editor, you can even pull the file apart and look at how the vocals were created.

Both audio engines can also compress sound files and synthesizer samples using IMA ADPCM compression. This provides a four-to-one compression of 16-bit sounds with surprisingly little loss of audio quality. Still, compression does cause some audible artifacts — a certain grittiness and noisiness added to the sound. However, in most cases, I suspect the impressive size savings will make the compression worthwhile, especially inside MSS where the compression provides savings both in storage and in RAM requirements. HAE also offers a proprietary lossless compression for synthesizer samples that results in about a 20 percent size reduction.

Performance and Resource Use

I did some rough performance monitoring of these products using Windows 95's supplied System Monitor application for processor usage. These tests were run in Windows 95 on an Intel Pentium 200MMX with 64MB RAM, Turtle Beach Pinnacle sound hardware (driver version 4.03), and the DirectX 6 beta.

This is the part of my article that sound designers will want to keep away from the programmers doing the technical implementation of their game. Here's a fact: software synthesis, runtime mixing, audio decompression, musical interactivity, and many of the other tools that these systems offer to make a sound designer or composer happy absolutely devour system resources. Think 20 to 30 percent processor usage on my test machine for what seemed to me a reasonable combination of software synthesis and sound playback. Granted, my test system is a low-end target machine for games beginning development now, and presumably a faster machine would report commensurately lower resource usage.

It wasn't difficult to get MSS to consume 40 percent of the test machine's processor cycles consistently by simultaneously playing the software synthesizer (using a sequence averaging around 16 voices and reverberation, filtering, and 16/44/stereo rendering all on), triggering sounds out of RAM, and streaming sound from disk. When playing multiple streams along with an .RMF file, the HAE exhibited somewhat less predictable performance, jumping from as little as 20 percent to peaks as high as 50 percent of resource usage, but stayed in about the same range as MSS. The ability to handle all of this simultaneously opens up some wonderful sound design possibilities, but we know that performance will always be an issue with games, so tradeoffs will still have to be made.

While both audio engines can compress the sounds that will be used in the software synthesizers, MSS has a bit of an advantage in that you don't have to decompress sounds before playing them. MSS decompresses directly in its mixer (the last stage before the sound is sent out to hardware), so it can use as little as one fourth the RAM required in uncompressed form. Interestingly, this takes essentially no performance hit during playback, because while it requires two to three times as many cycles to perform the decompression, there are one fourth as many samples to process. It's almost like getting the RAM for free — you just have to be willing to put up with a small degradation in audio quality.

Do You Need These Systems?

At the moment, each of these systems offers a killer-app feature: the software synthesizer. However, once DirectMusic ships, with its support for a DLS software synthesizer, complete tool set, and its low, low price (free), will HAE and MSS still matter? I think they will, because they provide much more than just the synthesizer. MSS offers a very solid, easy-to-use API layer that can save a great deal of time and hassle, and RAD is committed to adding features to stay current. Frankly, it's conceivable that the MPEG Layer-3 playback in the forthcoming version will be reason enough to license the whole package. HAE provides a level of cross-platform capability with which, obviously, the Microsoft product will not even try to compete. And at least for now, Headspace has an edge in the authoring tools with its Beatnik Editor.

The reality is that while neither MSS nor HAE is free — and DirectMusic is sure to cause RAD and Headspace some headaches and sleepless nights about that issue alone — they are also proven, known products. When it comes time to consider an audio platform for a game, it would be irresponsible not to take that into account. Both products have been around for so long that generations of games have come and gone during their lifetimes — in fact, the very concept of what game audio should be has evolved a long way. In the meantime, both products have grown into extremely powerful, robust tools for supporting audio content.

There is a lot to like about the

Headspace Audio Engine, and if your project could use its impressive cross-platform capabilities, or if, say, your sound designer already has a great deal of content prepared for the Beatnik system, then it could prove an excellent choice. It certainly does everything it claims to do, and it produces high-quality audio. But without considering the needs of a specific project, if I were to recommend one of these systems for PC game development, I'd have to go with the Miles Sound System. Version 4.0 is an excellent system — robust, stable, and easy to use. It has the tools and features necessary to build an interactive audio system of any complexity and it supports industry standard formats. If the next version really incorporates all the features RAD is promising (and since they previewed it at the CGDC last May, I have little doubt that it will), it should prove to be a powerhouse of an audio platform for PC gaming. ■

Miles Sound System 4.0

Rating (out of five stars): ✪✪✪✪

RAD Game Tools
Kirkland, Wash.
(425) 893-4300
http://www.radgametools.com

Software Requirements: The audio SDK supports Windows 95/NT/3.x, Win32s, and DOS.

