gd

GAME DEVELOPER MAGAZINE

SEPTEMBER 1999

UBM Tech

Fast, Cheap, and Out of Control

This past July I was fortunate to fly out to Monte Carlo for Medpi, a gathering of game publishers and representatives from the largest French game distributors. In a sense it's like E3, except that the entire Medpi exhibition area could have fit within Nintendo's E3 booth. During my second day at the show, I met a developer on the show floor who works for a major game development/publishing company. As we toured his company's booth and he demonstrated their upcoming titles for me, we began discussing movies — *The Phantom Menace*, specifically. This guy told me that the official release date for *TPM* wasn't until October 13, but confessed he'd already seen the movie — he had downloaded it from the Internet. I was taken aback.

Ethical issues aside, the thought of downloading a few hundred individual movie segments from alt.binaries.multimedia, piecing them together, and then watching the results on my 15-inch computer monitor just isn't appealing to me. Sadly, I have to admit it's not the risk/reward ratio that deters me, it's the work/reward ratio. The consequences of downloading an illegal movie, game, or song from the Internet are virtually nonexistent. While I'm pretty sure that the effort required to do so forces many people simply to use the legal channels, it's still way too easy these days to download digital content illegally. The sheer volume of pirated media available leaves little doubt that enforcing intellectual property laws is going to be one of the biggest law enforcement challenges in the coming century. It will exert downward pressure on publisher revenues and developer royalties, and keep game (and probably movie and CD) prices higher than necessary.

And then there's the flip side to the coin. What happens when other entertainment media embrace the Internet as a (or the) primary means of delivering their content? Music, of course, is well down this path already, thanks to formats such as MP3 and RealAudio, and support hardware like the Diamond Rio. If major motion picture studios, television networks, radio stations, and record labels decide that the time's right to push their content onto the Internet (I'm talking in a major way — not the half-hearted attempts we're seeing today), what will that do to game sales? Will the competition from other forms of digital entertainment mean opportunities for game developers, or will it threaten the preeminence of games as computer-based digital entertainment?

While the current retail model for games is far from dead and competition from other forms of digital entertainment is still nascent, we all need to be prepared for a completely wired, digitally integrated world. The Internet will have a major impact on game development and distribution next century, and I'm convinced that as the Internet matures as a delivery mechanism, it will continue to benefit the game industry.

To what extent we'll feel the pinch of piracy and competition from other forms of media as the Internet evolves is anyone's guess. Piracy enforcement itself may be too tough to overcome. It may require gigantic changes in the current business model of game publishing, or an advanced licensing scheme based on advanced encryption technology not yet invented. Much of tomorrow's popular software (such as Linux) will be a free commodity which drives ancillary money-making businesses, like training or consulting. In gaming, the model might be to distribute free games and charge a fee to participate in professional leagues or against other players online.

Like the title of Errol Morris's wacky 1997 documentary, the Internet of the next century is going to be "fast, cheap, and out of control." In that climate, governments must be prepared to step up enforcement of intellectual property laws, and games companies must step up to compete against (or cooperate with) other forms of media. ■

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

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

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

EDITORIAL

Editorial Director
Alex Dunne *adunne@sirius.com*

Managing Editor
Kimberley Van Hooser *kvanhoos@sirius.com*

Departments Editor
Jennifer Olsen *jolsen@sirius.com*

Art Director
Laura Pool *lpool@mfi.com*

Editor-At-Large
Chris Hecker *checker@d6.com*

Contributing Editors
Jeff Lander *jeffl@darwin3d.com*
Mel Guymon *mel@surreal.com*
Omid Rahmat *omid@compuserve.com*

Advisory Board
Hal Barwood LucasArts
Noah Falstein The Inspiracy
Brian Hook id Software
Susan Lee-Merrow Lucas Learning
Mark Miller Harmonix
Paul Steed id Software
Dan Teven Teven Consulting
Rob Wyatt DreamWorks Interactive

ADVERTISING SALES

Western Regional Sales Manager
Jennifer Orvik e: *jorvik@mfi.com* t: 415.905.2156

Eastern Regional Sales Manager/Recruitment
Ayrien Houchin e: *ahouchin@mfi.com* t: 415.905.2788

International Sales Representative
Breakout Marketing e: *breakout_mktg@compuserve.com*
t: +49 431 801703 f:+49 431 801797

ADVERTISING PRODUCTION

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

MILLER FREEMAN GAME GROUP MARKETING

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

CIRCULATION

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

INTERNATIONAL LICENSING INFORMATION

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

Miller Freeman
A United News & Media publication

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

BPA INTERNATIONAL

www.gdmag.com

BIT Blasts

News from the World of Game Development

New Products: Raindrop Geomagic simplifies NURBS, Digimation dishes out Particle Studio, and Digital Origin debuts RotoDV. **p. 9**

Industry Watch: Nintendo's Dolphin strategy, Activision's big game hunt, and the latest from EA Sports. **p. 10**

Product Review: Dan Teven takes Virtools' NeMo for a test drive in rapid game development. **p. 12**

New Products

by Jennifer Olsen

Whip 3D Models into Shape

RAINDROP GEOMAGIC has introduced Shape, a new 3D modeling tool designed to relieve artists and designers of the ponderous methods associated with creating high-quality NURBS models. Not all designers are great mathematicians, after all, and vice versa.

With Shape, users can start with data from a physical object scanned and modeled in Geomagic Wrap, or import polygonal data from various 3D file formats. Shape then generates NURBS patches based on an automatically computed patch layout, lays a grid structure on each patch, and fits a NURBS patch to each grid. Adjacent NURBS patches are guaranteed to be connected seamlessly, unless the user specifies otherwise. Shape's tolerance function can then evaluate the approximation of the NURBS surface by comparing it to the original polygonal data, and map a color-coded tolerance computation directly onto the NURBS surface.

Shape runs on Windows 95/98/NT and IRIX. It's available as a stand-alone or as a component of the new Geomagic Studio, which also includes the polygon-reduction tool Decimator and a new version of Geomagic Wrap.

- **Raindrop Geomagic Inc.**
  Research Triangle Park, N.C.
  (800) 251-5551 or (919) 474-0122
  http://www.geomagic.com

Party with Particles

DIGIMATION has released Particle Studio, its new particle generation plug-in for 3D Studio Max and the successor to Sand Blaster.

It's difficult to imagine a game today that wouldn't incorporate some kind of particle system in its architecture. Particle systems are useful for handling a variety of common effects such as smoke, fireworks, fountains, stars, and did I mention explosions?

Particle Studio works with an event-driven paradigm, breaking the particle system up into time-based events. Most particle systems work under a set of parameters created at the beginning of the system. In order to control the system over time, the user must either alter the original parameters, or use other tools such as space warps. With Particle Studio, users can define a new event with its own set of parameters at any point along the life of the particle system. Users can activate or deactivate space warps in a similar fashion when each new event is created.

Particle Studio comes with three plug-ins: Particle Studio, Particle Studio helper, and the Particle Studio Snapshot utility. It is priced at $595, with discounts available to current Sand Blaster users.

- **Digimation Inc.**
  St. Rose, La.
  (504) 468-7898
  http://www.digimation.com

DV Effects in Real Time

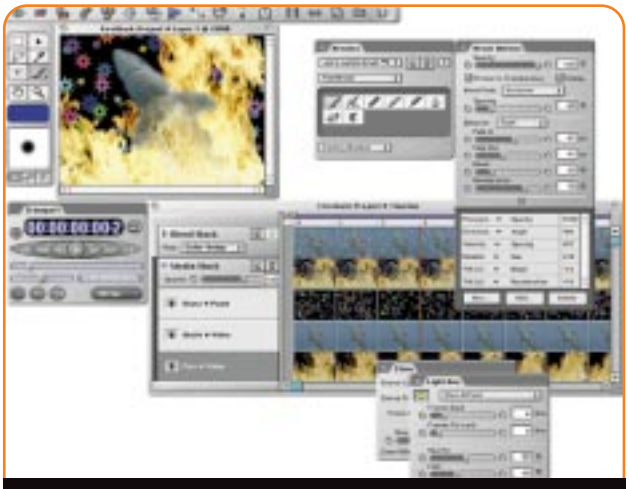
DIGITAL ORIGIN has begun shipping RotoDV, its new Macintosh-based digital video manipulation tool for rotoscoping, special effects, animation, and touch-ups. RotoDV offers real-time playback at full resolution, depending on RAM availability, enabling users to view their work quickly as they go.

With native QuickTime architecture, RotoDV promises to make fast friends with many other non-linear video editing tools you and your team might be using, including Adobe Premiere and Digital Origin's EditDV, and it plays well with compositing applications such as Adobe After Effects.

Users can customize RotoDV's brushes by setting and linking different parameters, enabling a wide variety of effects. Replicator brushes can transfer certain images or entire frame sequences from frame to frame or layer to layer, even during playback. The Media Stack offers unlimited paint and video layers for endless combinations, and the layers are non-destructive — handy for anyone who has ever accidentally marred a treasured video clip in a fit of creativity.

Available for Mac OS 8.5 or higher, RotoDV has a suggested retail price of $699, with special introductory pricing available on a limited basis.

- **Digital Origin Inc.**
  Mountain View, Calif.
  (650) 404-6000
  http://www.digitalorigin.com



RotoDV's Media Stack features unlimited layers, enabling users to implement just about any wacky idea.

Industry Watch

by Dan Huebner

CHANGES AT LUCASARTS. The summer of *Star Wars* may have tempted some Lucas employees to follow Anakin Skywalker's lead by leaving the nest. A series of recent departures from LucasArts Entertainment, including ROGUE SQUADRON project leader Mark Haigh-Hutchinson, GRIM FANDANGO lead programmer Brett Mogulefsky, along with several others in various departments, culminated in the resignation of eight-year Lucas veteran and director of production Steve Dauterman. Though Dauterman's parting was amicable and there doesn't seem to be a connection between the departures, rumors around the ranch suggest that other executive exit visas may be pending.

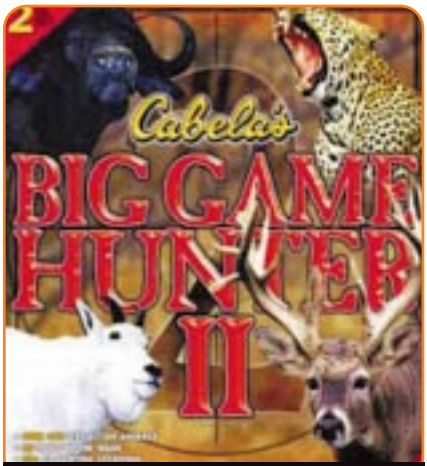
CHEAP FISH. Nintendo seems to be planning to follow the example of its own Game Boy by bringing its next-generation console to market at the lowest price possible. Though competitors Sony and Sega are offering pricey thrills such as 56K modems and DVD movie playback, Nintendo plans on creating a stripped-down version of its upcoming Dolphin that some analysts believe could be priced as low as $99. This low-cost Dolphin would not be able to play CDs or DVD movies, nor would it be likely to include a modem. Nintendo's partner, Matsushita, will produce a fully loaded model with DVD and CD functionality. Though $99 may be a bit unrealistic, anything approaching that would still put Dolphin well below the $340–$440 Playstation 2 price that was floating around the Tokyo Game Show.

ULTIMA ASCENDANT. Origin Systems is consolidating its efforts into massive multiplayer online worlds. Following on the success of ULTIMA ONLINE, Origin is canceling production of JANE'S A10 WARTHOG at its Austin, Tex. studio. Those working on the fighter jet title are to be relocated to other projects within the company. WARBIRD flight fans need not worry, Origin parent Electronic Arts plans to continue Jane's Simulations series at another studio.

ACTIVISION BAGS BIG GAME HUNTER. Demand for hunting games has remained strong, and Activision has renewed its commitment to this unexplainable market segment by acquiring Florida-based Elsinore Multimedia. Despite a name that conjures up images of a brooding Hamlet, Elsinore is the creator of Wal-Mart favorites CABELA'S BIG GAME HUNTER I and II. Elsinore had previously been published by Activision label Head Games, and now joins as a wholly owned subsidiary.

EA GOES HANDHELD. Slackers who spend weeks at a time tied to EA Sports' football and hockey offerings may finally have a reason to get off the couch. Electronic Arts has entered into a deal with Radica Games, best known for developing shaking electronic bass fishing games, to take EA's sporting line mobile. Radica will bring these franchises to market as EA Sports brand handheld electronic games, which will more than likely beep and vibrate their way into the hearts of Madden fans everywhere. Radica will also hold the right of first refusal over any game in the EA line.

FURNITURE-FRIENDLY DEATHMATCH. Deathmatches are about to become a whole lot safer when Hasbro and Visionary Media bring us NERF ARENA BLAST. Though it may seem a bit incongruous at first glance, Nerf and 3D first-person shooters are actually very similar: both allow players to kill each other violently with exotic looking weaponry without scuffing the furniture. Put the two together and you have mayhem the whole family can enjoy. The game was developed using the UNREAL engine. ■



Activision is making sure they stay where the action is.

Radica's handheld line will soon feature EA Sports titles.

UPCOMING EVENTS CALENDAR

ECTS '99

OLYMPIA CONVENTION CENTRE
London, England
September 5–7, 1999
Cost: variable
http://www.ects.com

Game Developers Conference 1999 Road Trips

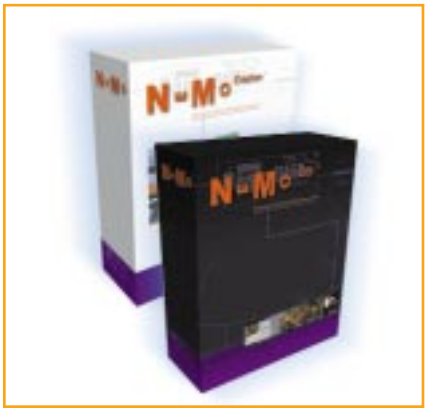
BATTERYMARCH CONFERENCE CENTER
Boston, Mass.
September 8, 1999

NAVY PIER CONFERENCE CENTER
Chicago, Ill.
September 10, 1999

RALEIGH CONVENTION AND CONFERENCE CENTER
Raleigh, N.C.
September 13, 1999

SEQUOIA CONFERENCE CENTER
Buena Park, Calif.
September 27, 1999

Cost: $120 ea. (discounts available)
http://roadtrips.gdconf.com



Virtools' NeMo

by Dan Teven

Virtools wisely avoids the phrase "Rapid Application Development" when it describes NeMo. So-called RAD tools make me think of the control I'm giving up, not the speed I'm gaining. So I was pretty amazed when I realized that NeMo is a Rapid Game Development tool — and I like it.

NeMo comes in two flavors, NeMo Creation and NeMo Dev. Both let you create a 3D world populated with dynamic objects and assign behaviors to those objects. Using Creation, you can develop simple games that run in a special player or "gamelets" that run in a browser. With Dev, which adds an SDK for C++, you can integrate NeMo into your own application.

To its credit, Virtools doesn't push you to buy Dev right off the bat. They believe most people need to create 3D prototypes, demos, or multimedia presentations. If you need the extra flexibility of Dev, you can upgrade easily.

CONCEPTS. At its heart, NeMo is a library that implements the concept of a behavior, coupled with a scene management database, a simulation engine, and a library of more than 200 stock behaviors. It's not a rendering engine, but it can use Direct3D or OpenGL. You use the NeMo environment built on top of these components to engineer your content; you're only exposed to the low-level interfaces through the SDK.

NeMo organizes data into levels, scenes, places, and groups. These contain 2D and 3D entities: 3D objects (made of meshes, materials, textures, and sounds) plus sprites, curves, cameras, and lights. Characters are just collections of 3D entities. Although some behaviors are character-specific, most are generalized, so you can build a 2D sprite game if that's your goal.

You'll need to create most of your data with third-party tools. Although NeMo can handle the .X file format, it seems better suited to .3DS — many of the standard behaviors seem to map directly to 3D Studio Max modifiers. You can import the usual 2D and audio formats. Afterwards, you can do all the integration, behavioral scripting, debugging, and tuning work in the NeMo environment.

A behavior is a program associated with an object, allowing the object to change during the simulation. In NeMo, anything can have a behavior. An obvious character behavior is "wait for a mouse click, then walk to the clicked-on object." But behaviors can also be used to make tracking cameras, procedural textures, levels that keep score, and just about anything else.

THE ENVIRONMENT. NeMo's user interface is a mixed bag. Virtools deliberately strove for a tool that looks different from other Windows applications, and the result has loads of style. I love the overall look, the way you can accomplish almost anything with drag and drop, and the way you can play your content without having to wait for compilation.

I really love the filter graph (flowchart) metaphor for creating behaviors. You don't need to write a line of code — just drag behaviors from the Building Blocks pane into the Schematic, click to connect the appropriate pins, set parameters through a dialog box, and you're ready to roll.

You can really get work done in this environment. The Trace mode (showing the active behaviors in real time by highlighting them in red) is cool, and there's breakpoint and single-step support. There's also an integrated profiler.

Although it's attractive, the UI is maddeningly inconsistent. Only a few of the commands are available from the menu bar, so you have to learn a lot of special-case controls. Sometimes right-clicking works, sometimes it doesn't. Many controls are close at hand on the window borders, but few are associated with tool tips. The help is not context-sensitive. And it's risky to click blindly, because there's no undo.

I'm not crazy about the windowing. On-screen, there are three tiled, non-resizable window frames, and each can hold multiple tabbed panes. You zoom windows to full screen by double-clicking on the tab, not the top border. You can drag panes from one frame to another to make the best use of space, but it's too easy to inadvertently open a new pane, which causes the one you were working with to vanish (actually, the tab scrolls out of view). And, there's no consistent way to close windows.

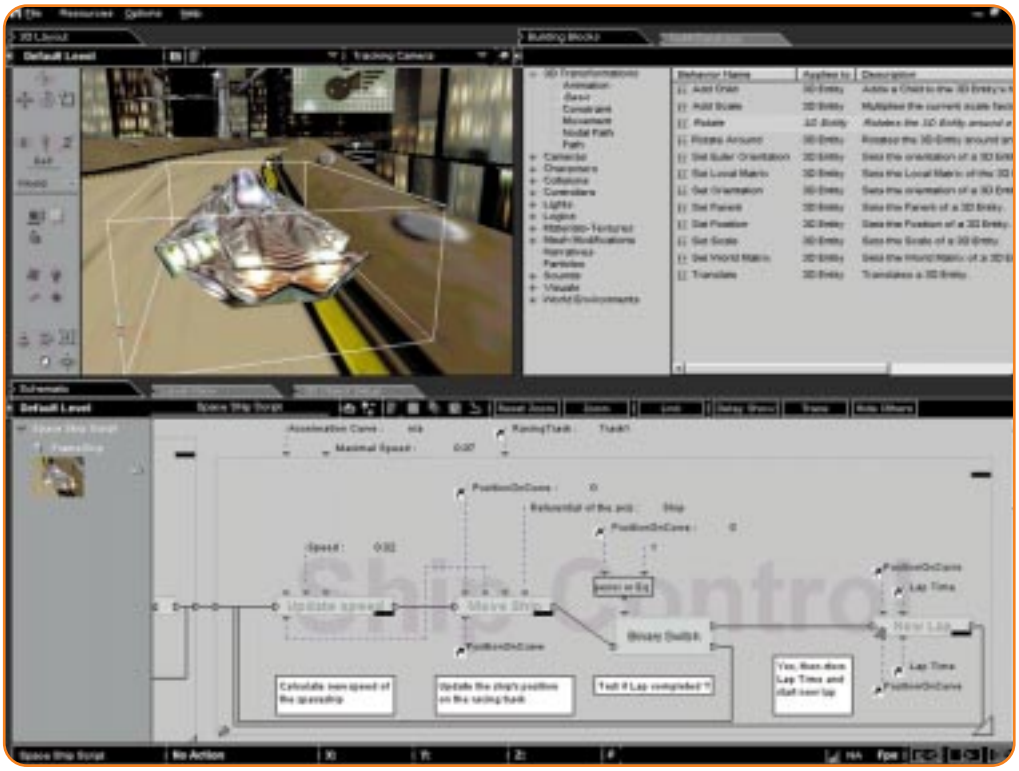
Tree controls let you explore the data and behavior hierarchies. These are adequate, but limiting. It would be nice if certain behaviors appeared more than once under Building Blocks. For example, Character Prevent from Collision should be listed under both Collision and Character.

