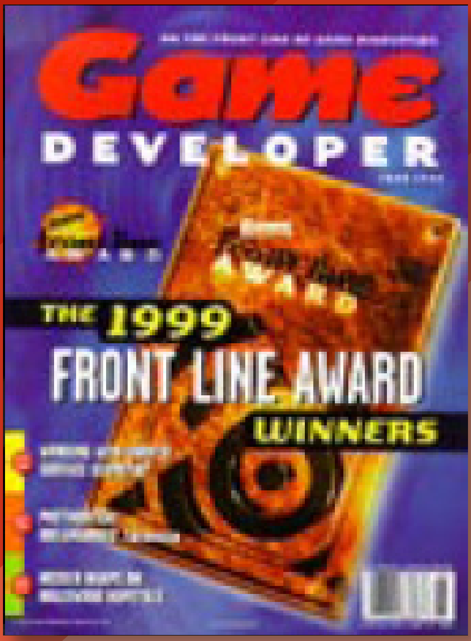
GAME DEVELOPER MAGAZINE

JUNE 1999



Back

A Business Case for Publishing Source Code

"This is your web site — it's Open Source — and I want you to help us build it.... Take a look at our source code. Send us your ideas on how to improve it, and build a better campaign web site. The Gore 2000 Volunteer Source Code Project wants to hear from you today!"

—**Vice President Al Gore, from his presidential campaign website**
http://www.algore2000.com

"I've heard reports of a number of companies calling software 'open source' even though it does not fit the official definition. I've observed some instances myself."

—**Richard Stallman, from the GNU Project web server**
http://www.gnu.org

At the Game Developers Conference in March, I got into a conversation with someone as to whether Open Source software could ever be commercially successful in our industry. While some Open Source game projects already exist (such as CRYSTAL SPACE — see http://crystal.linuxgames.com for more information), we quickly came to the conclusion that the chances of an Open Source game becoming a commercial success à la Linux were slim to nil. That's not to say that projects like CRYSTAL SPACE are fruitless — on the contrary, I can't think of a better route to learn about game technology and meet like-minded individuals than by working on a Open Source project. It's just that an Open Source project carries with it certain baggage, like implied permission to modify and derive other works from the source code. In an industry that already gets pummeled by critics for publishing too many copycat games, this reason alone makes it unlikely. That's just the beginning, too — there are a host of other reasons why a game company might not want to introduce the world to its intellectual assets. It can be a scary proposition.

All of this isn't to say that releasing source code is necessarily foolish. For some companies, releasing large chunks of source code has helped to sell product. id, of course, is the obvious example to point to. It's been releasing large amounts of its games' code to the public for years. I asked Brian Hook about the company's motivation for posting DOOM, QUAKE and QUAKE 2 source to id's website. His response might shed some light on both why this company went ahead and posted their code and whether you ought to consider it also.

Hook is quick to point out while their recent games had some form of source code released with them, none of the graphics, networking or core client code was distributed. The QUAKE game logic was based on QuakeC, and that was released. QUAKE 2 encapsulated its game logic in a .DLL, and that source code was also released. In other words, the company surgically removed certain portions of their games and posted that code. Furthermore, letting someone see your code does not imply permission to use it in another commercial game.

There were some strong benefits that id felt made its decision to post the code worthwhile, and you might take these reasons into consideration if your team is mulling over the idea. The bottom line, Hook explained, was that it helped build their player community. Once players had access to specific sections of the code, people began writing mods, building new levels, designing new artwork for the game, and so on. That in turn extended the shelf life of the game and fostered communication within the Quake community. Hook also said that an unintended benefit was technical feedback: "Individuals with strong competency in a field that is not one of our strengths may be able to offer advice on a subject; and legitimate bugs can be found in our code by 'virgin' eyes."