Prices: $3,000 per shipped title (source code included). Various site-license options available.

Pros:
1. Clean and elegant API.
2. Industry standard DLS software synthesizer.
3. Run-time decompression of IMA ADPCM compressed sounds.

Cons:
1. Lacks a dedicated or integrated authoring tool.
2. Supports Intel-based systems only.
3. Tools' interfaces need polishing.

Headspace Audio Engine 1.0

Rating (out of five stars): ✪✪✪½

Headspace
San Mateo, Calif.
(650) 696-9400
http://www.headspace.com

Software Requirements: The audio SDK supports Windows 95/NT with DirectSound, and MacOS.

Price: $5,000 per shipped title per platform. Embedded version or source license negotiable.

Pros:
1. Incredible cross-platform support.
2. A well developed and easy-to-use authoring tool.
3. Excellent sound quality.

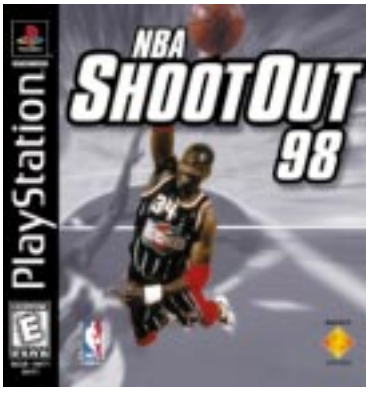
Cons:
1. Proprietary format for software synthesizer.
2. Authoring system available only for MacOS.
3. More expensive to license.

POSTMORTEM

989 S di NBA SHOOTOUT 98: S d f Vic

by Chuck Carr

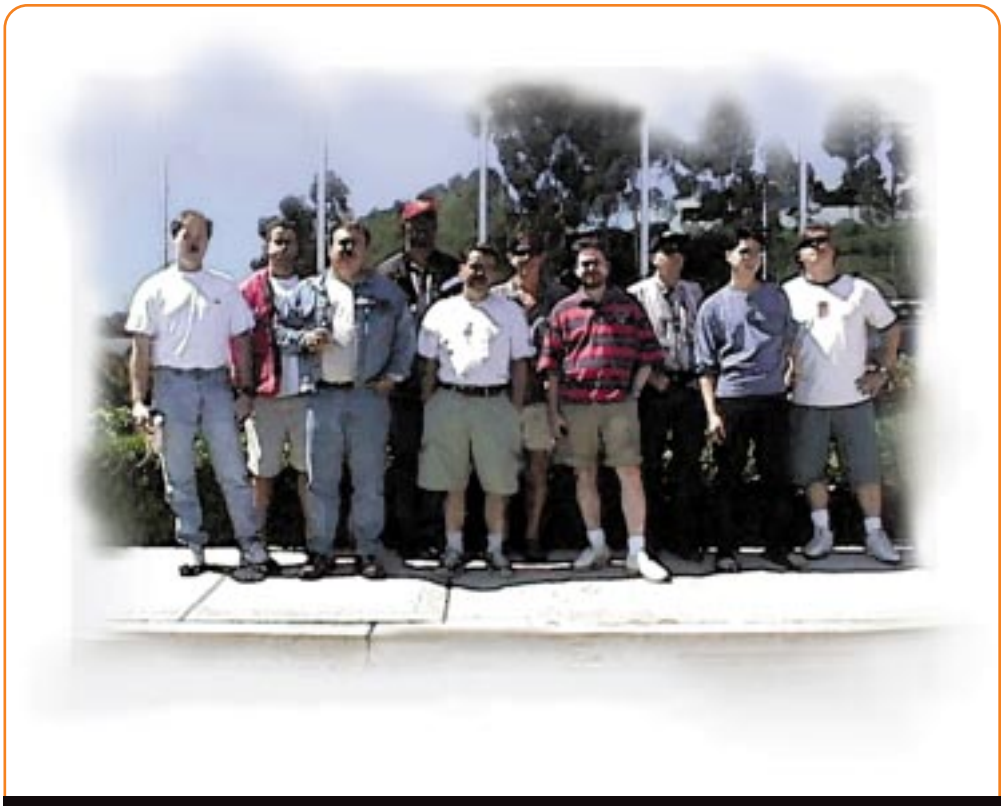
NBA SHOOTOUT 98, a simulation/arcade basketball game for the PlayStation, was developed by Sony Interactive Studios America (now known as 989 Studios). 989 Studios, in cooperation with Sony Computer Entertainment America, produces the majority of the sports titles that are available for the PlayStation today. When *Game Developer* magazine approached me to write this article, I wasn't sure how an audio guy such as myself could stay true to the guidelines of a Postmortem, seeing that it's usually written by game programmers and producers. Only after input from a reader