I found NeMo to be very keyboard-unfriendly. For some reason, Alt-F doesn't bring down the File menu, even though the F is underlined. The behavior that steers a character around in the tutorial works with the keypad arrows, but not the special-purpose arrow keys. And, if you play your scene in full screen mode, the only way to get back is by Alt-Enter. (I tried mouse clicks, Esc, Enter, spacebar, all the function keys, Alt-Tab, and Ctrl-Alt-Del before I figured this out.)

Even the color scheme — blacks and grays — is problematic. By shunning color in the interface, Virtools makes the content look better. But this makes it harder to tell when a control contains editable text, or even which window has the focus.

BEHAVIORS. Creating reusable behaviors with NeMo couldn't be easier. You can collapse a filter graph into a black box that behaves just like an atomic behavior. You can name your compound behavior and annotate it with comments and screen snapshots. When

Dan Teven specializes in systems architecture for game development. He's always wondered why a solitaire game comes with Windows. Sue him for pointing this out at dteven@ici.net.



you use it, NeMo will create the "Edit Parameters" dialog box on the fly.

With Dev, you can create new behaviors in Visual C++. These will show up in Building Blocks and the help system, and they'll work normally in the schematic, except you won't be able to expand the black box. All the stock behaviors were built with the SDK, and their source is included.

Some of the stock behaviors go far beyond what I expected. There are lots of mesh modifiers, including a morpher and a skin-join, and there are eight kinds of particle systems. There are some interesting camera effects, such as vertigo. There's an interpolator that works in HSV color space and one that works along a 2D Bézier curve. Many rendering effects, such as motion blur, are also available. The leverage you get with NeMo, right out of the box, is really impressive.

DOCUMENTATION. Except for the lack of an undo function, the UI problems are superficial. If you're looking for an Achilles heel in NeMo, look to the documentation.

Creation has a paper manual that will get you up and running, but it runs out of steam soon after that. Parts of the Advanced Topics section read like a programmer's notes on a napkin. There's also an online reference; the two together are barely adequate. Dev has plenty of sample code and another online reference, labeled "under construction," and it assuredly is.

What's missing? High-level information about the architecture, so you can understand how NeMo is put together, and how you can extend it through the SDK; more step-by-step, task-oriented tutorials; the technical details of behaviors and parameter operations; documentation of the debugging tools; documentation of preferences; a full index; and searchability. Dev would benefit from templates for behaviors, or even a wizard that generates them for you.

Since Virtools is a French company, there are occasional bad translations. The meaning is usually obvious — as in "2ème param" — but in a few places the awkward translations affect "lisibility." The terminology can be strange: I might have chosen the terms "local" and "global" instead of "place" and "trans-place."

At least there's an active user group. On Virtools' web site, you can ask questions, share techniques, and show off your work. Since NeMo will introduce a lot of newcomers to real-time 3D, Virtools hopes the user group can provide the background culture required to master concepts, such as efficient collision detection and optimized rendering.

PROTOTYPE OR PRODUCTION? Currently, Dev lets you create behaviors, integrate the engine into your application, plug in your own renderer, and create import plug-ins. The architecture looks modular, and in theory you can interface NeMo to just about any external engine. Discreet is currently using NeMo as a test bed for releasing core 3D Studio Max R3 character animation algorithms as game engine components.

NeMo Dev has tremendous potential, but until the documentation's finished I can't recommend it for production. NeMo Creation has some 1.0 flaws, but it's Rapid Game Development at its best. ■

NeMo Creation: ★★★★ **NeMo Dev:** ★★½

Virtools S.A.
Paris, France
+33 (1) 42-71-46-86
http://www.nemo.com

Price: Creation is $990 per seat. Dev is $3,490 plus a negotiable license fee.

System Requirements: Pentium 200, 48MB RAM, Direct3D or OpenGL accelerator, 60MB disk space, Windows 95/98/NT 4.0, DirectX, Internet Explorer 4.

Pros:

1. Flowchart metaphor lets you create behaviors rapidly without coding.
2. Behaviors can be smoothly reused, combined, and extended.
3. The impressive selection of canned behaviors makes it a great prototyping tool.

Cons:

1. The documentation ranges from spartan (Creation) to woeful (Dev).
2. Inconsistent user interface.
3. Dev isn't mature enough to rely on for production — yet.

Physics on the Back of a Cocktail Napkin

I am generally pretty disciplined about working normal hours during the week. However, on Fridays I like to shoot pool and eat pizza at the local sports bar. Since happy hour starts at three, I sometimes need to move work down the street. I was working out how to win a serious game of nine ball when one of those geeky discussions broke out about pool table physics. Pool, like many sports, is dominated by the laws of physics. Good players have an excellent sense of the application of force, the physics of collisions, and the influence of friction on objects in motion.

Last month I described how friction could be used to increase the realism of the physics model in real-time games. The demo program made it possible to see how various coefficients affected a mass-and-spring model. However, it wasn't very much fun. In order to demonstrate how a solid physical foundation can actually create interesting game play, I need to pull some of these concepts together into a real application. A pool table simulation is a natural choice. It will allow me to apply many of the techniques I have covered as well as provide some ideas that can be converted easily to other sports such as golf or tennis.

Two Ball, Corner Pocket

In order to understand pool, I need to understand collisions between billiard balls. Fortunately, billiard balls are all spheres of equal size and weight. That makes the collision calculations a bit easier. Let me begin by looking at the frictionless case. I suspect that if I ignore the ball's rotation and do not consider friction, the ball collision will behave exactly like a particle at the ball's center of mass. That would be great, as I could use the code from a previous column. However, I want to make sure. Figure 1 shows a typical collision.

Ball A is moving at a speed of 20 feet per second and collides with ball B at a 40 degree angle. In order to determine the velocity of each ball after the collision, I need to apply dynamics. A collision between two

Like most things in life, pool provides an excellent venue for a physics lesson.

v	Velocity of body (vector)
m	Mass of a body
ε	Coefficient of restitution
n	Line of collision or collision normal
t	Line tangent to collision
j	Impulse force
N	Force normal to surface
g	Gravitational force
μ_S	Coefficient of sliding friction
μ_R	Coefficient of rolling friction
R	Radius of the ball
I	Inertia tensor
ω	Angular velocity
α	Angular acceleration
r	Vector from center of mass to point of contact

TABLE 1. *A summary of the notation used in this article.*

When not wasting time at the pub eating hot wings and shooting pool, Jeff can be found at Darwin 3D. There he creates real-time 3D graphics for a variety of applications. Drop him a line at jeffl@darwin3d.com.

rigid bodies, which occurs in a very short time and during which the two bodies exert relatively large forces on each other, is called an impact. The force between these two bodies during the collision is called an impulsive force, which is symbolized by *j*. The common normal to the surfaces of the bodies in contact is called the line of collision, represented as *n* in Figure 1.

The first step is to break the initial velocity of ball A into its components along the line of collision, *n*, and the tangent to the collision, *t*.

$$v_A = 20 \text{ ft / sec}$$
$$(v_A \cdot n) = v_A \cos(40) = 15.32 \text{ ft / sec}$$
$$(v_A \cdot t) = v_A \sin(40) = -12.86 \text{ ft / sec}$$

The impulsive force acting during the collision is directed along the line of collision. Therefore, the *t* component of the velocity of each ball is not changed.

$$(v'_A \cdot t) = -12.86 \text{ ft / sec}$$
$$(v'_B \cdot t) = 0 \text{ ft / sec}$$

In order to determine the new velocity along the line of collision, I need to look at the impulsive force between the bodies. The impulse acts on both bodies at the same time. You may remember Newton's third law of motion, the forces exerted by two particles on each other are equal in magnitude and opposite in direction.

Since the impulse forces are equal and opposite, momentum is therefore conserved before and after the collision. Remember that the momentum of a rigid body is mass times velocity (*mv*).

$$m_A(v'_A \cdot n) + m_B(v'_B \cdot n) = m_A(v'_A \cdot n) + m_B(v'_B \cdot n)$$
$$m(15.32) + m(0) = m_A(v'_A \cdot n) + m_B(v'_B \cdot n)$$
$$(v'_A \cdot n) + (v'_B \cdot n) = 15.32 \qquad \text{(Eq. 1)}$$

This equation can't be solved without some more information. In my column "Collision Response: Bouncy, Trouncy, Fun"(Graphic Content, March 1999), I discussed the coefficient of restitution. This is the scalar value between 0 and 1 relating the velocities of bodies before and after a collision via the formula:

$$(v'_B \cdot n) - (v'_A \cdot n) = \varepsilon[(v_A \cdot n) - (v_B \cdot n)] \qquad \text{(Eq. 2)}$$

For this example, I'm using a coefficient of restitution ε of 0.8. I can use this formula to create a second equation.

$$(v'_B \cdot n) - (v'_A \cdot n) = \varepsilon[(v_A \cdot n) - (v_B \cdot n)]$$
$$(v'_B \cdot n) - (v'_A \cdot n) = 0.8[(15.32) - (0)]$$
$$(v'_B \cdot n) - (v'_A \cdot n) = 12.26 \qquad \text{(Eq. 3)}$$

Solving Equations 1 and 3, I get the velocities of the two billiard balls after the collision.

$$v'_A = (1.53, -12.86) \text{ ft / sec}$$
$$v'_B = (13.79, 0.0) \text{ ft / sec}$$

In order to solve this problem in the simulation, I need to derive the impulse force directly. The impulse force creates a change in momentum of the two bodies with the following relationship.

$$m_A v_A + jn = m_A v'_A$$
$$v'_A = v_A + \frac{j}{m_A} n$$
$$m_B v_B - jn = m_B v'_B$$
$$v'_B = v_B - \frac{j}{m_B} n \qquad \text{(Eq. 4)}$$

These formulas can be combined with Equation 2 to determine the impulse force given the relative velocity and the coefficient of restitution.

$$j = \frac{-(1+\varepsilon)v_{AB} \cdot n}{n \cdot n \left(\frac{1}{m_A} + \frac{1}{m_B}\right)} \qquad \text{(Eq. 5)}$$

You can plug Equation 5 back into the example problem and make sure it works. Remember that because of Newton's third law, the impulse is equal and opposite for the two colliding bodies. When you apply Equation 5 to the B ball, remember to negate it.

Those of you who read Chris Hecker's column on collision response ("Physics, Part 3: Collision Response," Behind the Screen, February/March 1997) will recognize Equation 4 as the impulse equation for a general body that does not rotate. When we do not consider the rotation of the billiard balls, they behave exactly like the particles used in my March 1999 mass-and-spring demo. My suspicion was correct, and I can use the particle dynamics system as a base for the demo.

For many applications, this would probably be more than enough to get a decent physical simulation. In fact, I imagine many pool simulations end right there. This level of simulation is probably sufficient for other games, such as pinball. However, anyone who has played much pool knows that this is not the end of the story. The rotation of the ball caused by the reaction with the table makes a tremendous difference in the realism of the simulation.

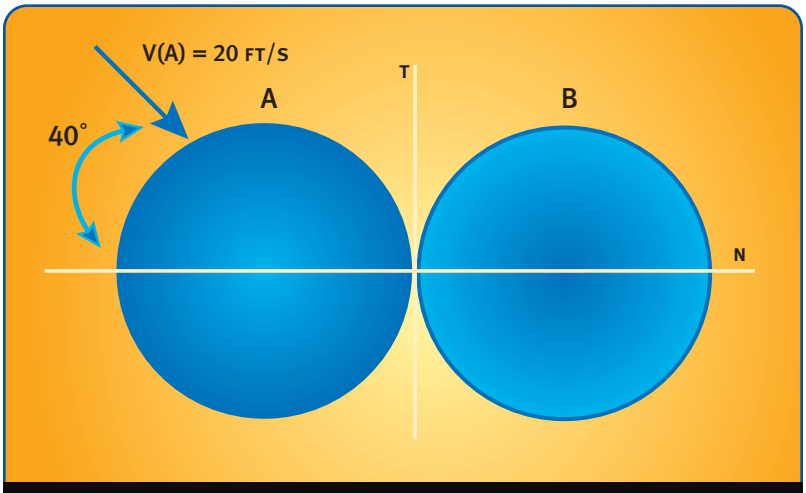


FIGURE 1. *A simple collision between two billiard balls.*

Slip Sliding Away

When a billiard ball is hit with the cue stick, the ball starts moving across the table. If the ball is struck along its center of mass, the ball is not initially rotating.

However, soon the ball starts rolling along. Friction between the ball and the table causes this roll to occur. You can see this situation in Figure 2.

The ball is traveling with the forward velocity, *v*. In last month's column ("The Trials and Tribulations of Tribology," Graphic Content, August 1999), I discussed the use of kinetic friction via the Coulomb dry friction model. For our purposes, I'm going to call this force "sliding friction." The force of friction applied to a body sliding over a surface is given by the following formula:

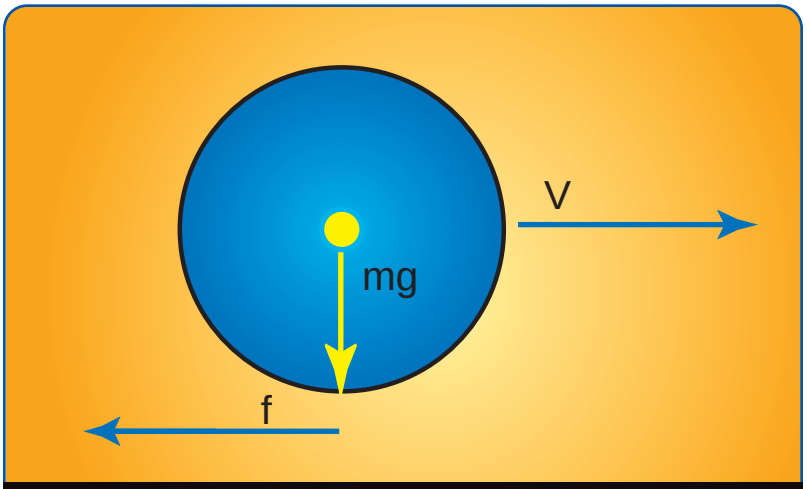


FIGURE 2. *A billiard ball sliding across the table.*

$$f = \mu_s N = \mu_s mg \qquad \text{(Eq. 6)}$$

The friction force is applied in the direction opposite the velocity. Since this force is applied to the surface of the ball and not its center of mass, the frictional force causes angular acceleration in the ball. As the ball rolls across the table, the angular velocity increases because of this sliding friction force. This continues until a time of equilibrium is reached, where the velocity of the point contacting the table equals the velocity of the center of mass. At this time, the ball is no longer sliding and is now rolling on the table. This situation is called a natural roll or rolling without sliding. In mathematical terms, this situation happens when

$$v = R\omega \qquad \text{(Eq. 7)}$$

where *v* is the velocity of the ball, ω is the angular velocity of the ball, and *R* is the ball's radius.

Now I need to show how the angular acceleration actually changes. This is going to mean bringing up another term, the inertia tensor, or *I*. You may remember from Chris Hecker's column on 3D physics ("Physics, Part 4: The Third Dimension," Behind the Screen, June 1997) that the inertia tensor relates the angular velocity of a body to the angular momentum of that body. For arbitrarily complex objects, creating the inertia tensor can be quite difficult. However, for a uniform sphere where the density is uniform across the sphere, it's quite easy. The inertia tensor for a sphere is

$$I = \frac{2mR^2}{5}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{(Eq. 8)}$$

Therefore, the product of this matrix with any vector is a simple scaling of that vector. The relationship between the angular acceleration and the friction force then becomes

$$\alpha = \frac{r \times f}{I}$$

$$\alpha = \frac{r \times f}{\frac{2}{5}mR^2} = \frac{5(r \times f)}{2mR^2} \qquad \text{(Eq. 9)}$$

If I now take a look at the problem in Figure 2, I can calculate how long it will take for the ball to achieve natural roll given an initial velocity *v*. From the principle of impulse and momentum, I know some information about the linear momentum and angular velocity of the ball at a later time.

$$mv' = mv - f(\Delta t)$$

$$\omega' = \frac{f(\Delta t)r}{I} = \frac{5f(\Delta t)}{2mr}$$

In other words, the momentum at some later time is the initial momentum minus the impulse created by the friction force, *f*. I know the friction force from Equation 6.

$$mv' = mv - \mu_s mg(\Delta t)$$

$$v' = v - \mu_s g(\Delta t)$$

$$\omega' = \frac{5\mu_s g(\Delta t)}{2r}$$

At the point of natural roll, I know the state of equilibrium between angular and linear velocity from Equation 7.

$$v' = r\omega'$$

$$r\left[\frac{5\mu_s g(\Delta t)}{2r}\right] = v - \mu_s g(\Delta t)$$

$$\left(\frac{5\mu_s g}{2} + \mu_s g\right)(\Delta t) = v$$

$$\Delta t = \frac{2v}{7\mu_s g}$$

So you can see that as a result of the friction force of the table, a sliding billiard ball will always reach a point where it is rolling without sliding on the table. This is the type of realism I want to have in the simulation. A ball when struck should slide across the table, slowly settling to a state where it is rolling without slipping.

How Do I Stop This Crazy Thing?

One glaring problem remains. I can run the simulation with all of the physics discussed so far. When struck hard, a billiard ball will slide and then roll. Once the ball has reached this natural roll, there is nothing in my simulation that will keep it from continuing to roll forever. The friction

force is gone since the point of contact is not moving relative to the table. I need to add another force that will slow down a rolling ball. I can add another frictional force, called rolling friction, which is applied when the ball is in natural roll. The form of rolling friction is

$$f_R = \mu_R mg$$

It is applied exactly like the sliding friction whenever the natural roll conditions apply. It is important to note that the coefficients of rolling and sliding friction are not necessarily the same. Think of a ball moving on a rubber surface. The coefficient of sliding friction would be very high. However, the rolling friction would be comparatively low, allowing the ball to roll across the surface easily.

Bumpin' the Cush

Collision with the table's side cushions can be handled in a couple of ways. If I consider the table to be completely 3D, I will need to handle 3D collision between the ball and the cushion. That would be the most realistic. It would allow the balls to move up and down as well as side to side. This might be interesting if I wanted to be able to perform a jump shot (when the cue ball jumps up and over other balls on the table). However, I'm not really ready to tackle the physical and interface issues involved in making this happen.