To release code or not to release — it's a big question. There are issues of additional customer support, fear that your competition might learn something from your code that you'd rather not teach them, and so on. But the concept has merits that you shouldn't dismiss without investigating what you might gain. "Power to the people" might be your company's motto someday. ■

[illegible]

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

ADVERTISING SALES

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

ADVERTISING PRODUCTION

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

MILLER FREEMAN GAME GROUP MARKETING

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

CIRCULATION

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

INTERNATIONAL LICENSING INFORMATION

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

Miller Freeman
A United News & Media publication

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

BPA INTERNATIONAL

www.gdmag.com

BIT Blasts

News from the World of Game Development

New Products

by Alex Dunne

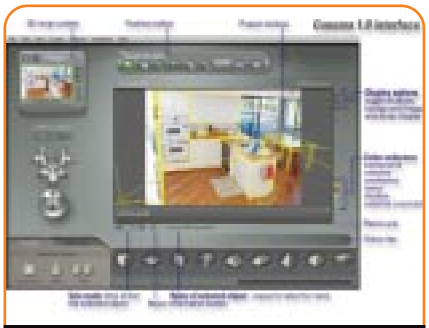
glAnalyze Dissects Games

3D PIPELINE, a contract programming firm specializing in graphics software development, recently released glAnalyze, an instrumented OpenGL driver which provides real-time information about OpenGL applications. glAnalyze lets you peek inside OpenGL applications to better undertstand how they work, on the premise that your analysis will help you build faster applications by studying the competition.

As an OpenGL application runs, glAnalyze allows you to set breakpoints to stop rendering. Breakpoints can be set on end of frame, end of primitive, glError, on specific OpenGL API entry points, and so on. This feature is especially useful for understanding individual primitives and frames. At break, the user can review all the prior API calls (including data) to see what led up to the current display. Breakpoints can also be invaluable for tracking down unexpected state changes or errant polygons.

Here's a scenario where glAnalyze would come in handy. Say you've just bought a game similar to the one you're currently working on, and you want to see how it works. Without the source code, it could be tough. However, using glAnalyze to control and manage the game's OpenGL calls, you could determine the type of geometry that the game uses, count and classify primitives (triangles, quadstrips, and the like), gain insights as to how the developers optimized it, what types of state changes are going on, count textures, and so on. glAnalyze runs on Windows 95/98/NT, and is priced at $295.

- 3D Pipeline Corporation
  La Jolla, Calif.
  (619) 551-5493
  http://www.3dpipeline.com/products/glAnalyze.htm



The Canoma interface for creating and exploring 3D environments.

EZ Environments in 3D

METACREATIONS unveils Canoma. Over the years, various companies have come up with ways to turn a 2D drawing or photo into 3D geometry. The latest company to throw its hat in this ring is MetaCreations, and I'd say it's done a pretty nice job with its new product, Canoma. With Canoma, which runs on either the Windows or Macintosh platforms, you can easily take a photograph and turn the entire scene into some basic 3D geometry that you could stroll around in.

Canoma imports photos, and using a beautiful KPT Bryce-like interface, you can start overlaying primitives onto objects in the scene. You can scale primitives so that they align well with objects in the photo, and you can add and subtract control points on the geometry when working with irregularly shaped items (like people). In other words, the product lets you apply geometry to textures, rather than the other way around.

Once you have defined the primitives in the scene, you can move about in the scene and see objects within the photo from different angles. Canoma takes care of correcting the perspective of textures. Of course, you'll have to take photos from different angles to capture back sides of object that a single photograph wouldn't reveal, but the application lets you stitch textures from different photos into the scene to make this feasible. Pricing for Canoma has not yet been revealed.

- MetaCreations
  Carpinteria, Calif.
  (805) 566-6200
  http://www.metacreations.com

Middleware for Physics

MATHENGINE. If the TRESPASSER Postmortem in this issue doesn't scare you away from the complicated world of realistic physical modeling in games (and hopefully it won't), a new product called MathEngine might bear some investigation. MathEngine is game "middleware" for handling object physics in real time. Like its name suggests, it's a number-cruncher that takes care of some of the hairy math code needed to model object interactions rapidly and in a way that looks realistic.

MathEngine's C language interface supports real-time articulated bodies, friction and a generalized collision handling system, and different surface material properties, plus a variety of other features. Though based in C, the company states that the SDK has "an object-oriented feel" that centers around a world object and body objects within that world.