Chuck Carr is a composer/sound designer/audio engineer at 989 Studios in San Diego, Calif. (formerly Sony Interactive Studios America). He has been in the audio industry for 13 years and the gaming industry for 5 years. He can be reached at gdmag@mfi.com.

focus group held by the magazine at last spring's CGDC did I get some answers to that question.

First, let me start by telling you a little about the game itself. Chris Cutliff, NBA SHOOTOUT 98's producer, describes the game as, "a simulation/arcade basketball game intended for all skill levels, from the novice to the advanced game player." The game uses licenses from the NBA for teams, players, uniforms, and logos. With such an incredible license, the audio team's goal was to create NBA-quality sound to complement this popular title.



The NBA SHOOTOUT 98 development team, including several of the game's audio designers. Left to right: Scott Murray, Chris Cutliff, Geoff Goldberg, Algon Leighton, Mike Bolger, John Settles, Joel Copen, Rex Baca, Fred Shic, and Andre Booriakin.

A Tough Act to Follow

NBA SHOOTOUT 98's predecessor, NBA SHOOTOUT 97, was a great game that did very well. When development of SHOOTOUT 98 began, our San Diego, Calif.-based audio team (consisting of Rex Baca, Joel Copen, and myself) was relatively new. Rex had been at 989 Studios from the beginning (when we were known as Sony Imagesoft), Joel for about nine months, and I for about four months. Rex created sound effects and produced the voice work talent for SHOOTOUT 98 (which was voiced by Mike Carlucci, who also contributed his talents to NFL GAMEDAY 98, MLB 98, and MLB 99). Joel wrote four of the in-game music pieces. My job was to compose the front-end song and sound effects (the front end is the point in the game where you choose your options and preferences before game play begins), compose one in-game tune, and like Rex, create sound effects such as exaggerated net swishes, bouncing basketballs, squeaking shoes on a court, and so on. In addition, all three of us had the grueling task of regionalizing hours of voice work.

Early on, we had to make some important decisions about the audio. We decided to go with a contemporary hip-hop/R&B music style for the in-game music. The in-game music consisted of six songs, which were played during the introduction movie and at various stages throughout the game. The songs were in Red Book audio format. Finally, we decided to rerecord the sound effects from SHOOTOUT 97 and add some incredible 3D stereo samples into the mix, such as ambient room basketball bounces and shoe squeaks. We knew what kinds of sounds we were looking to get before we started recording and discovered some great audio nuggets such as shoe stomps, grunts, and springy basketball rims after listening to the finished recording.

Working with the audio for the play-by-play announcer in SHOOTOUT 98 was especially challenging. As I stated earlier, Mike Carlucci was used to introduce players, announce stadium names, call the game as it's played, and so on. Our plans called for his commentary to be streamed as .XA audio from the CD during game play. As with most sports titles, this meant that many hours of voice work had to be regionalized and organized.

The process of regionalizing the audio was an arduous task. During a few five-to-six hour sessions at Studio West in San Diego, we recorded Carlucci reading a script of player names, game calls, and in-game commentary onto a DAT (Digital Audio Tape) cassette. The development team members wrote the script, with additional editing help from Rex. We then dumped (recorded) these recordings into a computer in real time, including all of the mistakes, ad-libbing, and empty space between phrases. Keep in mind, these files are sometimes hours in length; at a rate of 16-bits and 44.1KHz, they can consume gigabytes of disk space. Regionalizing is the process of selecting only the desired

NBA SHOOTOUT 98

989 Studios
San Diego, Calif.
(619) 824-5511
http://www.989studios.com/
Team Size: 8 (11 including audio team)
Time in development: 9 months

The author recording sounds of the wild with the Crown SASS MK-II microphone and a Sony TCD-D10 Pro II portable DAT machine. (Photo by Dominic Perricone.)

player name, game call, or phrase of audio, while making sure no extra space is selected before or after the selected audio (thereby creating regions). Keeping your region selections tight, without extra space, helps keep file sizes smaller, thereby leaving more space on your game CD. These regions are then saved as individual sound files, which are then saved in the appropriate format and used in the game accordingly.