If I'm willing to give up the flexibility of allowing the balls to move in 3D, things become a bit easier. For one thing, I can eliminate the gravitational force acting on the ball. A ball sitting on a table is in a constant collision battle with the table top. By getting rid of the gravitational force, I save having to deal with the ball constantly interpenetrating the table. I still need to keep track of the gravitational force as it is used in calculating the friction force applied by the table. However, I just assume the balls are in constant sliding or rolling contact with the table.

Also, if I ignore vertical motion of the balls, I can turn the collision with the side cushions into a 2D computational geometry problem. The boundaries of the table are now line segments and I can use the 2D collision detection routines developed in my column "Crashing into the New Year" (Graphic Content, January 1999).

Later, I may wish to allow the balls to jump. It would then be easy enough to convert the collision back to 3D. These kinds of decisions are made all the time during the game production process. Since game simulation is all about speed versus realism, simplifying the problem if it works for your particular application makes sense.

FOR FURTHER INFO

- Beer, Ferdinand and E. Russell Johnston. *Vector Mechanics for Engineers: Statics and Dynamics*, Sixth Ed. New York: WCB/McGraw-Hill, 1997.
- Hecker, Chris. "Behind the Screen," *Game Developer* (October/November 1996–June 1997). Also available on Chris's web site, http://www.d6.com.

Rack 'Em Up

Using these techniques, I have created a demonstration of a simple pool table. The simulation uses rolling and sliding friction to simulate the way a real billiard ball moves across a table. Collision between balls is handled through conservation of momentum and the elastic collision model. There are several areas that still need work. Of course, I didn't talk about applying "English" to the shot by striking the ball off the center of mass. This technique is what makes shots such as a Masse, draw, or topspin possible. This is largely just a matter of where the impulse from the cue stick is applied. Also, the lack of friction between colliding balls does not allow effects such as collision-induced spin.

Another problem arises when we consider simultaneous collision between several billiard balls. When calculating the resulting force when two balls collide, it was fairly easy to determine the resulting force. However, when several balls collide simultaneously, the law of conservation of momentum becomes much harder to enforce. In order to calculate the resultant forces correctly, I need to solve several simultaneous equations. Obviously, this tends to complicate things quite a bit.

Alas, that will have to wait for another time. Until then, see if you can modify the source code to handle these effects. You can download the source code and the executable application off the *Game Developer* web site at http://www.gdmag.com. ■

Raising the Bar: Content Creation for Next-Generation Hardware

Recent years have seen the emergence of real-time 3D (RT3D) as the dominant venue for computer gaming. With the arrival of the Sega Saturn, Sony Playstation, and Nintendo 64, RT3D became an integral part of mass-market gaming. The "hardware-accelerated PC" followed shortly thereafter, and now many developers consider it the standard target platform, a view that might have seemed naïve just a few years ago. While the advent of these revolutionary platforms set the stage for RT3D to become a standard, the next generation of consoles (Sega's Dreamcast, Sony's Playstation 2 and Nintendo's much rumored Dolphin) and hardware accelerator cards promise to expand the technological boundaries of game development by a quantum leap, exponentially increasing the amount of digital canvas we developers have to work with.

With this increase in capability comes a correspondingly higher level of expectation on the part of the consumer, and an increased burden on us as developers to evolve with and take full advantage of the new hardware. This month we'll scratch the surface and attempt to predict some of the ways in which the artistic aspects of the development process will be affected by the latest and greatest technology.

More and Faster

At the most basic level, the technology advances can be distilled down to the following statement: You can display and manipulate much more content then ever before, and you can do this faster than ever before.

Sony Playstation 2 Specifications

CPU: 128-bit "Emotion Engine"
- 300MHz system clock frequency
- Cache memory instruction: 16KB, Data: 8KB + 16KB (ScrP)
- Main memory: Direct Rambus (Direct RDRAM)
- 32MB memory size
- 3.2GB per second memory bus bandwidth
- Co-processor FPU (Floating Point Unit)
  Floating Point Multiply
  Accumulator x 1, Floating Point Divider x 1
- Vector Units VUo and VU1
  Floating Point Multiply
  Accumulator x 9, Floating Point Divider x 3
- 6.2 GFLOPS Floating Point Performance
- 66 million polygons/sec 3D CG geometric transformation

Graphics: "Graphics Synthesizer"
- MPEG2 compressed image decoder
- 150MHz clock frequency
- 48GB per second DRAM bus bandwidth
- 256-bit DRAM bus width
- RGB: Alpha: Z-Buffer (24:8:32) pixel configuration
- 75 million polygons/sec maximum polygon rate

Sound: "SPU2+CPU"
- Number of voices ADPCM: 48ch on SPU2 plus definable, software-programmable voices
- 44.1KHz or 48KHz selectable sampling frequency

CPU Core
- Playstation (current) CPU
- 33.8MHz or 37.5MHz selectable clock frequency
- 32-bit sub bus
- Interface types: IEEE1394, Universal Serial Bus (USB)
- Communication via PC-Card (PCMCIA)

Disc Device
- CD-ROM and DVD-ROM

Sega Dreamcast Specifications

CPU: Hitachi SH-4
- 200MHz clock rate
- 360MIPS (millions of instructions per second)
- 1400MFlops (900MFlops with external memory)
- 64-bit data bus
- 100MHz bus frequency
- Capable of 5 million polygons/sec
- 800MB/sec bus bandwidth
- 32-bit integer unit
- 128-bit floating point bus

GPU: NEC PowerVRSG
- 100MHz clock rate
- 32-bit bus
- 9 billion operations/sec
- 3.5 million polygons/sec
- 120 million pixels/sec fill rate
- Perspective-correct texture mapping
- Bilinear and trilinear filtering
- Anisotropic filtering
- Gouraud shading
- 32-bit Z-buffer
- Colored light sourcing
- 16 levels of transparency
- Full-scene edge anti-aliasing
- Fog vertex
- Per-pixel fogging
- Bump mapping
- 24-bit color

Sound: Yamaha ARM7 ASIC
- 45MHz clock rate
- 40MIPS
- 64 sound channels
- Full 3D sound support

(continued next column)

(Dreamcast, continued)

Modem
- Upgradable 33.6KB per second transfer rate

CD-ROM
- Designed by Yamaha
- 1GB data storage
- 12 speed
- 1800KB data transfer rate

Memory
- 16MB main RAM
- 8MB video RAM
- 2MB sound RAM

3dfx Voodoo3 3500 Specifications

AGP Texturing
16MB SGRAM
Internal clock and memory speed of 183MHz
366 Megatexels/sec peak fill rate
8 million polygons/sec
Single-pass multi-texturing
128-bit 2D/3D processor
350MHz RAMDAC
D3D, Glide, OpenGL support
Video acceleration for DirectShow and MPEG 1&2

Nvidia Riva TNT2 Specifications

AGP Texturing
32MB SDRAM/SGRAM
9 million triangles/sec, 300 million pixels/sec
2.9GB/sec bandwidth
300MHz RAMDAC
128-bit TwiN-Texel architecture
Optimized Direct 3D and OpenGL acceleration
Video accelerated for DirectShow, MPEG1 and 2, and Indeo

FIGURE 1. *A sampling of the next-generation hardware specifications that developers must contend with in the near future.*

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

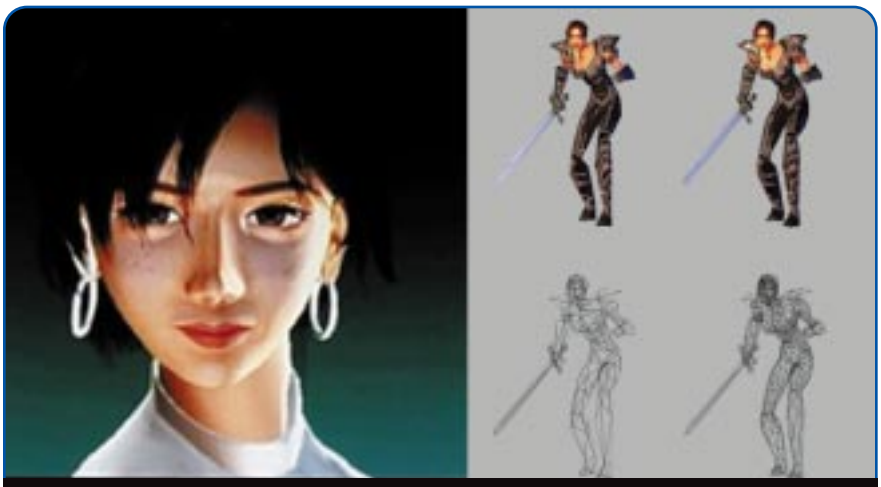


FIGURE 2. *Know when to splurge on polygons. At left, the polygon budget has been spent in facial details; at right, a more uniform polygon distribution.*

How much and how fast? See Figure 1 for some of the nitty-gritty details. Depending on the game premise, it's not unfeasible to be looking at on-screen polygon counts in the tens of thousands, and characters animating upwards of 60 frames per second. Our challenge as developers is to create enough content to showcase the technology while at the same time maintaining the high level of quality and polish which players have come to expect in today's RT3D games.

Managing Polygon Counts

One of the toughest restrictions to meet in RT3D development is the target on-screen polygon count. Since almost every object displayed on-screen is made up of polygons, each object contributes to the total on-screen count. In order to keep the game running at an acceptable frame rate, artists are forced into a balancing act that pits quality against quantity; trying to keep the polygon count of each individual object high enough to maintain a certain graphical feel, yet low enough so that you can put enough interesting things on-screen to keep the player continuously occupied. With the latest hardware advances, on-screen polygon counts of 20–30,000 polygons per frame are now feasible. As artists, we can take advantage of this by creating hyper-realistic characters, or by populating our worlds with vast numbers of objects.

Consider the images in Figure 2; the face on the left is the now familiar image from the early PSX2 demos, while the character on the right is our much beloved Rynn, from DRAKAN. In the case of the facial close-up, it's obvious that much of the detail has been spent on the character's head and face. In Rynn's case, a more uniform polygon distribution results in a more curvaceous body. The extra polygons in the face are not needed for Rynn because her character is usually a set distance from the camera. Figure 3 shows a landscape scene from DRAKAN with its current polygon count on the left (around 5,000 on-screen polygons) and a higher-resolution scene, where all of the trees have been pumped up to around 1,000 polygons each (this yields an on-screen polygon count of around 50,000 polygons).

So now we're back to the balancing act again, trying to decide where best to spend the extra polygon budget. This decision largely depends on the focus of the game play. For example, in a ground-based adventure game, where having an immersive environment is critical, it's easy to imagine spending 10,000 polygons per frame on trees, rocks, and undergrowth. In a character-based fighting game, most of your detail can go directly into the characters, modeling fingers and faces as well as weapons and special effects.

In either case, the increased detail and definition comes at a high development cost in that the artists' time increases disproportionately with the detail in the models. Consider how long it would take one of your artists to generate a model with only half as many polygons as the girl in Figure 2. What method would the artist use? Traditional polygon modeling methods would be tedious and time-consuming. More likely, the model would be built first with NURBS, patches, or some other surface tool, and subsequently converted to polygons. The task, which might previously have taken two to three days, may end up taking a week or two. And that's assuming your artists are already familiar with the techniques necessary to create the higher-resolution geometry.

Don't fall into the technology trap. Adding polygons to objects just for detail's sake can waste valuable time that should be spent on other areas. Take the time to plan out each scene efficiently, and prioritize the objects in the scene as they relate to game play and art direction.

Texture Generation

Even more important than polygon construction, textures on a RT3D object serve to flesh out the wireframe foundation and give the illusion of depth and detail. One of the main reasons you don't want to go overboard on the modeling is that you need to

FIGURE 3. *The landscape at left is comprised of 5,000 on-screen polygons. At right, each tree is around 1,000 polygons, causing the on-screen total to skyrocket.*

save enough time to add as much detail as possible to the textures in your game. And with texture memory footprints of 10MB or more, and data transfer rates clocking in gigabytes per second, it seems odd to speak in terms of "limits."

Thankfully, the days are behind us when our textures had to be 16 colors and 64 pixels square. With texture resolutions of 256 pixels square and higher, along with true-color bit depths, there's no limit to the level of realism we can achieve. And whereas we are usually forced to generate high-detail textures for characters and limit the texture size for environments, with the increased texture memory (upwards of 16MB or more on some cards and platforms), it's now possible to achieve photo-realism and a uniform pixel density in both the characters and environments concurrently. Additionally, we can now use streaming MPEG video as textures mapped onto 3D surfaces (an excellent tool for creating believable water, smoke, and pyrotechnics).

Here again, the hardware's increased capabilities prove to be a double-edged sword. The larger and more detailed the texture, the longer an artist has to spend generating it — and consequently, fewer textures can be created in any given time period. Where once we only had to worry about a single texture with its alpha component, now each texture has a multitude of options and fancy gimmicks from which to choose. Figure 4 shows an example of how a single texture map can be modified with the addition of several modifier maps. In addition to the main texture map, we also have the following: luminescence, for self-illuminating maps (objects that have a glow about them); environmental maps for reflective surfaces (water, metals, glass, and so on); "detail" textures to add random diversity to surfaces (fractal- or noise-based, these can act independently of existing mapping coordinates so that as you get closer to the object, the perceived detail remains high); bump maps to give definition and form to an otherwise flat surface; and masks to combine with any and all of these. And all of these effects can be animated.

Obviously, very few textures will use all of the potential variations available to them. Most will only use one or two variations, although this still increases the overall production time, especially if there is any reworking of textures. Previously, texture changes could be made at any time in the production cycle, even late into the beta period. Now with each level of complexity added to the texture, the overhead required to adjust or rework the texture is multiplied, so that instead of changing just one texture, you can end up having to rework several.

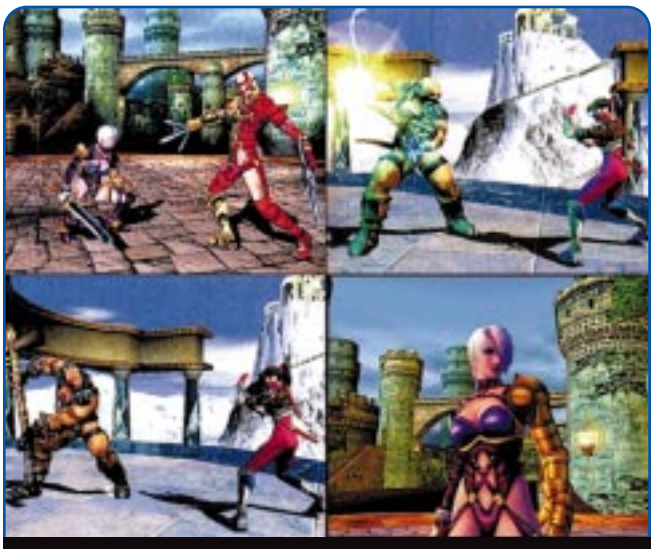


FIGURE 5. *A frame rate of 60 FPS, as in SOUL CALIBUR, can fool the eye into perceiving a continuous data stream.*

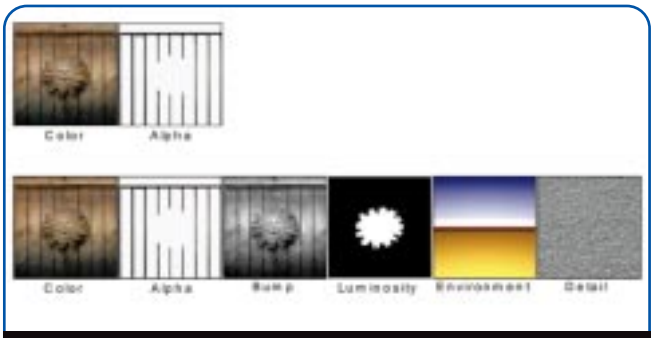


FIGURE 4. *Gone are the days of single textures with alpha components. Today's textures have large extended families of modifier maps, compounding the artist's workload.*

Animation at Break-Neck Speed

With graphics processors capable of manipulating geometry at rates greater than 50 million polygons per second, games such as Namco's SOUL CALIBUR (shown in Figure 5) can boast animation frame rates of 60 frames per second and higher. What's so special about 60 FPS? Well, it just so happens that 60 FPS is very close to the critical flicker frequency (CFF) for normal human vision. When this speed is reached, the images on screen stop looking like a sequence of frames and start looking like a solid, continuous data stream, just like the one we get when we look at the world around us.

Steven Schwartz (*Visual Perception: A Clinical Orientation*, 1st ed. Norwalk, Conn.: Appleton & Lange, 1994), explains the CFF of human visual perception as follows: "Consider a light that is pulsing, such as the blinking cursor on a computer screen. It is easy to discern that the light is blinking because of the low frequency, or rate at which it flashes. Imagine gradually increasing the frequency. As the light blinks faster and faster, it would eventually reach a rate where it would no longer be possible to detect that the light is flashing, it would look like a 'solid,' or continuous light. We say that our visual system has fused the flicker. The frequency at which that occurs is called the critical flicker frequency (CFF) or flicker fusion frequency (FFF). The CFF for human vision can vary with several environmental and psychological factors, but in general it varies between 50 and 70 FPS."

What this means to animators is that in animations of sufficiently high-fidelity (60 keyframes per second, or lower with an effective interpolation system), characters on-screen will display a stark, startling reality not seen before in the gaming world. Meeting this challenge means generating a lot of keyframes, either through procedural methods, motion capture, or by sheer, grunting hand-animation. All

this translates into a potentially more complicated and time-consuming process of character animation, further increasing the time involved in the development process.

In addition to the basic advances which allow us increased capabilities in modeling, texturing, and animation, there have been significant improvements which may bring even more drastic changes in the way we develop content. To engineers, the radically increased processor power and availability of dedicated geometry processors add up to more particles in our particle systems, faster and more accurate collision detection, more realistic and higher-resolution lighting and shadow casting, and the potential for trading out the normal polygon rendering routines for Bézier curves and even NURBS surfaces. One of the recent rumors swirling around Nintendo's Dolphin platform is that it will support an advanced NURBS renderer. As unlikely as this may seem, consider the potential benefits and corresponding upheaval in the development world if our real-time characters and environments had the smooth-edged look and feel of NURBS surfaces.

Ask Not for Whom the Bell Tolls

While all this increased capability offers us artists the chance to produce games with more flash, dazzle, and realism than ever before, the price we pay is an across-the-board increase of the time it takes to develop the content. A project which two years ago may have taken a ten-man art team 18 months to develop, could easily take the same team 36 months. Since publishers are still loathe to go beyond a two-year development cycle, without significantly changing the techniques we use to generate content, we will be obliged to add personnel, reduce the scope of our games, or both. Considering how hard it is to find talented, experienced artists in the gaming industry, it's easy to imagine a situation where the larger publishers with the most money begin siphoning off all their artists and animators to meet the demands of a more robust production cycle. This could ultimately leave many of the smaller development houses out in the cold, since they wouldn't have sufficient personnel to carry on the development of an A-title.