Our Audio Gear

As far as audio gear is concerned, we had everything we needed to get started. We employed a Crown SASS MK-II microphone, a Sony C-70 shotgun microphone, and a Sony TCD-D10 Pro II portable DAT machine to record sound effects, which I'll discuss later. We used both PC and Macintosh platforms to process and edit the audio. Our core tools included Digidesign's Pro Tools 4.1, Sonic Foundry's Sound Forge 4.0d, Digidesign's Sound Designer 2.82, Syntrillium Software's Cool Edit Pro 1.1, Cakewalk 6.01, and Opcode's Studio Vision Pro 3.5.

Sound Forge 4.0d, which I believe is the best software of its kind on the planet, was my workhorse tool for sound effects editing. It has very useful and functional features for the audio professional, tremendous third-party support, and so far, all product upgrades have been easy to attain via Sonic Foundry's web site, free of charge.

Cakewalk 6.01, which I've been using since the first Windows version, was the tool I used to compose my music. With version 7.0 out now, it's more of powerhouse than before, although it still has one huge drawback: no .AVI or QuickTime support for importing videos. This support is essential for scoring music and sound effects to video. I talked with Greg Hendershott of Cakewalk at last May's CGDC about this problem, and he said that this feature would be implemented in a future upgrade.

I used the awesome noise reduction feature in Cool Edit Pro 1.1 for cleaning up audio, as well as some of the cool effects algorithms for tweaking some front-end sounds. Pro Tools 4.1, Sound Designer 2.82, and Studio Vision Pro 3.5 were used on the Macintoshes.

Things That Went Right

1. A LARGE ALLOTMENT OF RAM DEDICATED TO AUDIO. With PlayStation games, the trick is to fit songs and sound effects into the PlayStation's 512K of sound RAM. Usually, one doesn't have the luxury of having all 512K to work with; you may have to share that RAM with another sound designer's sound effects or music, or some of the sound RAM may be allotted for sound effects that will be streamed off the CD. But front-end programmer Fred Shic let me have it all, so I alone was in charge of how we'd use the full 512K.

I decided to take full advantage of the space by creating short stereo 3D sound effects. The particular kind of

3D sounds that I implemented weren't achieved by using a 3D sound API or surround sound decoder; I'm talking about simple binaural audio. Binaural audio doesn't require any encoding or decoding at all. All that's required for playback are left and right channels, and all you need to create the sounds is the appropriate microphone (in our case, the Crown SASS MK-II). Although binaural audio has some limitations when compared to using an API or decoder, it works exceptionally well for ambient environmental sounds and one-shot sound effects.

2. RECORDING OUR OWN SOUNDS. Using sounds from a sound library is sometimes your only choice; Nonetheless, when the situation presents itself, recording your own sounds is often the most rewarding experience. In the case of NBA SHOOTOUT 98, we rented a local gymnasium for two hours to record our assistant producer and his brother playing basketball. We used the Crown SASS MK-II stereo ambient microphone to record the big 3D sounds, the Sony C-70 shotgun microphone for close-up in-game sounds, and the Sony TCD-D10 Pro II portable DAT machine to record it all. Using the assistant producer as our recording talent had several great advantages. We realized some cost savings by using our own team members for this recording session, and it was fairly easy to schedule. Our assistant producer knew the sounds that we were looking for, which later helped take some of the guess work out of choosing which sounds to use. It may seem easy to choose which sounds to use in a basketball game (and most times it is fairly easy), but having the assistant producer with us as our recording talent helped him remember what source material we had to work with later.

3. Success with noise reduction tools. How many times have you recorded what you thought was some great audio footage and had it ruined by too much room noise? That used to a big concern for me (and, to some extent, I still worry about it), but thanks to some of today's newer audio software, recording in noisy situations has become a little less troublesome. For instance, while in the gym I noticed a low tone coming from the ventilation system, which kicked on and off periodically. One time while it was on I recorded the hum by itself, making sure that the talent didn't make any noise. We then went about recording our effects. Later in the studio, I used the noise reduction feature in Cool Edit Pro 1.1 to remove the noise from the usable effects. With very little tweaking of parameter settings, the noise was all but gone from the source material.

4. Working closely with programmers. Even before creating the first sound effect for the front-end sections of the game, I met with the front-end programmer, Fred Shic. Fred was responsible for programming the opening menu screens, including assigning sound effects to the various PlayStation controller button actions. We discussed the sound needs and began to formulate ideas about what sounds were required for each button-action. As I created sounds, I would fire them off to Fred over the network so he could plug them into the game and test them. I would send him these sounds a couple of times per week. His feedback helped me determine the correct length for sounds attached to any given screen action, and his careful accounting of sound RAM let me choose the optimal sample rate for each sound. This close communication really helped us make the best use of the available sound RAM, and also ensured the highest-quality audio and shortest load times possible. I realize that some development teams don't have the luxury of this kind of a close working relationship — many use third-party developers or have programmers who stay locked up in their rooms most of the time — but it certainly made a big difference in our case.

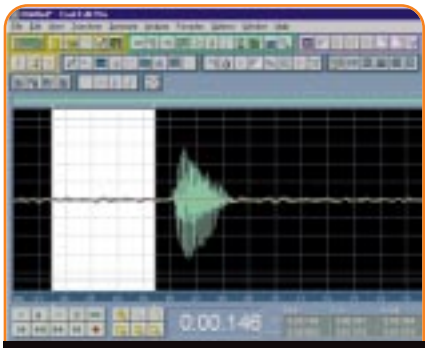
5. The Yamaha 03D digital mixer. I'll let you in on a little secret. Using the Yamaha 03D digital mixer is a joy I never felt before until I used it to mix my songs in NBA SHOOTOUT 98. About a month before the project began, I ordered the 03D and started to work with it. I had heard a lot of great things about the 03D, and I can attest to its quality. I mixed every track entirely in the digital domain, complete with digital dynamics, effects, and equalization. Using Cakewalk Pro Audio as my multitrack software, I was able to use software plug-ins on all sixteen tracks in real time, automate the 03D via MIDI during mixdown, and save all of the different mixes for later changes. Top this off with the desk space and money I saved by not using any outboard gear whatsoever. Simply put, I love new, functional technology.

What Went Wrong

1. Poor communication. In the beginning of the NBA SHOOTOUT 98 sound development process, the desired feel of the music as explained by the development team wasn't exactly clear. As songs were written and presented to the team though, the consensus seemed to be that the music was headed in the right direction. But as the beta date approached, opinions about some of the songs changed; it began to look as though some of the tunes weren't going to cut it.

Consequently, we set up a meeting with the producer to discuss the sections of some of the songs that needed changing. In the meeting, we discovered that some of the development team members who had initially voiced dissatisfaction with the songs did so mostly because they thought that the majority of the team didn't like the music. As it turned out, there was less dissatisfaction with the music than we had initially thought. So we went back and fixed the parts of the songs that didn't work and were able to deliver the goods. I now know how important it is to really get to know the development team and producers as you work on a title. My experience has shown me that good communication will often result from close relationships.

2. Cross-platform difficulties. I'd really like to have an easy method of exchanging and manipulating standard audio files on both the PC and Macintosh platforms. For instance, let's say you convert a .WAV file to an .AIF file and save it to a folder on a network drive. You'd think that a Macintosh would be able to recognize it, right? (After all, that's the main purpose of the conversion in the first place.) However, that was simply not the case in our experience with NBA SHOOTOUT 98. The Macintoshes wouldn't recognize the .AIF files that we'd already converted and saved on the PC. We had to reconvert hundreds of files to .AIF format on a Macintosh. The reason is that Sound Forge and other PC audio programs don't embed into their audio files the contextual information that the Macintosh needs to identify these files. (For the record, I don't have a problem reading Macintosh files from the PC.) Thankfully, Sonic Foundry has a little utility program for the Macintosh called AIF Typer (available at Sonic Foundry's web site). You simply drag .AIF files onto the AIF Typer icon on the Macintosh desktop, and the utility does the rest. Another way to make the Macintosh recognize a PC .AIF file is to



Extracting a noise profile with Syntrillium Software's Cool Edit Pro 1.1. The designer will use the highlighted section to remove the background noise from the file.

edit the file creator information with a utility such as ResEdit or SetItsType, but I don't wish this chore on anyone. With AIF Typer, I can now send files back and forth between the Macintosh and PC with little heartache.