In a related side note, the word on the street is that Sony is aggressively pursuing development teams experienced in producing content for the PC. This is a sound strategy for two reasons. First is the obvious tactic that Sony wants to snatch up as many uncommitted development houses as possible to develop exclusively for their platform, thus reducing the market share and viability of its competitors. Second, those developers who have developed on the previous generations of consoles are used to working within the restrictions imposed on them by these soon-to-be-outdated platforms. Conversely, developers who have been working on PC games have been dealing with a constantly evolving platform, whose capabilities are at present only slightly below that of the next-generation hardware. Theoretically then, they are already familiar with the production levels required to create a product on the newer hardware. It will be interesting to see whether this strategy ultimately pays off.

Wrap Up

The history of videogames is, of course, also the history of the lumbering juggernaut of technology. Consumers do their part by dutifully soaking up the latest advances in technology, but the real burden of the technology race will always be shouldered by developers. With the imminent release of Sony's Playstation 2 and Nintendo's Dolphin platform lurking behind it, developers need to start preparing now in order to be ready. The tidal wave of technology is looming on the horizon, and we can either ride its crest or flail about in its wake.

And this wraps it up for me as well. I'll be taking a much needed couple of months off to do some research and replenish my creative juices. In the interim, id Software's Paul Steed has magnanimously agreed to man the Artist's View. I'll see you all again in a few months' time, tanned, rested, and ready to render. ■

Acknowledgements

Special thanks to Louise Smith, Vince Desi, Wyeth Ridgeway, Dave Coathupe, and Stuart Denman.

Matrox: Rolling with the Punches and Coming out Swinging

The graphics industry has a few old veterans, battle-scarred and a little weary, but one company continues to defy the aging process, and seems to be growing in strength. Matrox, a Canadian company founded in 1976, is still one of the leading brands in the business. It must be said that 1998 was a tough year for Matrox, a time when it had dropped off most game developers' radars as a hot technology company. I guess everyone was too busy with 3dfx and Nvidia, and then along came S3 to don the mantle of comeback kid. So, having sat in the pole position in the 2D graphics arena, Matrox seemed an almost-ran in the era of high-performance 3D graphics, driven as it is by the gaming community. Now, with the G400, a product that sits more comfortably in the worlds of both 2D and 3D performance graphics, Matrox has shown itself to be a perennial favorite of reviewers and consumers. The secret to Matrox's success cannot be easily explained, and is often shrouded in mystery, even to those in the industry that have known the company for all its years.

The Backgrounder

Matrox likes to be in the background. It is secretive and unapologetic. It is focused completely on its customers and products. It shouldn't be any mystery why the company is successful, but it's unusual for any graphics company to continue to ride new product cycles and remain with the leaders. The graphics business is supposed to keep knocking its winners off their pedestals, and the amount of investment required to compete in the modern world of highly complex 3D circuitry should be enough to bar all but a handful of heavily financed players. But Matrox remains insular, and less vain than its competitors.

Matrox was born as Matrox Electronic Systems, based in Montreal, Quebec. The company has always been privately held, and as a result, it has been subject to all kinds of speculation from its competitors. Initially, that didn't matter. Matrox made its money supplying multi-display subsystems to Wall Street traders, and stayed below the radar of other graphics companies more interested in the drawing power of CAD. It wasn't until the 1980s that Matrox got into supplying CAD graphics accelerators, but even then Matrox was eclipsed by bigger companies ranging from Video Seven and Hercules to, eventually, ATI, the other Canadian graphics company of note.

The company won a $100 million contract to supply the U.S. Army with a video disk training system in 1986. For many years, Matrox's competitors assumed that it was this single contract that kept the company afloat, and allowed it to build up its expertise in the PC graphics arena. Of course, there is no way of knowing the truth behind such rumors, but had Matrox not come out with the MGA chipset and Millennium graphics boards in 1993, they would have been unlikely to continue in the graphics business. In fact, in 1994, when the company established Matrox Graphics Inc. as an independent company, it was partly gambling on the success of the Millennium, and partly protecting itself from the vagaries of a graphics market that had decimated many of its pioneers.

Having dealt with the company as both a competitor in Europe and as an analyst, I assume that most of Matrox's success has come from a built-in confidence and determination, exemplified by people such as Lorne Trottier, Matrox Graphics' president and one of its cofounders. This may be part of the Matrox mystique as well, that the company receives an influx of young talent from local technical colleges, and senior management instills enthusiasm for Matrox and its products in the minds of a smart, young set of employees who aren't tempted the way their counterparts are in Silicon Valley; Matrox is a high-tech haven and breeding ground for its own biggest fans. To this day, I continue to be amazed by the strength of conviction among ordinary Matrox employees, and this element is as important to Matrox's success as its products.

Getting Sex Appeal

Still, all the OEM presence and distribution channels in the world have done little to enhance Matrox's reputation in a consumer market that is highly influenced by game developers. That may all change with the G400, which is aimed specifically at improving Matrox's retail awareness, and restoring some sex appeal to the

Omid Rahmat is the proprietor of Doodah Marketing, a digital media consulting firm. He also publishes research and market analysis notes on his web site at http://www.smokezine.com. He can be reached via e-mail at omid@compuserve.com.

brand. Matrox's 2D performance remained unchallenged for many years, but in the 3D arena the company has been sorely lacking in any significant technology or market position, despite having an incredible pedigree in developing 3D drivers and hardware since the 1980s. The results of Matrox's lack of 3D sex appeal is reflected in the big revenue hit the company's graphics business took in 1998, as shown in Chart 1.

O.K., so the figures are from a private company's press releases, but they're probably not too far off the mark. The Matrox folks keep their cards close to their chest, but they also tend to be quite honest when they do share information. Although Matrox continued to maintain most of its market share and unit sales in 1998, it took a big hit on the selling price of its chips because it didn't have a performance graphics part to compete with the likes of Nvidia, or 3dfx. Furthermore, Matrox was suffering from a squeeze on profit margins brought on by Intel's entry into graphics in 1997, a situation that resulted in almost every other graphics vendor taking a hit on pricing as well. Only 3dfx escaped the worst impact of the pricing pressures of 1998, and the gaming community became the focal point of all the graphics vendors. Matrox just didn't have enough to offer at the time.

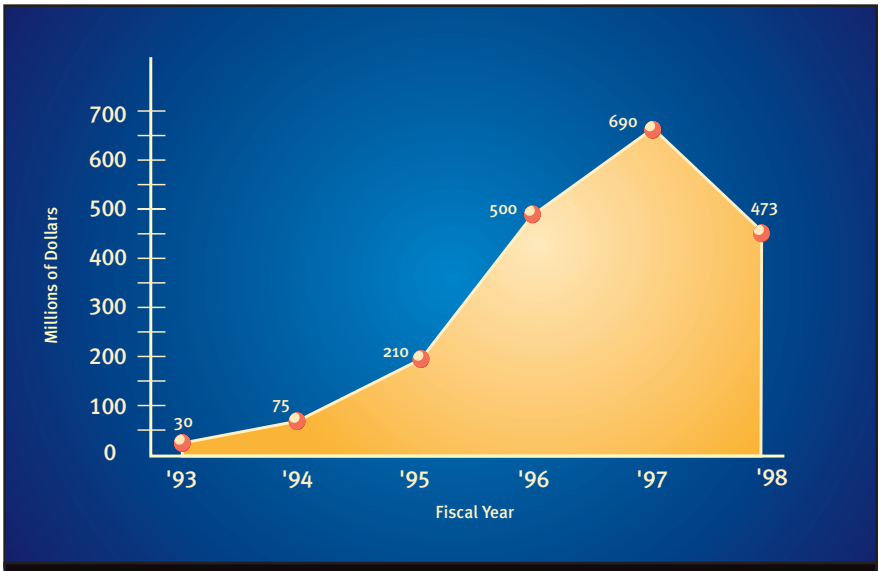


CHART 1. *Matrox Graphics' revenues, in millions of U.S. dollars. (Source: company press releases.)*

Back on Track

However, the G400 gives Matrox a unique position in the performance graphics market of 1999 — namely, environmental bump mapping in hardware. Of course, Matrox has also gone back to its roots in multi-screens, and provides a dual-screen output from a single G400 board. The dual-screen feature is not unique to Matrox (Diamond has had it available on its high-end Fire GL boards for a number of years), but Matrox is providing this feature on what is, in effect, a consumer, game-enthusiast product. And, as if Matrox wanted to confirm that it understands the new world of graphics, the G400 hits all the usual specification points for performance 3D: It has AGP 2X and 4X support, a 32-bit rendering pipeline, a 32MB memory load, and an 8-bit stencil buffer, with a dash of DVD video playback acceleration thrown in for good measure.

Then there are the usual flourishes by Matrox; the company has always added a wealth of software and utilities to its retail products, and with the G400 you get a Matrox software DVD player, MicroGrafx Simply 3D 3 and Picture Publisher 8, PointCast Network and Expendable from Rage Software, which, naturally, showcases the bump mapping capabilities of the board. However, even more commendably, Matrox has stepped up its online support and game oriented information sections of its web site. Matrox is as good as any of its competitors at developer relations, although the company continues to lag behind ATI, Nvidia, S3 and most definitely 3dfx in the consumer marketing stakes. With the G400, there seems to be a change of emphasis on the part of the company; gone are the days when Matrox preached to game players and developers to make up for its shortcomings, replaced by a clear-cut message that the company now gets gaming.

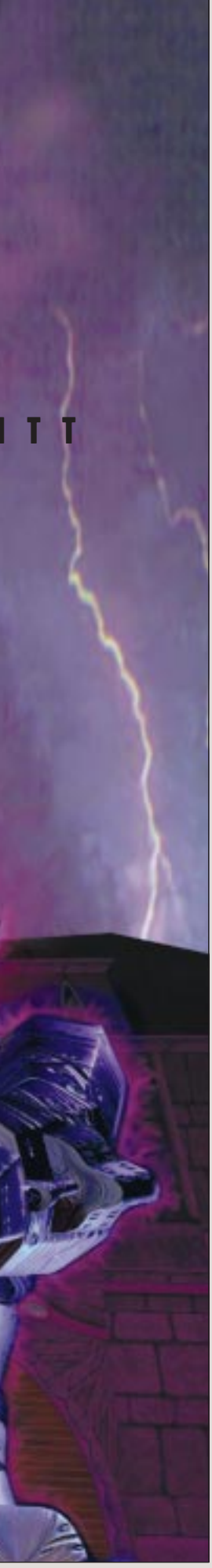
The game section of the company's web site is not the sexiest of the hardware vendors' 3D gaming sites, but it's practical and comprehensive, reaching the casual gamers who are most likely to be interested in the company's products. It's as efficient and unambiguous as anything else you'd see coming out of Matrox. Perhaps the word I am searching for is professional, but with oomph.

Epilogue

In the past year Matrox has successfully transformed itself, and had a drink from the fountain of youth. Whereas in 1998 the market was content to keep Matrox in the background, today Matrox is looking like an aggressive player: It has well reviewed and well received products; it has established its credibility in the gaming market with the G400 in a very short space of time; it is covering its traditional OEM and system integrator markets as well as ever, while leveraging a higher consumer profile with the G400's unique features.

In addition, under the surface, Matrox has reorganized its manufacturing operations to take advantage of better pricing in Asia, and is courting the Taiwanese motherboard makers to increase its market share in the PC graphics chip business. The company looks less and less like the Matrox graphics board company of Millennium days, and more like a competitor to ATI, Nvidia, S3, and 3dfx, just as it should be. Now there are five strong brands in the graphics business. Matrox's timing couldn't be better because around the corner, waiting to spoil the show again, is Intel. This time, Matrox is ready. ■


Back
LESSONS IN COLOR THEORY FOR
SPYRO THE DRAGON
BY CRAIG A. ST



In a relatively short period of time, videogame art has gone from the single white blocks found in PONG to thousands of colors wrapped around thousands of polygons. This in turn has allowed the worlds in games to evolve from a single black screen to immense 3D worlds. For those of us who find ourselves in the lucky position of being game artists, the trick is to find ways to leverage this cache of materials and palettes to create powerful, realistic worlds that draw in players. How does one do that? By educating yourself about the elements of color and production design and applying these lessons to your games, you can focus the player's emotion within a world, as we did at Insomniac Games in our recent Playstation title, SPYRO THE DRAGON. In addition to color theory, SPYRO's production design is explored by Insomniac Games artist John Fiorito (see p. 42).

THE DIFFICULTY OF COLOR Consider the example of the traditional painter. Subject matter, design, composition, and color must all be balanced together for the painting to come alive. Now take the example to the next level: A movie director or cinematographer has the same issues as the painter, plus the added complexities of motion, changing

After finishing a degree in Art Education from San Jose State, Craig Stitt started making videogames for Sega in 1991. While at Sega, he worked creating 2D textures and animations on Genesis title's KID CHAMELEON, SONIC 2, and SONIC SPINBALL. In 1996, Craig joined Insomniac Games and began creating both 2D and 3D art for Insomniac's debut title, DISRUPTOR, followed by their hit title, SPYRO THE DRAGON. He, and everybody else at Insomniac, is currently busy working on the soon-to-be-released SPYRO 2. Visit Insomniac Games' web site at www.insomniacgames.com.

perspectives, and timing. In games, we face the challenge of making a movie in which the viewer can go anywhere and do anything whenever he or she wants. Our "viewers" can see our world from any distance or perspective anytime they want. How much harder does that make our jobs? We not only have to worry about what is in front of the "camera," or more precisely, in front of the player, but what is behind, below, above, and all around him or her, in real time — and it has to be fun to boot (see Figure 1).

One cornerstone of traditional art and great games is the careful use of color. What makes getting color "just right" so complicated is the fact that color has a powerful effect on our senses, and we're also very sensitive to subtle color changes. A little too much blue in a scene, and the mood of the whole world changes. Fortunately, there are a couple of techniques that can make the process of coloring a game world more manageable.

A Gallery of Worlds and Emotions

Once your basic game design has been completed, as an artist or level designer you ought to start thinking about specific ways that the world you are creating will draw the player in. Which emotions will your world or your level need in order to draw players in and entice them to stay? When selecting a level's color palette, you're also making a decision about the underlying emotion that the level will convey to the player.

I have found that it helps for me to think of the game as a gallery of paintings. In a gallery, each painting must stand by itself, yet it should also support and strengthen those paintings around it. This happens only if each painting in the gallery is balanced; each painting has been placed with complementary paintings which have been thoughtfully selected, and carefully arranged and lit. So it must be with the art and colors in a game. Each level's colors and textures should be chosen to support and strengthen not just the level itself, but the whole game.

Broad Strokes of Color

I like to work in broad strokes of color, picking two or three colors that will be the foundation for the color design for each individual level. It's important to ask yourself, what emotions do I want to evoke in players when they first step out onto the playing field? The color palette you choose will naturally depend on the nature of the terrain, the architecture of buildings, time of day, and the effects of weather. However, if you're trying to evoke a particular emotion as well, you'll have to take that into account. Do you want to fill the player with awe and wonder, or fear and turmoil? Do you want them to feel comfortable and at home, or unsettled and far from home? Once the core emotions are laid out for each level, decide which colors best elicit those emotions from the player and also work well with the other level elements (terrain, architecture, and so on).

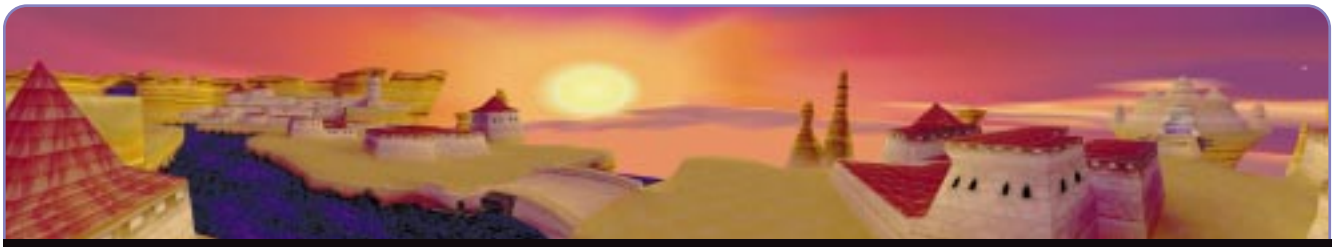


FIGURE 1. *An artist's job is complicated by the fact that in the 3D worlds, players can now see everything from everywhere.*

Consider how each of the various characters within the game will fit into your color scheme. For SPYRO THE DRAGON (a character-based platform game with a cast of dragons set in a medieval fantasy world), we made Spyro green in the earliest stages. But we quickly discovered that this didn't work with many of our primarily green environments — Spyro kept disappearing into the environment. We experimented with over a dozen different colors for Spyro before we finally found one that satisfied all our concerns: purple. As purple, Spyro didn't disappear into the grass on many levels, he was no longer the same color as several of our competitors' characters, the detail in his textures stood out, and he was a bright, fun character.

These same questions had to be asked for each and every object and creature in SPYRO's world. It also was important for game objects. For example, a great amount of treasure has to be found and collected in SPYRO, so it was important that players could easily identify treasure at great distances. We made this easier by applying motion and a little animated sparkle to the gems to make sure they were always the brightest thing around.

A level's core palette should contain only two or three base colors. Using these base colors, a level's detail is then defined using various shades and values along with small amounts of complimentary or contrasting colors. Be careful when extending this core palette because using too many colors can lessen the impact of any one color and you end up with emotional mud — just as if you mixed too many different colors of paint together. A variation to this rule is that some worlds may have multiple, distinct palettes, one for each area found within that world. For example, one palette for inside a building versus outside it. Even in these cases though, each of these distinct palettes should be limited to two or three colors, and there will still be one master palette, of two or three colors, which sets the tone for the entire world.

FIGURE 2. *During the development of SPYRO, a white board was used to display all 30 levels and their respective colors. It went through many changes before the end of the project.*

One way to check the overall state of your "gallery" — the color continuity between game levels — is to view screenshots or test swatches from each of the levels side by side, as if they were a color wheel or contiguous screenshots in a consumer game magazine. Decide if each individual level's look supports and strengthens the others. If not, rework the colors in one or more of the levels. More often than not, it doesn't take much of a change to find that balance if you catch the problem early. With SPYRO, as soon as we had a rough design document with its specified worlds, we put up a white board in the art room with a brief description of the sky and the core palette for each of the levels. Further along in the development process, small color print outs of screenshots augmented the white board (see Figure 2).

The Power of the Sky

If your game takes place outdoors, one of the dominant colors in the master palette will be that of the sky. Time and time again at Insomniac we have been surprised at the tremendous impact that the color and nature of the sky has on a world. Many times when we had a difficult time getting the right emotional impact from a SPYRO level, someone would suggest that we try a different sky. Every time we were surprised at the extent of the change brought to the level by the new sky. We learned that unearthly colors in a sky can create unexpected emotions, which is sometimes good if you want to make the player uncomfortable (as if he's in an alien environment), but if that's not your goal, use caution when dealing with sky colors (see Figure 3).

Recently, I designed a sky that was one of my all-time favorites. It was a misty green sky, filled with wispy clouds and distant planets. It complemented the world it was designed for, but something wasn't right. While beautiful, it also evoked negative reactions from people. Finally someone pointed out that it looked poisonous. That would have been great if that was what we intended, but this particular world wasn't supposed to be poisonous; threatening yes, but not poisonous. In the end, we went with a more naturally-colored night sky, which contained several moons and a haunting red glow on the horizon.