3. Out with the old, in with the new. The PlayStation's native file format for audio effects is called a .VAG. Collections of .VAGs are organized into what is known as a .VAB using an application developed by Sony called Sound Delicatessen. A .VAB also holds the instructions for how the individual .VAGs are played back in a game (for example, volume, panning, tuning, effects depth, envelope parameters, and so on). Early in our development process, we used the old .VAB from NBA SHOOTOUT 97 as a placeholder for the programmer. As development proceeded, we created a new .VAB that contained new sounds.

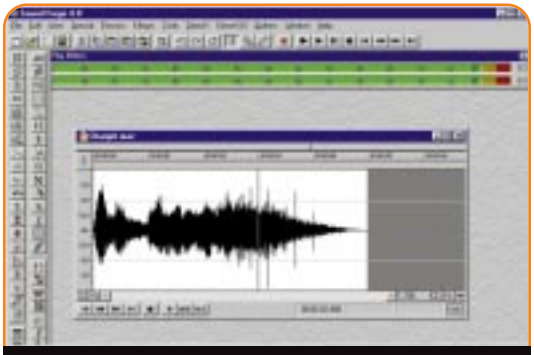
Then one day I went to Fred Shic's office to hear the latest sounds in the game. For some reason, the game was emitting some of my new front-end sounds along with some of the old sounds from the old .VAB. Fred assured me that this problem would be fixed.

As time went by and the beta date drew near, I completed the final version of the .VAB and handed it off to Fred. When I heard the latest CD burn of the game, some of the old front-end sounds were still being played. After some troubleshooting, we discovered that the new .VAB was in place, but that the old placeholder code was still calling the old .VAB. Fortunately, this was an easy fix. The error was completely understandable considering the programmer's workload at the time, but still, during a game's development process, it's a good idea to review the latest revisions of the game and make sure the audio is being implemented properly.

4. Not relying on an uninterruptable power supply (UPS). Picture this scene: I'm working out one of my last tracks — the drums are groovin', my bass is pumpin', and I'm feeling that this is going to be a great track for the game. On top of that, I'm ahead of schedule. Suddenly, I notice some audio stuttering in the playback. I don't think it's a big deal, but just in case, I check to see if I'm running out of space on my hard drive. Sure enough, I notice I have only 2.4MB of space left, and, just in time, I switch the recording over to a drive with plenty of capacity. Once again, I'm feeling pretty smug and continue composing. Suddenly, the power goes out.

I sit in a moment of blackness wondering if I saved my work. I didn't. I now use a UPS. I highly recommend you use one, too.

5. Not using the latest software. When we were first deciding on a format in which to create our game's songs, we looked at MIDI, Red Book audio, .XA audio, and looped .VAGs. I knew that I had to use my Macintosh to compose these songs, as my Nu-Bus Sound Artist audio card was only available for the Macintosh at the time. The Sound Artist card is used to play back .VAGs as MIDI instruments and sound effects on a computer. I used Opcode's Studio Vision Pro and Open Music System for the sequencing. The Sound Artist card is compatible with Open Music System, which makes the computer believe that the Sound Artist card is really a MIDI instrument. Once your .VAGs are loaded into the Sound Artist, you can start sequencing — or at least that's what I thought. I could trigger the sounds in the Sound Artist card using my Kurzweil K2000, but there was a considerable lag between triggering the sounds and their playback. After many hours of pain and anguish trying to solve this problem, I finally realized that my version of Sound Delicatessen was outdated. A simple upgrade fixed the problem. Lesson learned: don't forget to double check that you're using the latest version of your tools, lest you run into problems and/or forego the benefits of new features.



Sound Forge 4.od, Sonic Foundry's waveform editing program.