FIGURE 3. *By experimenting with different skies, the nature and emotion of an entire world can be radically altered.*

Consider the relative contrast between a world and its sky, and the differences in their color saturation. If the terrain is bright and saturated, then often it's helpful to color the sky using softer, desaturated colors. The reverse is also true. This helps set off the horizon against the skyline, which in turn gives the player some depth cues and aids navigation. When the terrain and sky are too similar in color or saturation, they lack the necessary contrast and appear to flatten out. Definition gets lost, and players often do as well.

Wallpapering Worlds

At Insomniac, we try to keep both the contrast and the color saturation down and still keep colors bright in our textures. Typically televisions, especially NTSC-based ones, pump up both the contrast and saturation of game col-

ors, pushing the images over the top into a cartoonish look. Sometimes that's the goal of the game developers, but more often then not, a slightly softer, more realistic look is better, even on a cute or light-hearted game.

SHADING AND LIGHTING The final major source of color comes from shaded textures, from techniques such as colored vertex shading. The addition of colored shading on top of rich textures can create a spectacular look, but sometimes it can be too rich. The problem is that the cumulative effect of colored vertex shading can over-saturate or intensify the contrast of texture colors. This is one more reason to keep the base textures somewhat desaturated — there is plenty of room to apply color shaders with out blowing the colors over the top (see Figure 4).

Where Am I, Where is it Safe to Stand?

The two practical aspects of videogame art show the player where it is safe or unsafe to travel, and to give him visual landmarks so that he doesn't spend too much of his time wandering around lost and confused.

The simplest way to show the player safe areas to walk is by defining edges. It's a time consuming task, but making sure that a real-time strategy game, for instance, has terrain with carefully highlighted nooks and crannies will add reality to the game and alleviate much of the frustration a player is bound to feel if he keeps getting killed because he inadvertently walks off cliffs, into quicksand, and so on. Sometimes edge definition requires a not-so-subtle value or color variation to properly indicate the edges of a walkway or where a ledge exists on the far side of a canyon. Proper texture design can also help define edges and boundaries.

Creating navigational landmarks can be helped by careful level design, but it also depends upon careful coloring. Sometimes it's the case of simply color coding similar looking objects so the player can differentiate between them. Sometimes it is difficult, when "landmarking" an area, to set aside the desire to create a truly realistic environment. There may not be a good reason why one section of the wall is colored differently from the rest, or why one cave would emit a blue light and another cave had a red light, but the player won't care about that nearly as much as they will if the caves aren't color coded and they repeatedly get lost because the caves look alike (see Figure 5).

Ageless Principles

Overseeing the color management within any game is an ongoing process. One of the most important things to do during development is to regularly take a step back and view the game as a whole. Examine each level and make sure that it works on its own and also supports the overall gestalt of the game. The textures and shaders should strengthen the settings, the settings should strengthen the game play, and the player, at a glance, should be able to see what's important what's simply eye candy. It's constantly a surprise to me, and everyone here at Insomniac, how powerful a role color plays in pulling all these elements together, or in tearing them apart.

FIGURE 4. *Top: A model with only simple vertex shading and no background sky. (Note the use of the different colored shaders on the island, bridge, and tunnel.) Middle: The same model with shading and the sky in place (Note how the color of the sky is reflected in the use of color in the shaders.) Bottom: The finished world with models, shaders, textures, and sky.*

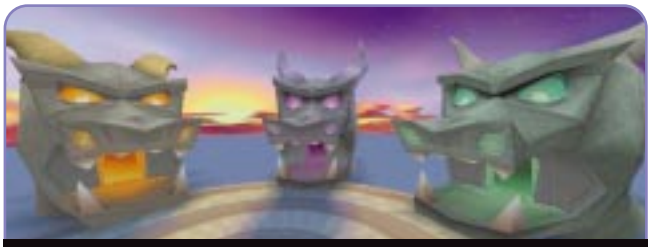


FIGURE 5. *Using color to differentiate or "landmark" areas can prevent players from getting too lost or confused.*

Take advantage of the lessons learned from the traditional schools of art. Basics such as color theory, balance and composition are ageless, and we must understand them and keep our creative edge sharp by using them. It is all too easy to get caught up in tile counts and polygon limits, and miss the fact that a level is lifeless or confusing because we failed to show adequate respect for the power of basic artistic concepts.

SPYRO Production Design

BY JOHN FIORITO

FIGURE 6. *Insomniac Games' production design for SPYRO THE DRAGON created a distinctive look for a new character and game. SPYRO's medieval fantasy world was composed with bright colors, soft decorative textures, and a cast of fairy tale characters.*

For the Playstation title SPYRO THE DRAGON, Insomniac Games faced the challenge of creating a unique look for a new game. The production design needed to be strong and consistent enough to endure a year-and-a-half development schedule, yet even more significant was the need to set SPYRO apart from an already crowded field of platform games and various titles that involved dragons and medieval worlds. To do this, our team established and followed a rigorous set of production design guidelines while making the game.

Our goal was to make SPYRO THE DRAGON visually unique, coherent, and memorable. Well before starting production we developed a set of artistic guidelines that we came to know as our "production design bible." Our basic rules were as follows.

USE BRIGHT, SATURATED COLORS. We felt that too many games, especially 3D games, achieved their look of realism by using muted color palettes which favored gray, brown, and black. By applying bright, vivid color schemes throughout SPYRO THE DRAGON, we would stand out from the start.

LEVERAGE THE LOOK OF CAMELOT AND FAIRY TALES. While there were many games set in medieval and fantasy worlds, most seemed to favor a darker more serious "Dungeons and Dragons" look. To complement our use of saturated colors, we pursued a lighter medieval style, closer to Camelot than to D&D.

CHARACTERS SHOULD COMPLEMENT THE ENVIRONMENTS. SPYRO was going to be a character-based game, and much of its personality came from its animation. Therefore it was necessary to make the characters stand out from the environment while maintaining our fantasy theme. Often much of a character's body was gouraud shaded which allowed its design and movement to stand out to the player. By keeping the colors of the characters bright, we further emphasized their appearance while maintaining our global color palette.

USE SOFT TEXTURES AND SIMPLE DECORATIVE MOTIFS. Our early tests showed that the Playstation's display sharpened textures to the point where highly detailed artwork degenerated into distracting visual noise. By avoiding hard outlines, high contrast, and too much detail in individual tiles, we were able to create a soft atmospheric quality which was aesthetically pleasing, yet not distracting to game play.

These basic rules formed the foundation of the game's appearance and helped us achieve visual consistency throughout the title. With these rules established and understood by our entire team, there was little room for miscommunication or confusion since we had established a look and a definitive way of qualifying it. (Figure 6).

SPYRO's universe comprises six distinct worlds, dozens of animated characters, bonus flying rounds, a secret level, and introductory and win sequences. With so many characters and locations, adhering to our production design was essential. Yet at the same time we needed to give each world as distinct a look as possible without straying from our basic design rules. One of our solutions was to design extreme variation into the game's environments. Spyro begins his adventure in a castle garden and proceeds through a desert, snowy mountain peaks, a swamp, dreamscape, and finishes in a mechanical world. Furthermore, the flying rounds are made of glowing crystals. Finally, to differentiate each world even more, we designed a dramatic three-dimensional panorama at the start of each

John Fiorito is an artist at Insomniac Games where he creates wireframe models, textures, and conceptual sketches for SPYRO THE DRAGON. He received degrees in architecture from the University of California, Berkeley, CA and illustration from Art Center College of Design, Pasadena, CA. Contact him via e-mail at jwf@insomniac.unistudios.com.

world to give the player a memorable first impression (Figure 7).

Within each of these worlds there were three levels of game play, a boss round, and a central navigational hub. Again, we faced the production design challenge of giving each level an individual look while maintaining a consistency to the overall world. Our first method was to vary the locations and geography of a level. For instance, in the desert (or "Peacekeeper") world, the settings include a fortress, pueblo village, ice cavern, and volcanic crater (Figure 8). We also created unique looks within a level by depicting varied and dramatic times of day. In the Artisan world, the levels occurred at daybreak, mid-afternoon, sunset, and under a full moon.

In SPYRO's worlds we developed a series of visual motifs that emphasized the connection between them. One such motif was a member of a family of balloonists that Spyro had to hire to travel from world to world — so seeing one of these balloonists indicated the way out of the level. Likewise, within a world, a system of portals gave access to each of the levels and each portal was styled to fit its world. Similarly, the architecture within each world maintained consistent design sensibilities which included flared bases, crenellated turrets, and decorative buttresses. For added variety, we sometimes located rival civilizations in a level, which allowed us to display the contrasting styles of the indigenous and invading characters and their architecture. Regardless of a level's theme, each one shared the same amount of ornamentation, and this gave each level a rich appearance.

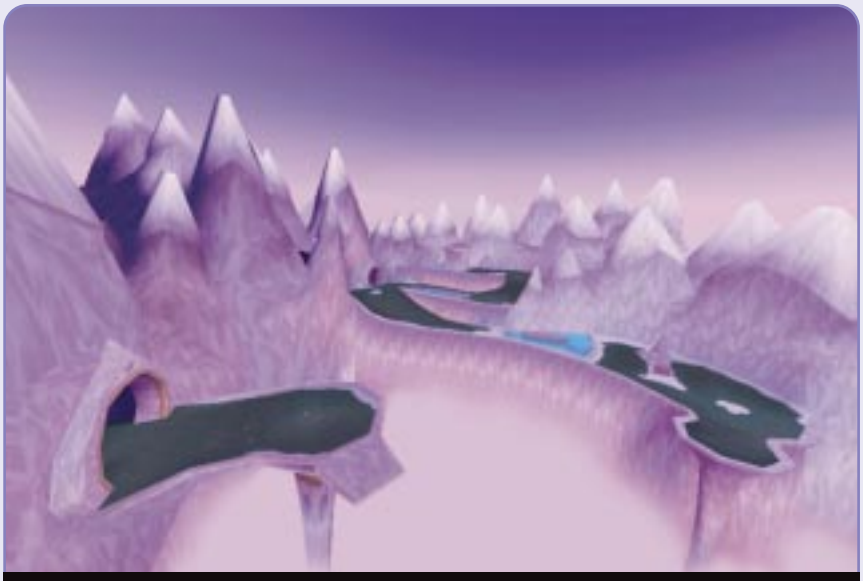


FIGURE 7. *To give each of SPYRO's six worlds a unique first impression, we often employed a dramatic three dimensional panorama. This opening scene of the Magic Crafters world established its basic characteristics and created a memorable view.*

Some of our greatest production design challenges arose within the levels themselves. In these massive, free-roaming, 3D environments, it was very easy for a player to become disoriented and lost. Just finding a level's exit often proved difficult. Searching for that last piece of treasure or completing a spatially-oriented puzzle could be especially taxing. Getting lost could even prove fatal in any of our timed flying rounds. To prevent players from getting too frustrated we needed to supplement our production design rules with a series of visual solutions that kept the navigation of a level as straightforward as possible. Here are the supplemental design rules we created.

USE LANDMARKS WHENEVER POSSIBLE. A landmark is a unique structure or geographical feature that distinguishes one area from another, which gives the player a reference

FIGURE 8. *Different settings, climates, and times of day added variety to SPYRO's levels while adhering to the overall production design. Here, in the Peacekeeper world, locations include a pueblo village at sunset, an ice cavern at night, and a desert fortress at midday.*

point in the level. Specific pieces of architecture, such as bridges, castles, fortresses, or other buildings that had a strong presence and appeared only once in a particular level proved to be the most effective landmarks (Figure 9). To enhance the uniqueness of a landmark, we added specific natural features around it, such as waterfalls, rivers, rock formations, or trees.

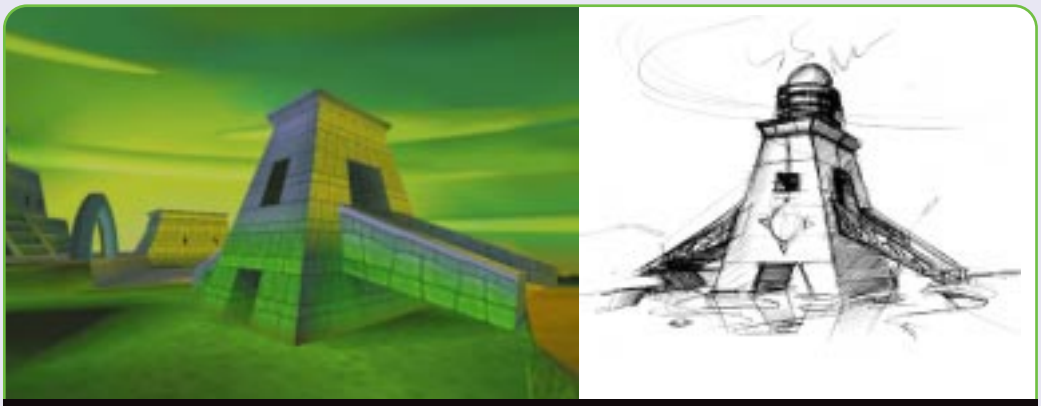


FIGURE 9. *Specific pieces of architecture proved to be one of the most effective ways of creating a landmark. From initial concept to the finished play field, we designed unique areas into the game to help players maintain their bearings.*

CREATE GATEWAYS AND CHECKPOINTS. To indicate that a player was making progress in a level, we tried visually to emphasize the transition between specific areas. This could be as fundamental as entering a castle gate or crossing a bridge, or as subtle as adding more snow on the ground to indicate higher elevations within the level. These transitions would often occur after passing through a narrow corridor or completing a long glide, and they served as memorable focal points in a level.

BALANCE INTERIOR AND EXTERIOR SPACES. We found that a variety of interior rooms and exterior spaces throughout a level provided a player with strong visual references. It also gave us the opportunity to create unique geometry. By placing a cavern between two valleys, for example, we not only defined different zones of game play, we were also able to change the look of the environment markedly and naturally. And by varying the types of spaces using caves, halls, valleys, or court-

FIGURE 10. *We found that solving a visual problem on paper was much faster than using a computer. Modifications to an area during the design phase could be completed in a matter of hours instead of the days or even weeks it took to implement the finished design on the computer.*

yards, we were able to create numerous unique locations throughout a level.

CHANGE SCENERY LIGHTING. Varying the lighting in different areas of a level created environments that were dramatically different. We lit interiors in a variety of ways, including natural light, fire and torch light, and any number of colored, glowing light sources. This helped players get their bearings, and also gave us added aesthetic opportunities throughout a level. Outdoor lighting varied as well. For instance, where a mountain divided two valleys, one side might be sunlit while the other lay in shade. Narrow spaces such as canyons allowed us to create strong shadows, while open places contained bright areas of sunlight.

In addition to our visual choices, a number of technical and game-play elements influenced our production design. Since Insomniac's game engine for SPYRO did not use fogging to reduce the on-screen polygon count, many areas of the game were designed to hide large portions of geometry in the distance. Wherever possible, the world had to be built in solid, continuous masses, such as mountain ranges, castle walls, cliffs, and buildings. Artistically, we were careful not to build monolithic fortresses, walls, or cliffs which dwarfed Spyro. Where we needed exceptionally large sections of geometry, we tried to reduce its scale by varying the surface with unique textures, buttresses, or decoration. Fortunately, and not coincidentally, our decorative approach, fairy tale theme, and softened textures also reduced the scale of Spyro's world.

Spyro's ability to glide great distances forced us to design areas that would contain him, yet not feel enclosed. This meant that most of our worlds had a horizontal orientation and simultaneously needed boundaries to halt his free-roaming nature. Wherever possible, we sought to vary the limits of the worlds. In addition to walls, we employed bodies of water, cliff tops, or infinite drops to define our outer edges — the more boundaries in a level the better.

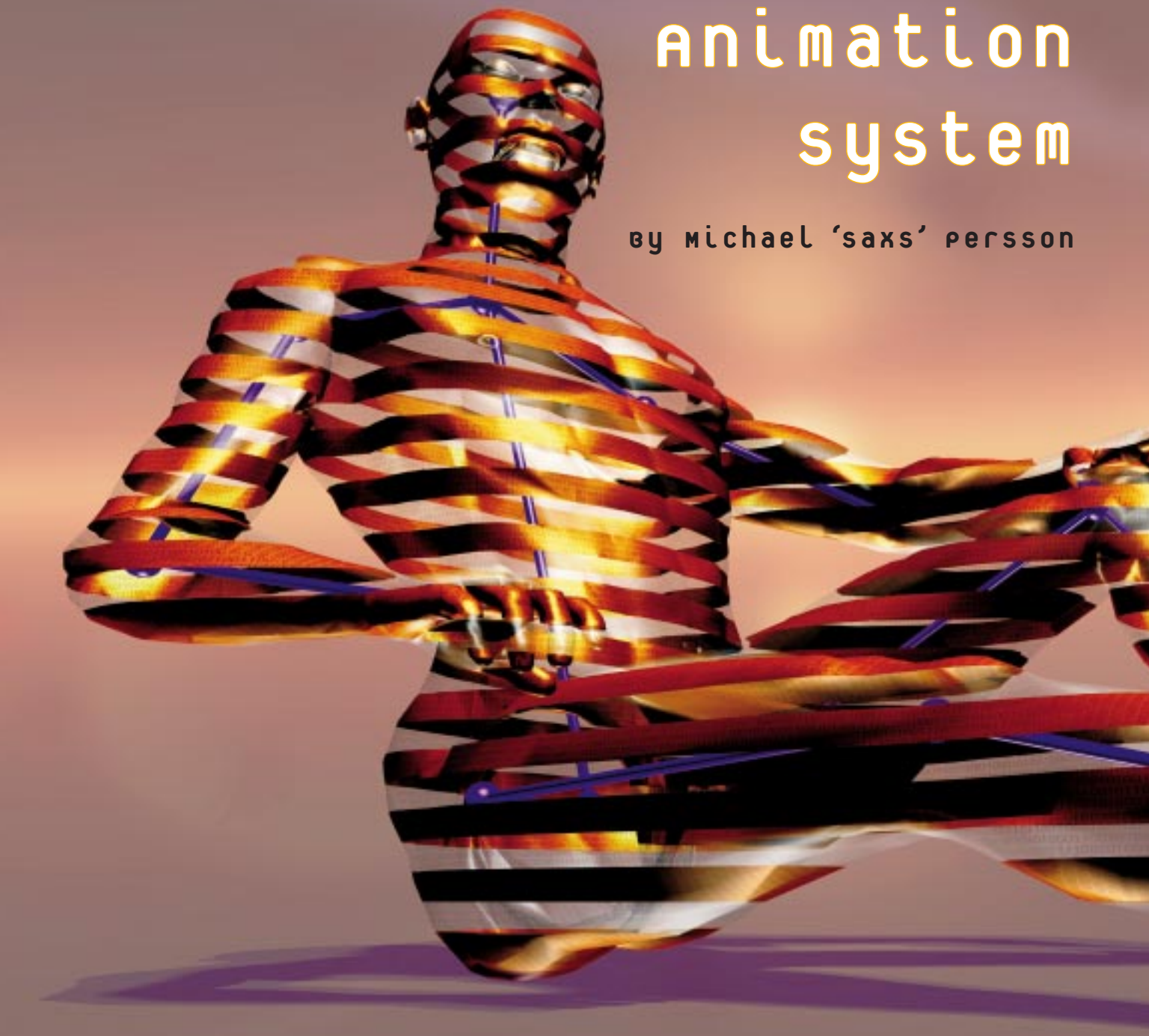
Our team soon learned that time was one of the biggest limitations in creating SPYRO's look. In every instance we needed to streamline our production processes. We relied heavily on sketches and diagrams in our production design, because solving a visual problem on paper was much faster than using a computer. Often, a series of preliminary studies could be created in a matter of hours, instead of the days or even weeks it might take to implement one finished design on the computer (Figure 10).

Ultimately, the production design methods we used to create SPYRO THE DRAGON were as much common sense as inspiration. We discovered, however, that defining them as rules or guidelines aided and quickened our production process. Our framework of visual motifs resulted in efficient development as well as consistent presentation. While we were always short of time, we used our production design to streamline the creation of finished artwork. Not only did the production design support the technology and game design, it also unified the playing experience. Players of SPYRO were rewarded with a game that was visually logical, possessed artistic continuity, and in the end was memorable. ■

Back

Behind the scenes of Messiah's character animation system

By Michael 'saxs' Persson



Saxs is busy working on MESSIAH, so if you have any questions you don't want answered, try e-mailing him at saxs@softhome.net.

This article is not a how-to guide, it's a brain dump from the perspective of the engine programmer (me) of Shiny's upcoming title, MESSIAH. Usually *Game Developer* articles are littered with formulas, graphs, and code listings that serve to up the intellectual profile of the piece. However, I'm not a mathematician and I don't feel the need to state any information in the form of a graph — in this article I describe problems, solutions, and things I've learned in general terms, and that allows me to cover a lot more ground.

My interest in character systems started more than four years ago, when I was working at Scavenger, a now-defunct development studio. I was assigned to develop a "next-generation" X-Men game for the Sega Saturn. Sega wanted motion-captured characters and chose to use pre-rendered sprites to represent them. I observed the

(Illustration by Spencer Lindsay. lindsay@etribe.com)

planning of the motion-capture sessions, examined the raw mo-cap data that these sessions generated, saw it applied to high-resolution characters on SGIs, and then received the frames which I was to integrate into the game.

The results were disappointing. The motion-capture data, which could have driven characters at 60 frames per second (FPS), was reduced to little bursts of looping animation running at12 to 15 FPS, and could only be seen from four angles at most. The characters were reduced to only 80 to 100 pixels high, and still I still had problems fitting them in memory. The models we spent weeks creating came out as fuzzy, blurry sprites.

Around that time, two new modelers, Darran and Mike, were hired for my team (and the three of us still work together at Shiny). These two talented modelers wanted to create the best-looking characters possible, but we didn't know how to justify the time spent on modeling super-sharp characters when the resulting sprites came out looking average at best.

Eventually, Sega Software stopped developing first-party games and X-MEN was canned. Soon thereafter we were asked to develop our own game. That provided me with the incentive to figure out how to represent characters in a game better. We knew we wanted at least ten or more characters on the screen simultaneously, but all the low-resolution polygonal characters we had seen just didn't cut it. So I decided to keep pursuing a solution based on what I had been working on for X-MEN, hoping that I'd come up with something that would eventually yield better results.

At first I flirted with a voxel-like solution, and developed a character system which was shown at E3 in 1996 in a game called TERMINUS. This system allowed a player to see characters from any angle rotating around one axis, which solved a basic problem inherent to sprite-based systems. Still, you couldn't see the character from any angle, and while everybody liked the look of the "sprite from any angle" solution, many people wanted to get a closer look at the characters' faces. This caused the whole voxel idea to fall apart. Any attempt to zoom in on characters made the lack of detail in the voxel routine obvious to people, and the computation time shot up (just try to get a character close-up in Westwood's BLADE RUNNER and you'll see what I mean). I tried a million different ways to fix the detail problem, but I was never satisfied. The other problem with a voxel-based engine was the absence of a real-time skeletal deformation system. Rotating every visible point on the surface of a character in relation to a bone beneath the surface was not a viable solution, so we had to pre-store frames and again, as in X-MEN, cut down in the playback speed and resolution. At that point I was ready to try a different solution.

When my team and I were hired by Shiny a little less than two-and-a-half years ago, I had done the prototype of a new character system after leaving Scavenger. Shiny was really excited about it and I continued to develop the system for the game that would eventually become MESSIAH. Let's look at that system and examine the solutions I came up with.

System Goals

There were a number of goals for the new MESSIAH character animation system. The first was to put as few limitations as possible on our artists. Telling Darran to do his best in 600 polygons would surely kill his creativity. At the very least, it was an excuse to create only so-so characters. At that time for PC games, the polygon count for real-time animated characters was around 400 each, and TOMB RAIDER topped the scale at about 600 to 800 polygons per character. My fear was that this number was going to change significantly during the time it would take to develop MESSIAH, and apparently I was right in that assumption.

Another problem I wanted to solve was the need for our artists to create a low-resolution version of a character for the game, a higher resolution for in-game cutscenes, and a high-resolution version for the pre-game cinematics and game advertisements. Why, I thought, should we have to do all this extra work? I wanted the artists to have no excuse for creating mediocre models, and I wanted to eliminate their duplicitous work.

Whatever system I created had to be console-friendly. My two targets at that point were the Sony Playstation and Sega Saturn. Both had a limited amount of memory — in Sony's case, only 1K of fast RAM. So it was important that the system could perform iterative steps to generate, transform, and draw the model.

Finally, I was convinced that curved surfaces would rule supreme in a few years time, so I wanted to make sure my system had that aspect covered.

I liked the visual results of my limited-resolution voxel model, and decided to make that my quality reference. The no-limit-on-the-artist modeling method seemed to work, so we stuck with that. This system supported automatic internal polygon removal, and using it Darran was able to dress the characters like Barbie dolls: he could stick buttons on top of the clothing, or model an eyeball and move it around without having to attach it to the eyelid. Clothing looked great, since all wrinkles were created using displacement mapping. Clothing shadows and light variations looked just right. In fact, most characters in MESSIAH now average 300,000 to 500,000 polygons to make the most of the system.

After a bit of back and forth, I decided to develop a system that fits patch-meshes as closely as possible to the body, and then generates the texture by projecting the original model onto the patch-mesh. To accomplish that, the following steps were necessary:

1. Slice and render a volume representation of the model with all of the internal geometry removed.
2. Connect the volume pixels into strings of data so it's apparent what is connected to what.
3. Apply bone influences.
4. Separate the body into suitable pieces, so a patch surface can be fitted around it.
5. Unify the body parts.
6. Generate a special mesh that goes between the separate patch pieces.
7. Prioritize the unified points.

Of course, somewhere in this process, I also had to figure out how to get the skeleton attached to the model.

STEP 1. RENDERING THE VOLUME MODEL, REMOVING INTERNAL GEOMETRY. The first step is to cut up a model into a predefined number of horizontal slices. This determines the resolution of the rendering. A reasonable number is 400

FIGURE 1. *This is how the data looks after volume rendering is completed.*

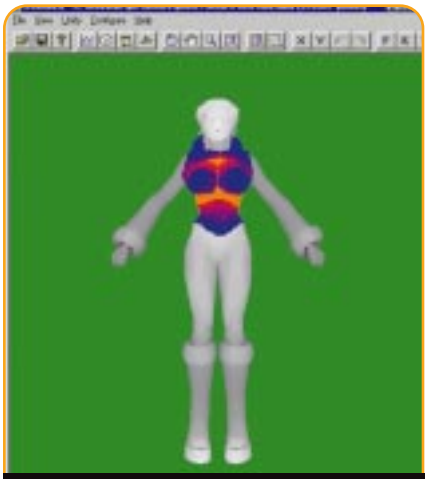


FIGURE 2. *Here is an example of how the bone influence is done on the upper torso.*

FIGURE 3. *The flexible projection paths and their editing.*

slices for models we're testing, and 1,000 to 1,500 for final models.

The dimensions of each slice's shell (outer surface) must be determined. I experimented with several methods, but in the end the simplest one was the one that worked best. (Usually, if a solution is too complicated, it probably wasn't the best solution anyway.) I "x-rayed" a character from four different angles (side, front, quarter-left, and quarter-right), noting the impact time for each ray, giving first and last hits priority since we know they belong to the outer shell, and storing the whole thing in four different databases, each corresponding to the x-ray orientation. Each database was cleaned for loose points (points having no immediate neighbor). Most internal points are removed by looking at the ray data. A combination of normal sign logic and proximity removes the inner layers, such as skin under clothing, and other points just under the surface.

STEP 2. CONNECT THE DATABASES. The next step is to connect the four databases into strings of data for each slice. This makes the final maps look a lot better since I can safely interpolate between points if I know they are connected. The routine goes through the four databases recursively, trying to find neighboring points one at a time. It finds neighboring points by taking into consideration criteria such as the surface continuity in the form of normal, continuous curvature, proximity to other points, UV closeness to other points, whether points lie on the same face, and so on. After this, we have neatly connected strings of data. A picture of the raw data can be seen in Figure 1.

STEP 3. APPLY BONE INFLUENCES. When we first began trying to apply an animated skeleton to the model, we didn't feel that any of the commercial packages would work for us. Neither Bones Pro nor Physique provided enough control. So Darran and I came up with the concept of painting bone influences directly onto the model (see Figure 2). The method is similar to using an airbrush: you set the pressure, method of application (average, overwrite, smooth), and start "painting" the bone influences. It's done in real time, so you can play an animation, stop at a frame when you see something that needs correcting, and just paint on the new influences. Using this method, we got very good deformation data from the start. Since the deformation is applied directly to the high-resolution model, you can regenerate your game model in any resolution without having to recreate all the influences. You can also incorporate influences from another model if its structure is similar, and just clean up the influences later.

STEP 4. SEPARATE THE BODY INTO SUITABLE PIECES. Before the model can be converted into the native game format, it is necessary to define all body parts. This is done using cutplanes. These planes cleanly separate the body into various parts (arms, legs, torso, and so on), and at the same time these planes describe the common spaces where patch meshes must be generated to cover holes between body parts that emerge during tesselation (more about that shortly).

Attached to the cutplanes is the definition of projection paths. These define the projection axis from which the patch mesh (think of the patch mesh as a tapered cylinder) is generated. The number of horizontal and vertical segments is defined for each body part, so you can change the output resolution of the mesh. Figure 3 shows what projection paths look like.

STEP 5. UNIFYING THE BODY PARTS. At this point, the model is ready for unification. This is the stage in which the mesh is fitted around the source material, at the appropriate resolution specified for each body part. It's as if you shrink-wrap cylinders around each body part. The strings of raw data are then projected onto the final cylinder, extracted into a texture map, and separated so it can be saved separately, and UV coordinates are stored for each mesh point. I don't call these points "anchor points," because we just use them as corner references for triangles, not for use in curved-surface calculations. Until now, it hasn't been feasible to do any spline interpolation of the points since hardware performance is still not able to handle the resolution that we save the game models in (about 10,000 to 16,000 points per model at full resolution), but the resolution is fine enough so we can snap to points when we tessellate down the model. It makes the run-time version a lot faster.

STEP 6. PATCHING HOLES IN THE MESH. Patching holes in meshes was an aspect

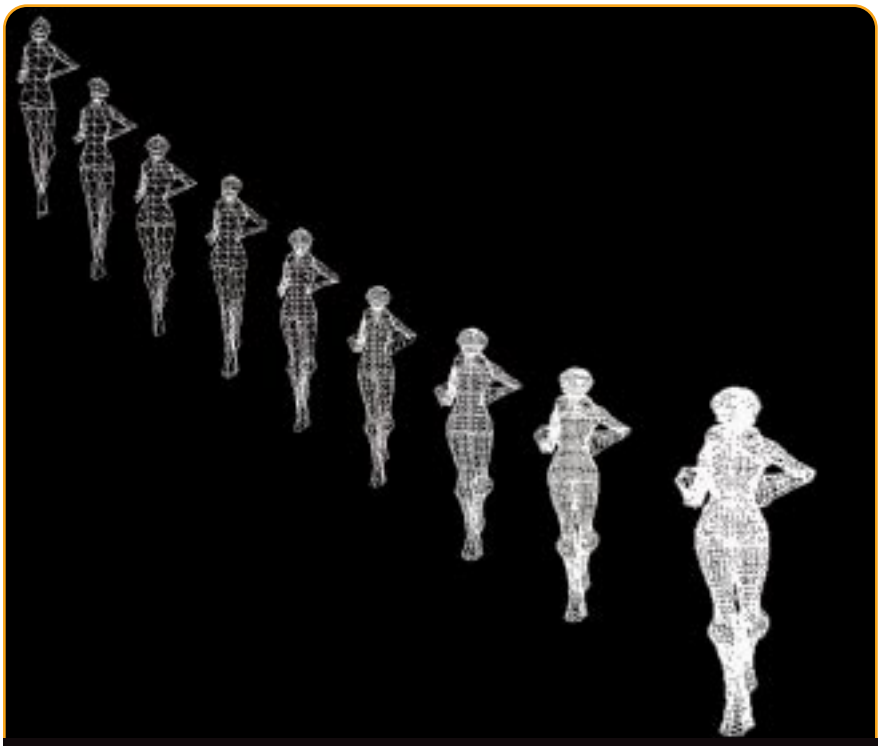


FIGURE 4. *A composite of the different stages of the tesselation.*

of the character animation process that proved to be very difficult to get right. We wanted a way to generate a mesh that would perfectly stitch up any hole that might be created when two cylinders of different resolution were joined together. For instance, to attach an arm to a torso, you cut away a round shape on the torso that fits the diameter of the cylinder representing the arm. A hole might appear at any point around the cylinder if connecting cylinders didn't match up perfectly.

The problem is especially acute when cylinders of different resolutions are connected; this generates a sharp break where they are joined. I changed the system so that the last slice of cylinder being attached (for instance, an arm getting attached to a torso) wasn't rendered prior to being connected. Instead, it was drawn directly onto the master cylinder, creating much better arm-to-torso transitions, and especially good leg-to-torso transitions.

Getting the texture mapping correct was difficult. Since the system uses intra-page mapping (to support the Playstation and for a more efficient video memory), wrapping is not supported. And because a character's individual body parts are basically just tapered cylinders, it was never necessary to have wrapping. However, during the process of unifying the various body parts into a single character, some method of handling wrapping had to be devised. To solve the problem of properly aligning textures so that body parts appeared in alignment, I generated a point on the master body part, typically the torso, corresponding to a UV coordinate of 0 on the body part being attached, allowing textures on different parts to match up correctly. That solved the texture wrapping problem.

Currently, I'm working on a routine that sorts out the drawing sequence of the patch mesh, since a bumpy master body part can screw up the projection and cause the drawing sequence to generate some incorrect points, thereby creating holes in the mesh. That can become a big mess.

STEP 7. PRIORITIZING POINTS FOR TESSELLATION. The final step is to prioritize the unified points. For instance, you want to make sure that the tessellator doesn't collapse the tip of a character's nose in favor of some less important surface point. As such, you can weigh that point so it won't disappear until the game turns off the priority routine when the model is displayed at low-resolution — in which case the nose doesn't matter anymore. Similarly, you can prioritize an individual slice of the model if it's important to the integrity of the model. That way you can make sure that the bend slice around the elbow is always there so the bend is kept clean. This process is vital for a stable-looking model — as a model drops polygons rapidly, there is a chance it will remove vital parts. Prioritizing slices and points goes a long way towards solving that problem. You might assume that prioritizing points all over the body would add a lot of polygons to your character, but in reality the tessellator just works around those points. In a wireframe view, you can see it just dropping more points in adjacent areas.

Working With the Final Model

The final model is saved with a separate map file for each body part, so you can easily load it into Photoshop to fix problems without having to do so on the model itself. When the run-time version of the character system loads a model for the first time, it reads in your preferences for the model's appearance, scales the maps to fit your restrictions, and quantizes the maps if you want indexed color. A compressed file of the model is saved at this point, so the system doesn't have to go through this routine every time the model is loaded.

Figure 4 shows the model in different resolutions. This shot was taken before the patch mesh was finalized, so there are some discontinuities in the hip area, but they are gone in newer versions of the tool.

System Pros and Cons

The process of creating a final model for MESSIAH is quite involved. It gets easier with each revision of the tools, but it still takes a bit of thought.

On the upside, we feel confident that the models we're creating are "future proof" (yeah, yeah — I know, nothing really is, but for the sake of argument, let's just let that comment

pass). When somebody gets the bright idea of upping the average number of polygons per character from 800 to 2000, we won't have to pull out our hair.

The tessellator is generally a lot better than other solutions at finding the right points on a model to eliminate. It doesn't create holes between each body part because of the patch mesh that stitches the holes.

Having separate body parts makes cleanly amputating a limb easy, and the model can still be tessellated away even after a limb is severed.

Multi-resolution mesh (MRM) technology as described by Hugues Hoppe (a researcher in the Computer Graphics Group of Microsoft Research), has won a lot of acceptance for its ease of use. It is a good solution for static objects, but if you have highly complex objects, modeling them with a limited polygon count and mapping restrictions is not so easy. Using our system, we can map each button on a shirt separately and the program determines the final map without screwing up the tessellation effectiveness. Another drawback to MRM is that as soon as you start animating MRM objects, you start seeing artifacts. These visual artifacts are generated by the way MRM-generated LODs bend, which in turn is due to the "spider web" appearance of the mesh around a collapsed vertex. The method by which MRM determines which vertex to collapse is based on the base frame of the model, so an MRM-based system wouldn't notice that the arm is bent in the animation and might remove vertices that might adversely affect the integrity of the model in that particular pose.

For static objects, MRM is preferred over the method I devised for MESSIAH, since it only changes one vertex at a time, so the mesh appears more stable. In fact, I made my own demo version of MRM for the Game Developers Conference earlier this year. Our team ended up using the routine for a few high-resolution objects in MESSIAH, and the technique worked nicely. On our system, we found that the tessellation artifacts get masked somewhat due to the fact that everything is moving and stretching with the animation.

Another important thing to note about our system is that the model can be processed in sections. You only need the rotation data from two slices at a time in order to render a model and make it fit easily in the cache. Even next-generation consoles have memory restrictions, so it's vital to control the temporary memory footprint that calculations require. Because our system generates strips and fans, the rendering speed of our system sees big improvements on any hardware.

Disclaimer

This article is not meant as an advertisement for MESSIAH or the character system I've written; I merely want to state an alternative to the conventional approach to generating characters. The process is being revised continuously. It's an on-going task to improve the process of converting models from 3D Studio Max to a game-optimized format. For now, Darran is keeping us busy with neat suggestions on how to make his life easier (and in doing so, we're swallowing our weekends to make the modifications). ■

MESSIAH's Character Animation Tools (Or, Why I'm Losing My Hair)

By Torgeir Hagland

As the tools programmer for MESSIAH, one of the challenges I faced as I built our in-house development tools was deciding how to handle the vast amounts of raw data — a model with 400 rings contained anywhere from 50,000 to 250,000 points (see Figure 1). Each of these points is weighted with up to three bones, which further slows the drawing process. Some design ideas were "borrowed" from 3D Studio Max, which uses a frame-rate threshold that drops the application from shaded mode into wireframe mode to increase drawing performance. So in the tool, if the user is rotating, panning, or zooming in on the model and the frame rate is too low, the program starts to skip points and slices to improve the display speed. And as soon an operation is finished, the model is redrawn again at the original resolution (see Figures 5 and 6).