More Audio

All in all, NBA SHOOTOUT 98 was a fun project. Our audio team has formed great relationships with other development team members, and since the release of NBA SHOOTOUT 98, we've added a couple of audio people (whom we certainly could have used during the development of NBA SHOOTOUT 98) and we're in the process of adding some more. Upgrading existing titles from previous releases and creating new cutting-edge titles is an ongoing job here at 989 Studios, and we're continuing to make great games and to push technology. ■

FOR FURTHER INFO

AIF Typer
http://www.sonicfoundry.com/products/utilities.html

SetItsType 1.3
http://home.highway.ne.jp/masa-u/q5/hqx/setitstype-13-r2.hqx

ResEdit 2.1.3
ftp://ftp.info.apple.com/

Studio West
http://www.studiowest.com/
(619) 592-9497

The Surround Sound Mailing List
This is a good source for information on binaural audio. Subscribe to sursound@darkwing.uoregon.edu.

KBIG 104.3 in Los Angeles
Voice talent Mike Carlucci's new home as weekend DJ. Carlucci also can be heard at the Anaheim Mighty Ducks' hockey games.

SOAPBOX

by Warren Spector

It's ROLE-Playing, Stupid!

The oddest thing about computer role-playing games today is that you never hear anyone talk about the importance of playing a role. You hear about "400 character classes!" "6,753 unique skills!" "827 errand boy missions!" and "A world so big you won't want to explore it all!" Give it a rest. This is shallow. It's silly. It betrays our geeky roots in paper gaming (a medium with only a dangerous, superficial relation to electronic gaming).

Role-playing isn't about statistics or exploring randomly generated worlds of crate-filled buildings. It isn't about random quests and combat encounters every sixteen steps. It isn't even about +37 Swords of Instant Critical Hits that do Double Damage From Behind! Role-playing is about giving players the freedom to act as they see fit, within the framework of a story we provide.

Role-playing is about characters developing in unique and meaningful ways as a result of player choices. It's about trying new behaviors in a safe setting before we try them in the real world. In the space I have here, I can't tell you how we make a game that allows us to do all that. But let's start by identifying problems, and by looking at character, setting, and story, and how we usually approach them.

Character

Most RPGs define characters by an arbitrary "class" and/or a tiresome list of statistics. Characters typically have 6-12 attributes (strength, intelligence, and so on) and dozens of skills tracked at a fine level of granularity (lockpick score of 12, sharpshooter 72, computer hacker 53). Secret die rolls determine success or failure in skill use. The problem with this is that two players can do exactly the same thing and get different results because of insignificant differences between their characters. The difference between a 72 and a 73 shouldn't have any impact on game play. Does anyone think this is fun?

We have to come up with game systems that tell players what their characters are capable of doing and why they succeeded (or failed). In a computer game, we don't need 42 skills tied to percentile die rolls to simulate skill use. We're clever. We can come up with something better. Leave the dice and character sheets to paper gamers.

RPGs often use characters' abilities to bake bread, charm NPCs, and so on. Yawn. Some think hack-and-slash is a more riveting way to use characters' attributes. Ah, combat! It's relatively easy to simulate and it gets adrenaline pumping. That's not enough. Here's a radical concept: let players control when and if combat happens. Our goal should be to make combat an option, but not always the best, and *never* the only one. Encourage noncombat interactions, especially conversation.

We can't compromise conversation — a terrific tool for differentiating characters — and still call a game an RPG. Here are some ideas for improving conversations and game play:

- Conversations should reflect game state. Nothing's goofier than NPCs who keep talking while orcs hack them to bits.

Continued on page 63.

Warren Spector runs ION Storm's Austin, Texas, office. He is currently working on a new role-playing game, DEUS EX. In the past, he has produced games for Origin and LookingGlass Technologies. You can reach him at wspector@ionstorm.com.

Continued from page 64.

- Conversations should *not* involve lists of keywords. They're not fun, nor are they revealing of character. They're filler. They reduce conversations to the status of another stupid puzzle.
- Conversations should reveal things about NPCs; your responses should reveal things about you. The best way to accomplish this is to make "Yes/No" options the rule in conversational interaction with NPCs. Take, for example, a situation in which you and a friendly NPC face several enemies. The friend says, "I'll hold them off while you escape and Do Important Things…" Leave, and your friend is doomed. Stay, and your mission may come to an end. A Yes/No decision becomes a dramatic moment that reveals something about your friend and about you. That's very compelling game play.

Conversations are made interesting by the things they reveal about the characters speaking, the game world, and the world's state — not the number of branches in a conversation tree.