Unfortunately, this method of boosting performance could not be used to paint bone influences onto the points (see Figure 2) — each point must be visible in order for the user to see the effect that bones have on points as the influences are altered. And as Saxs stated in his article, just drawing a character model was slow (especially when we increased the number of slices to 1,000 and we got a couple million points). This forced us to rely on regional updates when setting the influences within the model. Each time the model is redrawn, the 2D location and information about each point was stored in a huge buffer. So when the user actually "paints" influence with the brush, we perform a 2D region test (I actually use standard Windows functions for this, which is slow), recalculate the position (as the influence just changed), and redraw the data within this region. The results are satisfactory, and it lets users change influences within the model in real time (using a reasonable amount of points).

FIGURE 5. *A rotation is in process so detail is reduced on the model.*

FIGURE 6. *When the rotation is complete, original detail level is restored.*

The first generation of the character tools had statically defined linear projection paths, which meant that if a model had body parts that were arced, curved, or bent, the generated map would be skewed when unifying. In other words, it was not a visual process at all. When it was time to upgrade the tools, the modelers had come up with some not-so-humanoid characters, which meant linear projections were bad. So we added visual editing features and the ability to save spline projection paths. A spline projection path is a type of positional keyframe system: you create position keys, move the positions around, change tension, continuity and bias, and give each position/key a time value. This time value gives resolution to the spline, which in turn defines the resolution within the cylinder. This feature lets us change the resolution of the body part/mesh as we traverse down an arm, for example, and we generate higher resolution around the areas of a body part that need to bend, as we did in the case of the shoulder area shown in Figure 3.

A SCALABLE ENGINE REQUIRES SCALABLE TOOLS The scalability of the MESSIAH engine means more game data for developers to work with, which in turn means that our game development tools must be flexible enough to manipulate all that data. When we first started on the game, there was a limit of 400 slices per character. However, when that limit was raised, exporting the models from 3D Studio to raw rendering data became very slow. We had trouble fitting everything into memory, and machines often had to fall back and use swap drives to compensate (which is slow). When the rendering process ran out of swap space, we really had a problem. That's when we heard that Interplay had a network-rendering farm consisting of ten DEC Alpha workstations. We headed over to Interplay's offices to get some information on the setup, and afterwards we created a distributed computing version of our raw-data renderer. Using the new distributed renderer, 1,000-slice models that used to be rendered overnight could be rendered in one and a half hours.

The communication used for our distributed rendering system is very simple. The server saves a command file containing commands for rendering the slices from the different viewpoints to a shared directory and each client exclusively opens this file and looks for a command that hasn't been taken by any other client. The server checks this file at certain intervals to see if all the commands have been rendered. If it finds that all commands have been issued but some have not yet completed, it assumes that a machine is either very slow or has crashed, and it will reissue the command to another idle client.

When I started to write this article, the characters were rendered with a slice resolution of 1,000, but as I'm wrapping up this article, Darran has started to do 1,500 slices. That means that a raw binary file coming into our tool is 150MB, and that the problem now isn't just with drawing speeds anymore. It is actually becoming a problem for the artist to work with the model. Making sure that all the points are influenced and assigned to a body part is in itself a challenge. The tools were written with 400 slices in mind, and work beautifully even at around 800 slices, but with insane (Darran) resolutions, we need to come up with better ways to influence and generate the finished model. More sophisticated caching techniques and regional updates are going to be used, and that will hopefully enable us to go to even higher resolutions. In a year or so we'll probably have 2,000 to 2,500 slices, and 1GHz Pentium IIIs with 1GB of direct Rambus memory at our fingertips. When this happens, we want our tools to be able to take advantage of that power.

Torgeir Hagland programs his way around the world while conveniently dodging the Norwegian army. Write him at torgeir.hagland@powertech.no.

POSTMORTEM

Leading Lead CENTIPEDE 3D

by Richard Rouse III

Last year I was quite dismayed to see that Alfred Hitchcock's *Psycho* was being remade. It's one of my favorite films and I couldn't understand why anyone would think it necessary to remake it. How could a remake possibly approach the brilliance of the original? But then I noticed that Gus Van Sant, a filmmaker whose work I admire, was heading the project, so I thought the remake might have some merit. Still I wondered, what could have provoked him to tamper with such a classic? The irony is that at the same time I was dubious about the prospect of a *Psycho* remake, I was working on a new version of CENTIPEDE. The original CENTIPEDE, as designed by Ed Logg and Donna Bailey for Atari in 1980, is a work of art that I love

Richard Rouse III was Lead Designer and AI Programmer on CENTIPEDE 3D for both the PC and Playstation. Before that, he created the games DAMAGE INCORPORATED and ODYSSEY - THE LEGEND OF NEMESIS under the Paranoid Productions banner. Since working at Leaping Lizard Software, Richard has moved on to Surreal Software, where he is glad to be working on neither a remake nor a sequel. Feedback and other musings are encouraged at paranoid@panix.com.

and revere just as much as *Psycho*. So why didn't I have moral reservations about working on its remake? Perhaps looking at the *Psycho* remake from my point of view as a Hitchcock enthusiast who isn't a filmmaker, I could only see the new version as sullying the name of a classic. But working as an artist in the computer game medium, I saw that a new version of CENTIPEDE could be fresh and stimulating, using the classic game as a springboard for a completely new game playing experience. Without doubt, it would be a tremendous challenge to create a game that could live up to the reputation of the classic.

The Task at Hand

With plans for the PC and Sony Playstation as intended platforms, Hasbro Interactive was very clear in explaining that they wanted CENTIPEDE 3D to be true to the spirit of the classic CENTIPEDE. Hasbro Interactive had a commercial hit on its hands with FROGGER 3D, but at the same time was listening to complaints about the game. It seemed that many people enjoyed the first few levels of FROGGER 3D the best. It just so happened that these were the levels that most resembled the classic FROGGER. Because of this, Hasbro Interactive wanted to make sure CENTIPEDE 3D was closely tied to the original CENTIPEDE.

Leaping Lizard had already spent some time developing its outdoor 3D engine, which up until CENTIPEDE 3D was used exclusively by a flying combat game called RAIDER. Hasbro Interactive saw the technology and thought it ideal for their new version of CENTIPEDE. A prototype using the RAIDER engine with CENTIPEDE game-play components was created by Leaping Lizard in record time, and once delivered to Hasbro Interactive, the deal was soon signed. Eager to work on a project with the challenge of creating a new incarnation of CENTIPEDE, the team at Leaping Lizard was concerned with exactly the same goal as Hasbro Interactive: creating a distinctly modern game that still captured the feel of the classic. At the end of 1997 work began, using the existing RAIDER engine but without most of the RAIDER game play in order to make a new version of CENTIPEDE.

San Francisco art shop Mondo Media had impressed Hasbro Interactive with its excellent work on INTERSTATE '76, and it was brought in to do the cut-scenes for the new CENTIPEDE. Hoping to match the art style of the cut-scenes with the in-game art, Hasbro Interactive and Leaping Lizard decided Mondo Media would also do some of the animated game play art. The core CENTIPEDE game play demands that there be a large number of monsters and mushrooms on the screen at one time, and as such, a very low polygon count was required for all the models. The RAIDER engine utilized a level of detail (LOD) system, which swaps in lower polygon versions of models depending on the game's frame-rate and a given object's distance from the camera. Mondo Media, more experienced with high-polygon work, at first found it extremely challenging to stick to the 90-60-30 polygon limitations for the monsters' LODs. But as the project progressed, Mondo Media adapted to the polygon limitations and produced some great work, creating more than 20 animated character models for the game.

Designing a Remake

At Leaping Lizard, work began immediately on the project. Programmers started porting the RAIDER engine over to the Playstation, while designers and artists dove into the first world of the game, Weedom. Six levels were produced rather quickly, with the first two levels attempting to mimic the classic game play as much as possible. Hasbro Interactive didn't think these levels were close enough to the original, however, and by March of 1998 asked that we start work on a completely separate "classic game," turning what had been a one-game project into a two-game endeavor. As we studied the classic game by endlessly playing our authentic, coin-operated CENTIPEDE, we not only developed the mock-classic game, but started to rethink the design of the modern game as well. Up until this point, the modern game levels had not been very reminiscent of the original CENTIPEDE, and as we studied the classic, we began to see why.

Of the six original levels that we had designed, only two survived without major retrofitting of the landscapes, while one was completely overhauled and three were discarded. We then spent approximately three months working up three new levels, refining and balancing them until they were fun and reminiscent of the original CENTIPEDE. With these first six levels relatively finalized and armed with a better idea of what was going to work well in CENTIPEDE 3D, we spec'd out the rest of the game and designed and imple-

CENTIPEDE 3D

Leaping Lizard Software Inc.
Gaithersburg, Maryland
(301) 963-8230
http://www.lplizard.com

Mondo Media
San Francisco, Calif.
(415) 865-2700
http://www.mondomed.com

Real Sports Games, LLC
Elgin, Ill.
(847) 429-4670
http://www.realsportsgames.com

Release Date: October 1998 (PC); May 1999 (PSX)
Intended Platform: Windows 95/98, Sony Playstation
Project Length: 18 months
Team Size (PC): Seven full-project developers and two part-project developers at Leaping Lizard, working with the artists at Mondo Media.
Team Size (PSX): Three full-project developers and five part-project developers at Real Sports Games, with five part-project developers at Leaping Lizard, and the artists at Mondo Media.
Critical Development Hardware: (Beginning of project): 90MHz Pentium with 64MB RAM. End of project: 350MHz Pentium II with 128MB RAM.
Critical Development Software (PC): Watcom C++ 11.0a, Opus Make, Emacs, 3D Studio Max, Adobe Photoshop, RCS source control.
Critical Development Software (PSX): Metrowerks CodeWarrior for Playstation, Opus Make, Debabelizer, StarTeam source control.

mented 23 more levels in only three months. By working initially on only a few levels until they were actually enjoyable to play, we were infinitely more prepared to design the remainder of the game, and hence had to do almost no reworking for the rest of the project.

The Playstation Version

Around April, both Leaping Lizard and Hasbro Interactive became concerned that Leaping Lizard wasn't going to be able to get the PSX version of the game running by the holiday season deadline. This was largely because of difficulty finding an available PSX specialist, and a number of PSX programmers that Leaping Lizard had hoped to hire had backed out at the last minute. As such, both Leaping Lizard and Hasbro Interactive thought it would be best to bring in another team to handle the PSX conversion, preferably someone who already had a suitable engine running on the PSX. That team turned out to be Real Sports Games of Elgin, Ill. Real Sports Games had a very nice looking Playstation engine up and running its then-in-development JEFF GORDON XS RACING title, and thought they would be able to get the conversion done for the all-important Christmas deadline. The deal was finalized at the 1998 E3 Expo, and they started work on assessing and porting the title immediately.

Real Sports Games was given an extremely challenging goal: porting a game which was not yet complete. This made it more difficult to fully assess the scope of the project and to see how it would fit within the constraints of the PSX. Further complicating matters was the fact that, in order to get the game to work with their engine, they felt they had to rewrite large sections of the PC game's code from scratch. Though the decision was made to use the exact same level files as the PC version, the project became less of a straight conversion and more of a reworking of the PC version of CENTIPEDE 3D. Real Sports Games ended up neither doing truly concurrent development with the PC team, nor porting a completed game.

PC screenshot taken of first world, Weedom.

PC (top) and PSX (bottom) screenshots taken of the second world, Frostonia.

Following completion of the PC version in October, I was flown out to Elgin to work with the Real Sports Games staff on wrapping up the conversion and to make sure all of the correct design and game-play elements were functioning correctly. My stay was initially to be for two weeks. Once there, however, I observed how many of the game-play elements had yet to be implemented, and I began noticing how different systems in the game, though they might work on earlier levels, were not going to be adequate in later levels. My stay in Elgin stretched to three months as I started working on the coding of the game, and got all of the game-play elements working correctly. During this time, other members of Leaping Lizard were flown out to help finish the project, including programmers Chris Green and Gary Skinner, artist Jane Miller, and project manager Elaine Albers. In December, the game entered beta, where it stayed for four months, going through a particularly long and arduous quality assurance process. Fortunately, after December I was able to work remotely on bug fixes and to fine-tune game play using the excellent StarTeam source control system, which worked seamlessly over the Internet.

Working on CENTIPEDE 3D was at once an extremely gratifying, yet terribly confounding experience. The game's design seems to have worked out particularly well, with play that instantly reminds people of the original game. Getting that design implemented on the PC side was a challenging, yet rewarding experience. But then making that design function on a system significantly less powerful than the PC proved mind-numbingly difficult and unendingly frustrating. The entire team was disappointed when the product didn't make its Christmas ship date. Looking back on the whole project, there are some things that worked out gloriously well and other things that only cause us to hide our heads in shame.

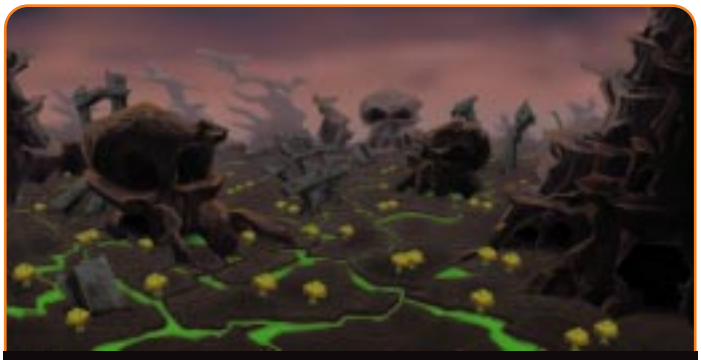
The evolution of a classic: From left, the original CENTIPEDE from 1981; how the mock-classic game looked at one point during development; how the mock-classic appeared in the PC version; and the completely 2D incarnation of the classic game, found in Playstation CENTIPEDE 3D.

What Went Right

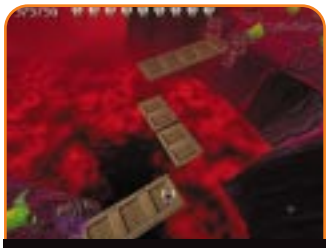
1. The scripting language. CENTIPEDE 3D made heavy use of Leaping Lizard Software's proprietary scripting language. Created to allow easy implementation of game-world objects with unique behaviors, the language was in part intended to allow non-programmers to make modifications to the game. But, as it turned out, this wasn't really the case; John Marzulli (a programming intern) and I did the vast majority of the scripting work in the game, and were able to do so only because of our programming backgrounds. Indeed, we pushed the scripting language far beyond what its creator, Chris Green, ever imagined it would be used for. The support for "`#define`" and "`#include`" style functionality made the scripts quite versatile and powerful. Except for the Centipede's behavior, all enemy AI in the game was implemented using the scripting language. Because the scripts are interpreted at run time, no recompiling was required when a script was changed, and only the scripts used on a particular level were even loaded into memory.

For the PSX version, however, Real Sports Games decided early on that a run-time interpreted scripting system was simply not going to work. In addition, floating-point numbers were used heavily throughout the scripts, and the PSX is notoriously slow at floating-point emulation. Some effort was put into hand-converting scripts into C code. But once the sheer number of scripts used in the game was fully understood, team members began to realize that perhaps there were simply too many scripts to convert in the time available.

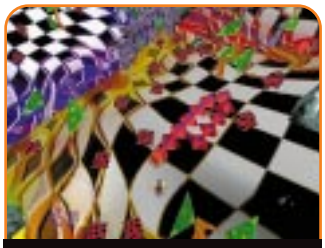
Chris Green came up with the idea of writing a converter that would take the scripts and convert them into C++ code, substituting fixed-point values for floating-point numbers in the process. The scripts, by their very nature, were completely platform-independent and lent themselves to direct porting. Chris's script converter worked out extremely well, and its success meant that once we had a one-to-one correspondence between functions called by the scripts and functions available in the PSX version, we instantly had all monster AI and other world-object behaviors converted as well. As a bonus, nearly all of the converted scripts were bug-free, since they had already been through a fairly rigorous testing cycle on the PC. Due to their strange syntax and frequent use of nested function calls, CodeWarrior took hours to compile the converted scripts and it generated a fairly large amount of code. But due to the PSX version's use of application chaining, only the scripts used on a given level were actually included in the executable, thereby allowing the game to fit in memory despite these inordinately large compiled scripts.



Concept art for the Evile levels.



PC screenshot taken of third world, Infernium.



PC screenshot taken of the fourth world, Enigma.

2. Studying the classic game. In creating the mock-classic game that was included with CENTIPEDE 3D, we had to spend a lot of time studying in fine detail the behavior and balance of all the elements in the original CENTIPEDE. We spent many hours in front of our authentic 1981 CENTIPEDE coin-operated game analyzing the game play. We came to understand the vital game-balance dependencies between the Flea, Spider, Centipede, and the mushrooms. Remove any one of them or alter how they work or when they appear, and the game falls apart.

Though the classic recreation game included with CENTIPEDE 3D didn't live up to anyone's hopes, studying the exact play mechanics of the original CENTIPEDE did help us immeasurably in designing the modern game. Understanding the classic game-play mechanisms was key to recreating the feel of the original game and allowed us to mimic exactly the movement patterns of the core monsters. Though the renderings of the insects in the new game are nothing like the classic, when players watch the monsters' mimicked movement patterns in the new game, they see a visual echo of the classic, thereby instantly reminding them of it visually. In the end, much of the game play that succeeds in CENTIPEDE 3D is the direct result of studying the game design work that Ed Logg did nearly two decades ago.

3. Correct PSX decisions. In the process of developing the PSX conversion of CENTIPEDE 3D, several seemingly insurmountable obstacles presented themselves. Fortunately, at these junctures the programmers were able to make the right decisions, which, had they been incorrect, could have doomed the project completely. Looking back now it's easy to see that the choices made were the only ones that could have worked, but when the decisions were made the programmers were relying primarily on intuition and gut instinct.

One of the earliest of these crucial decisions was to write tools which would convert the PC level and model files into PSX-usable form, primarily by removing all of the floating-point numerical data. Real Sports Games had bandied about the idea that perhaps new levels should be crafted for the Playstation version. But Real Sports Games decided that there was nothing

The Wee Miner from the Infernium levels, sketch and finished model.

The Wasp boss insect from the Infernium levels, sketch and finished model.

that the PC levels did from a design standpoint that couldn't be accomplished on the PSX as well, and since these were well balanced, thoroughly tested levels it was the right decision to use them in the console version instead of discarding them.

As the PSX development proceeded and the converted script files were added to the game, it soon became obvious that there was simply not going to be any way to fit all of the code into the two megabytes of main RAM found in the PSX. Once we had acknowledged that something had to be done, PSX project lead Brian Rice decided that application chaining was the only way to get the game to work, arguing down nay-sayers and cynics along the way. As a result of Brian's quick and authoritative decision making, the game has not just one, but 11 separate executables, with different applications able to quit and run each other as the player moves from menus to game play, or from level to level in the game. By separating out code that was used only in certain sections of the game and removing it from the executables (something which CodeWarrior's dead-code stripping capabilities simplified greatly), each of the 11 applications was squeezed into the two megabytes available. Without this application chaining, there is simply no way the game would ever have worked on the Playstation.