Setting

I've worked on games in which it takes hours to walk from one side of town to the other. Many popular, award-winning RPGs boast of hundreds of generic towns and randomly generated quests. The shallow simulation of huge environments isn't a good thing. Providing dialogue for scads of NPCs means none of them has anything interesting to say. Creating an entire country means any single building will be devoid of useful objects. It's a matter of time and storage space, and no amount of whack-on-the-side-of-the-head thinking allows you to finesse your way around the problems. Limit the size of your world. Provide several smaller maps. Increase the density of interaction. This accomplishes several goals:

- Players can explore without searching for something exciting to do. Aimless wandering is the enemy of fun.
- Developers can populate the world more densely with characters, objects, and quests, and give the illusion of a place with a life of its own.
- Action can be tailored to player skill. Difficulty can be increased easily as players get deeper into the game.
- Developers can create more varied locations than in a sprawling world.

This last point is critical, and most RPGs do this well. However, most RPGs feature wacky environments straight out of designers' fevered imaginations. It's not asking too much to think in terms of believable, recognizable locations instead of arbitrary game spaces. We should try to acknowledge the conventions of the everyday, even when we create fantasy worlds. In the real world, you can tell you're in a bedroom, as opposed to a bathroom, the instant you enter because of size, placement, and furnishings. More game designers should realize this.

Some games do hint at the possibilities of believable environments, but they don't go far enough. In DUKE NUKEM (a game I loved), the environment was a gimmick. You knew you were in a movie theater, and you could switch the projector on and watch a bikini'ed babe do her thing — let's talk about sexism another time — but you couldn't switch on that projector and blind a sniper before he fired. Imagine if shooting a fire hydrant allowed you to douse a fire. The ULTIMA games go further, but not always in significant ways (mea culpa!) — the key is not that every plate and knife and fork be usable, or that players reap wheat, grind it into flour, and bake it into bread. The key is recreating realistic locations and object interactions that are *exciting*. Give players believable worlds with lots of usable objects that produce predictable, useful results. Let them blast barricades, freeze enemies and then shatter them. Create worlds where water damages paper and gratings creak beneath players' feet.

Every game problem should have multiple solutions, by design or because alternatives arise naturally out of the simulation. How players deal with the problems they encounter (whether they choose violence over cleverness, talk first and shoot later, and so on) should affect subsequent interactions with the denizens of the game world as well as the substance of later missions.

Story

Is it just me, or does it seem like every RPG drops players into a huge, all-but-empty world and says, "Go. Hope you find some fun."? Man, have I been guilty of that. After stumbling around for a couple hours, players may even find a clue that they're supposed to Kill the Evil Foozle. It's almost as if there's some unspoken rule against offering RPGers clear goals. The trick shouldn't be figuring out *what* you're supposed to do (which isn't much fun); the trick should be figuring out *how* to accomplish what you *know* you have to accomplish. New goals can be revealed as you go, but damn it, reveal those goals! And make those goals more compelling than "kill everything you see," okay? If working with Richard Garriott taught me anything (and, believe me, it did) it's that an RPG can be more than just a slugfest. More than any other medium of expression, gaming lets people find their own answers to tough questions, rather than imposing an artist's vision of the world on them. It doesn't matter what issues we explore — tolerance, morality, relationships, whatever — but let's explore *something*.

Dungeon crawls are all well and good, but we can allow players to explore who they are and what they actually believe. Unlike authors and filmmakers, we can give people the opportunity to test behaviors they'd never try in the real world. I feel we have an obligation to do that. If we provide only one answer (usually violent) we do our medium and our players a disservice.

Allow players to make choices and then show the ramifications of those choices: kill everything you see and suffer the consequences; play the pacifist and pay a different price. Games should be rife with ethical dilemmas rather than right and wrong choices. "What are you fighting for?" and "How do you achieve your goals?" should be unavoidable questions. When all is said and done, story goals and tough questions are just tools used to suck in players. If we create small, deep, object-rich simulations that allow multiple solutions to tough problems, players will inevitably stumble upon the "real" goal of an RPG — to grow a unique alter ego.

Doing everything I've outlined above won't assure you a hit and accolades from peers, press, and players. All I know is we have to try. We have to fail gloriously. If we keep settling for RPGs that could have been made five or ten years ago — and that describes every RPG released in the last couple of years — we're doomed. ■