4. Separate "Classic" and "Modern" Versions of the Game. CENTIPEDE 3D started out to be one game but turned into two along the way. Hasbro Interactive, having learned that what people liked best in FROGGER 3D were elements that most resembled the original, wanted CENTIPEDE 3D players to have a game very much like the one they remembered from the early 1980s. Initially, the designers considered including all of the features that players would expect from a new console-style game — level-based play, exploration, power-ups, and so forth — in some of the levels, while having other levels which matched the classic game play exactly. The two styles of game play were to alternate throughout the levels, thus giving players both new and old style games in one.

But in attempting to mix the two modes of game play, neither was going to work out very well, and players would be confused and frustrated by the constantly shifting styles. The whole team soon realized that having completely separate games was the way to go. One game would feature modern-style game play that would be reminiscent, though quite different from, the original CENTIPEDE, and this style would be consistent through the whole experience. The other "classic" game would give the user game play identical to that found in the original game, though presented in a new isometric 3D view. For the PSX version, the classic game was taken one step further, using 2D graphics and sounds identical to the original game's. In this way, the two different styles of game play could exist separately, with players who wanted precise CENTIPEDE game play able to get that, while the modern game was allowed to flourish as a separate entity, no longer tied down to the constraints of the original game.

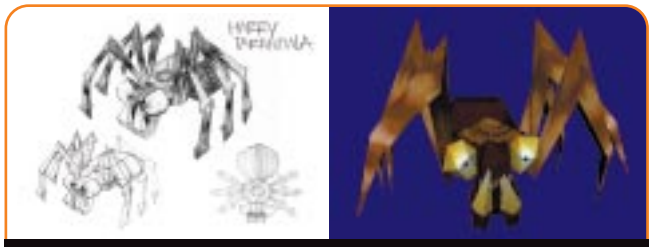
5. Designer Multi-tasking. There were two designers on CENTIPEDE 3D, and both of us were actively involved in the implementation of our design ideas. I served as both designer and programmer on the project, while Mark Bullock was both designer and artist. Together we worked out all the game's design issues and made all the levels found in the game.

By being intimately involved with both the programming and art in addition to our design responsibilities, we were able to come up with design ideas and then implement them almost instantly. Mark could throw together a concept sketch, model the piece, and then I could make the game element actually function in the game world. Instead of having to explain our vision to someone else, we were able to just "make it so" and see how well it worked in the game in a very short time. Though I have nothing against designers who are neither programmers nor artists, our ability to multi-task allowed us to achieve a certain Zen-like state during the game's creation, which let us crank out the game's levels in record time. Though we delegated art and programming responsibilities to other members of the team, because we had full understanding of the programming and art limitations of the project, we were able to avoid impractical or unaccomplishable design ideas, and instead focus on what we knew we could get to work in the limited amount of time we had to make the Christmas deadline.

What Went Wrong

1. Classic Game Should Have Been Emulated. Despite our best efforts, the classic game that comes with CENTIPEDE 3D is not precisely the game that Ed Logg created. There are differences which any hard-core CENTIPEDE fan will notice, and both the PC and PSX mock-classic CENTIPEDE games don't quite feel right. The new 3D visuals in the PC classic game don't really do anything to make the game any more fun, and even though the PSX goes so far as to look and sound

The Cockroach insect from the Evile levels, sketch and finished model.



The Tarantula boss arachnid from the Enigma levels, sketch and finished model.

exactly like the original CENTIPEDE, it still just isn't the classic.

Though many software companies are wary of emulators and the income they may be "stealing," here is an instance where one could have worked wonders. Many such emulators exist on the PC side, and one had even appeared for the PSX, as used in Midway's ARCADE'S GREATEST HITS: THE ATARI COLLECTION. Instead of the great quantity of man-hours spent trying to recreate the classic behaviors and game play, the same amount of time could have been invested in writing our own emulator that would have permitted us to include the authentic CENTIPEDE with CENTIPEDE 3D. Perhaps if we had realized early on that we were going to end up with two completely separate games, instead of the single game we originally designed, we might have seen that emulation was the best solution to the challenges that lay ahead. No one fully realized how much work would go in to recreating such a simple game precisely and I firmly believe no amount of work on the mock-classic would have resulted in a game that was as good as the original ROM. When attempting to pay homage to the brilliance of a classic game, nothing does as well as presenting the actual game itself, functioning exactly as it did when it was released.

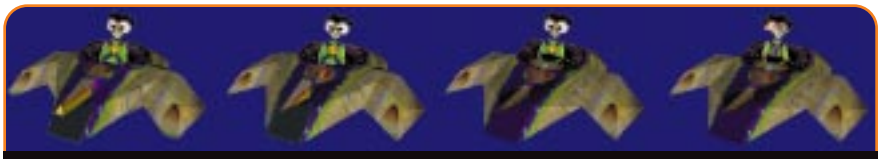
2. GAME WAS TOO HARD. From the beginning, CENTIPEDE 3D was supposed to be a mass-market title, a game anyone could pick up and play fairly well from the start. Definitely not aiming at the hard-core game player, Hasbro Interactive had seen wild success with its mass-market FROGGER 3D and wanted CENTIPEDE 3D to appeal to exactly the same consumers. It was said that even FROGGER 3D was too hard in places, and if we could make CENTIPEDE 3D easier, we'd be heading in the right direction.

But it was not to be. In the end, CENTIPEDE 3D turned out to be a far more challenging game than FROGGER 3D. As a designer on the project and the one largely responsible for balancing the levels from a game-play perspective, I take full responsibility for this shortcoming. The usual problems that lead to excessive difficulty can be found to be true here. First of all, none of the people playing the game during its development — designers, programmers, producers, and testers — could really be considered members of the computer game mass-market. We were all adept at a variety of games, including many distinctly hardcore titles, and as such, we were far more experienced than the target audience. The members of the team who were not such avid game players were not encouraged to play the game as much as they should have been, and consequently, they didn't voice complaints about its difficulty.

There were many complaints early on in the game's development that it was too difficult. Accordingly, I did some work to remedy this, and by that time the complaints had disappeared. I concluded erroneously that the tweaks I had performed had fixed the difficulty problems, and hence stopped worrying about them. What had really happened, I suspect, was that I made the game just slightly easier, and that in the time it took me to accomplish that, the people who had been complaining about the game's difficulty had gotten a lot better at it, and therefore stopped complaining. The solution to the challenge of testing the difficulty of a mass-market game, it seems, is always to test the product on a "newbie," someone who has never played the game before. Only then will one know for sure if the game is getting easier or harder.

Actually, I think CENTIPEDE 3D is just about as hard as the original CENTIPEDE,

perhaps even a bit easier. A game on the classic version only lasts a few minutes for the majority of players, and was designed as such to maintain a steady flow of quarters. Though CENTIPEDE 3D was aimed at the home market from the start and as such should have provided a much longer average play experience for users, I like to think it was the game's emphasis on being like the classic that made it so hard.



The Shooter — the ship the player controls through the game — has four different LODs. Since the default camera view is so far from the ship, most of the time players probably see LOD number three or four. Notice that Wally, the character in the Shooter, has turned into a plane by LOD four.

3. EXTENDED CRUNCH SCHEDULE PREVENTED LONG-TERM PLANNING. When I started working on the PSX version of CENTIPEDE 3D in October of 1998, everyone involved with the project was estimating that the conversion had about two weeks remaining. Two weeks later, after I had spec'd out all of the game-play elements that were still missing from the PSX version, it was decided again that there were two weeks left to get all of those game elements working. This pattern continued as the project extended on for another six months. No one involved fully realized the scope of getting CENTIPEDE 3D to function on the PSX.

From a business standpoint, the project needed to be done in two weeks, but unfortunately, from a programming standpoint, there was no way this could happen. Being in perpetual crunch-mode and constantly thinking the game was nearly finished, programmers were inclined to hack systems together in the quickest way possible, not fully considering all the problems that might result from such hastily constructed code. In the end, much of the rushed code had to be rewritten to fix all the bugs associated with it, which further extended the development time. The ongoing feeling that the game was nearly finished, but then not having it work out that way was also horrible for morale. If anyone had fully realized how much time was left on the project, the programmers could have taken the time to port systems correctly, look at the project wholistically, and structure their work better. In the end, coming up with a more realistic schedule might have shaved a month or two off the total project time and resulted in a less stressful experience for the team.

4. INCOMPLETELY PORTED SYSTEMS. In porting the game over to the PSX, some of the key systems in the PC game were initially rewritten without fully understanding or considering all the functionality they had to include. In part because of the intense schedule we were working under, programmers would often look at how a given game system worked in a few situations and then make sure their implementation of the mechanism would perform all that they had observed, instead of going over the PC code and converting it line by line. Unfortunately, usually only the early levels were considered during this observation stage and it would turn out that on later levels the system had to do a number of additional things that its ported incarnation could not. In the worst-case scenario, the only way to get that system to include this additional functionality was to rewrite it from scratch.

One example of this problem was water. CENTIPEDE 3D for the PC uses a massive textured plane for its water, which is drawn before any terrain geometry is rendered instead of clearing the screen, and which could thereby extend to infinity past the edge of the game world. This caused us to design many of the game's levels as islands set in the middle of infinite seas. The water also had the advantage that it could trivially be made to rise to any elevation, potentially flooding previous areas or the entire level, a feature we exploited when designing the game. Due to Z-buffer limitations on the PSX, Real Sports Games realized early on that it was impossible to implement water in the same manner in the console version. The developers at Real Sports went out of their way to create a water system for the PSX version that in some ways looked far better than the PC version's water. Instead of being a flat plane, its water was composed of triangles which undulated up and down in beautiful, wave-like formations. This looked great on the early levels that had relatively small quantities of water. Unfortunately, the more space on the level the water filled, the more memory it took up due to all the vertex data. The levels that were tightest on memory were ones that featured a lot of water, and a lot of time was spent pruning other objects from these levels in order to prevent them from overflowing the PSX's memory. Furthermore, the PSX version's water, due to its memory-intensive nature, could not extend forever as the PC version's water did, and consequently, the island levels looked ugly. If from the start the PC game's water usage had been looked at on all the levels instead of just the first few, it is possible that a completely different implementation of the water on the PSX could have been undertaken, one which would have included all the necessary functionality.

5. NO UPDATED DESIGN DOCUMENT. In a project such as CENTIPEDE 3D, where two separate teams were working on two separate code bases on games that were supposed to share a design, a living, up-to-the-minute design document is an absolute necessity. Unfortunately, beyond outlines written early in the project, most of the

The Centipede, Flea, Spider, and Scorpion each appear throughout the game, with different texture treatments for each of the five worlds. This helped give each world its own unique look. Here is the Scorpion in its five different incarnations.

design was understood by the two people who had to make it work — Mark Bullock and myself — and not by anyone else. Indeed, for the PC version it wasn't necessary for anyone else to understand all the intricacies of the design, since we were working as such a close-knit team, and when anyone had questions about the desired functionality of a particular piece of code or art, they could come to us and ask.

Most of the game play and all of the levels for CENTIPEDE 3D were created between March and September. Given that intense crunch schedule to get the PC version out for Christmas, there really wasn't any time for either Mark or me to maintain a complete and up-to-date design document, nor did anyone ever suggest we do so. Unfortunately, with a PSX development team a thousand miles away, a document that completely spec'd out everything that happened in the game was vitally important for keeping the two teams in sync. Without such a document, the PSX team didn't fully understand all of the different bits of game play and AI found in the PC version, nor did they fully comprehend the scope of some systems.

How Does It Feel?

Needless to say, everyone who worked on CENTIPEDE 3D grew stronger as game developers from the experience. I know that I've come to understand a lot about balancing game difficulty and the importance of an up-to-date design document in a project of this size. Leaping Lizard Software has since moved all its console development in-house, and its current project has simultaneous development on multiple platforms working painlessly.

Remakes themselves are tricky propositions, especially of much-loved pieces of art. Often it means that people are expecting you to screw it up and to have defiled a classic in the process. But when remakes work out well, they can take the strength of the original work and add to it their own interpretation. Think of the Jimi Hendrix Experience covering "Like a Rolling Stone"or "Day Tripper;" great songs before, great songs after (though very different in each rendition). Whether or not CENTIPEDE 3D succeeded in its aspirations to rework a classic into a fun, new experience is not something I can judge fairly from my perspective, though its development was a very stimulating creative endeavor. But of course, I never did see the new *Psycho*. ■

PC (top) and PSX (bottom) screenshots from the fifth world, Evile.

by Seonaidh Davenport

Smart Toys: Not Just Kids' Stuff

Smart toys are hot. Very hot. Last February's American International Toy Fair was flooded with new smart toys of all kinds: plush Teletubby Actimates from Microsoft, full play sets from Zowie Intertainment, the Playskool key-top toy, products such as Nerf Jr. Foam Fighters from Hasbro, Mattel's Barbie Digital Camera, and more challenging products for older kids, such as Intel's PlayX3 Microscope and Lego's Mindstorms Robotics Invention System.

All of these toys made it to E3 as well, but the game industry was more focused on poly-pushing to deliver the latest in muscle-popping heroes and big-breasted she-shooters. Color me sensitive: I develop kids' software. Color me hypersensitive: most game developers view smart toy development as a dull undertaking — *just another input device*, I hear, or, *how hard is it just to get the damned thing to play a bunch of audio files?* But as someone developing them, I can tell you that smart toys pose incredible challenges and rewards for game developers.

First, whatever adults think of that purple mush monster, Microsoft's Barney Actimate sold hundreds of thousands of units in its first year and generated $50 million in revenues. Second, the companies working in the field are advancing technologies such as voice recognition, infrared, and much more complex peripherals — which provide new ways for all audiences, not just kids, to interact with games. Third, creating new ways for users to interact with computers and each other poses truly tasty challenges for developers — particularly when those users are incredibly complex small creatures called children.

Designing kids' products is both maddening and delightful: children have less manual dexterity, may not be able to read, enjoy a thoroughly nonsensical humor, and have limited attention spans, yet possess a bizarre ability to obsess for hours over some detail that may be totally incomprehensible to adults. The challenge (read: fun) is delicately balancing "game" and "toy" to create a satisfying user experience that is greater than the sum of its parts.

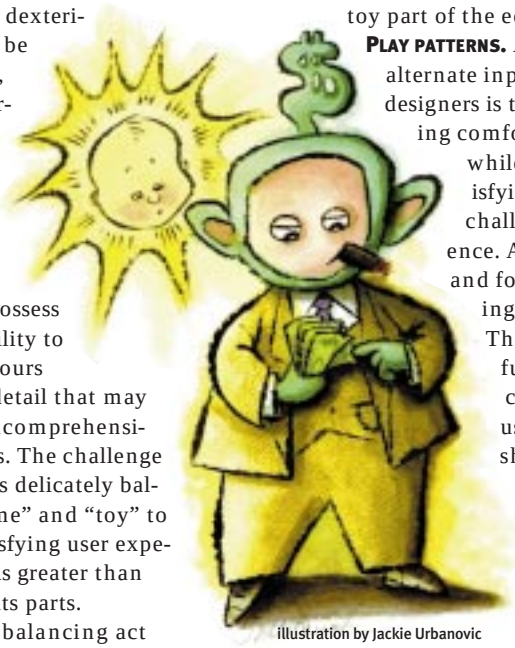


illustration by Jackie Urbanovic

I call it a balancing act because "game" and "toy" are in fact contradictory concepts: a game is an activity, governed by a collection of rules and logic, while a toy is a physical object around or through which play occurs. Game play is basically a linear progression through a finite set of interactions to a logical conclusion, at which point a player "wins" or "loses." Play with toys, on the other hand, is free-form and exploratory, with no defined endpoint; it stops whenever the child has had enough.

Applications written for smart toys are not too different from traditional computer games that we're all comfortable with: logical units of code provide rule sets that classify user input options and specify potential outcomes. As game developers, we're very good at this part. In order to find that game/toy balance, though, we have to crank our thinking around to accommodate the toy part of the equation.

PLAY PATTERNS. A toy is not just an alternate input device. The goal of designers is to keep the child feeling comfortable and in control, while providing a rich, satisfying, and pleasantly challenging play experience. Activities must first and foremost be fun, requiring minimal instruction. The program must gracefully accommodate a complex pattern of user attention as it shifts between the screen and the toy. (In our experience, attention is split about 50-50 between the toy and the screen when the toy is connected to the computer.) Finally, the child must have the latitude to explore: tying the child to a strict linear path and relying on error messages to keep them there only leads to frustration and the destruction of innocent plastic.

The program, in short, must be even more responsive to the subtleties of user interaction than is required in regular game play development. Any activity must be almost infinitely interruptible, because you can't dictate what the child should be doing next. You can't tell them they're wrong, but you had better not lose them. This introduces substantial complexities in the design of game

continued on page 71

Seonaidh Davenport is the Executive Producer for Consumer Titles at Human Code, an Austin-based developer currently producing REDBEARD'S PIRATE QUEST and ELLIE'S ENCHANTED GARDEN for Zowie Intertainment. She thinks she's in pretty good touch with her inner child. Contact her at seonaidh@humancode.com.

continued from page 72

or activity structure, state tables, dialog trees, asset loading strategies, error recovery, and so on.

PHYSICAL OBJECTS. As well as being conscious of play patterns, developers should get cozy with industrial design. Aside from logistical issues, such as how the connection works or where the toy can live in relation to the computer, the tactile experience must be satisfying for the child, with the input and feedback loop extending beyond what's happening on-screen into the physical toy. Physical toy design and software design must inform one another, so the toy and software representations should be designed in a recursive cycle. Keep in mind that the manufacturing cost of the toy will frequently drive the functionality of the software.

The physical toy radically affects software production and testing processes as well. Prototypes are expensive to produce (making them few in number), tend to be fragile, and don't work the same way the manufactured unit will.

So how do you deal with all this when designing and developing? How do you know you're on the right track? Early, frequent, and ongoing kid testing is absolutely critical. Developers must be unflinching in receiving feedback, and rigorous in its application to both the physical and software product design. Be patient and allow time for when your fundamental design assumptions are blown away by some tow-headed little darling.

I've only skimmed the surface of the major issues our teams have dealt with in smart toy development. While the range of products on the market prove that there isn't one right formula for the mix of toy and traditional CD-ROM game development, we're having tremendous fun experimenting with the mix. ■

ADVERTISER INDEX

NAME	PAGE	NAME	PAGE
Ascension Technology Corp.	22	Intel	7
Atari Games Corp.	67	Kinetix	2,3
Auran Developments Pty. Ltd.	46	MathEngine	29
Big Fat Inc.	69	Metacreations Corp.	30
Black Ops Entertainment Inc.	67	Metrowerks Inc.	C3
Boss Game Studios	66	Motion Factory	43
Cinram	69	Multigen	4
Conitec Datensysteme GmbH.	71	Nichimen Graphics	39
Crave Entertainment	65	Numerical Design	8
Credo Interactive	70	Okino Computer Graphics	21
Cyberware	27	Professional Employment	63
Digimation	16	Rad Game Tools Inc.	C4
Digimation	33	Resounding Technology	70
Digital Anvil	46	Retro Studios	68
Discreet Monsters	56	Savannah College of Art and Design	67
Duck Corp.	41	Savannah College of Art and Design	69
Evans & Sutherland	45	Seneca College	70
Future Light	36,37	Silicon Graphics Inc.	11
Hewlett-Packard	C2,1	Silicon Graphics Inc.	13
Immersion Corp.	19	Viewpoint	15