It requires Windows 95/98/NT, and is available as a free download from the MathEngine web site. Pricing for commercial products is $1, plus $.01 royalty per game sold.

- MathEngine Inc.
  San Francisco, Calif.
  (415) 495-1738
  http://www.mathengine.com

Industry Watch

by Alex Dunne

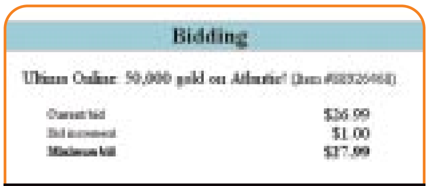
ACCLAIM REPORTED ROSY RESULTS for its second fiscal quarter ending February 28. The company's net revenues for that period were $135.7 million, an increase of 96 percent over the previous year. Net earnings for the quarter were $14.5 million, compared with a net loss of $1.2 million for the same period a year earlier. Acclaim's second quarter was buoyed by a great holiday season thanks to the N64 titles TUROK 2: SEEDS OF EVIL and SOUTH PARK.

SGI IDENTITY. Silicon Graphics, known the world over for years by its acronym, SGI, officially changed its name to SGI. According to the company, "the corporate identity strategy was supported by the results of global branding consultants, Landor Associates." For its next act, Landor will unveil a new identity for the National Aeronautics and Space Administration....

BLEEM BESTS SONY. Sony's legal department is getting plenty of experience as it jousts with various makers of Playstation emulators. Last time we caught up with Sony, they were wrestling with Connectix. This time, Sony's going after Bleem (http://www.bleem.com), a three-person company which manufactures a $25 PSX emulator for PCs. Once again, however, Sony was denied satisfaction in court: a San Francisco federal district court judge denied the company's request for a temporary restraining order, which would have stalled Bleem's product launch.

HIGH STAKES FOR SEGA. The importance of the U.S. launch of the Dreamcast is looking more and more like a make-or-break situation for Sega. Yasuo Tazoe, the corporate senior vice president at Sega Enterprises Ltd., estimated that the Japanese arcade market saw a 20 percent decline in the number of people playing arcade games from 1997 to 1998 — probably due to the increased use of cellular phones and the Internet by kids, he said. Sega of Japan's arcade division has been the only robust element of the parent company for some time, and to a large extent, it has helped float Sega of America over the past five years. If the arcade division continues to decline, a Dreamcast flop (i.e., insignificant sales in the next two years in the U.S. and Japan) could sink the company worldwide.

EBAY AND ULTIMA. If you've ever scanned the auction lists on eBay, you know that everything has a price tag these days, even money — game currency, that is. Yes, the ULTIMA ONLINE player community is putting eBay to good use selling all sorts of intangible game items, from high-level characters to real estate in the game worlds to, most interestingly, game currency. Players are selling their in-game nest eggs for U.S. dollars. The UO currency market hasn't quite settled down yet (week-long auctions don't make optimal currency markets), so if you've ever considered a career in arbitrage, here's your chance to make your first million.



Bidding

Ultima Online: 50,000 gold on Atlantic! (Item #[illegible])

Current bid	$26.99
Bid increment	$1.00
Minimum bid	$27.99

ULTIMA OLINE players bid for virtual money on eBay.

CROFT LANDS FIRST ENDORSEMENT DEAL. For a while, everyone was talking about ad placement in games. Finally, I'm pleased to see that the worm has turned. Welcome to game placements in ads. Lara Croft is the television spokesmodel for an Australian energy drink called Lucozade. Eidos proves that digital superstars are the best kind, because polygonal models don't care who they pitch for or how it affects their career.

GT INTERACTIVE HITS A ROUGH PATCH. The company said that it's going to lay off 35 percent of its staff, amounting to 650 employees, and that it will take a one-time charge of $25-$30 million related to its recently announced corporate reorganization and relocation. GT is expecting a net loss in the fourth fiscal quarter to the tune of $50-$55 million.

WHIFF OF Y2K? If you thought that the Y2K hysteria was exactly that, and that games had nothing to fear from clock-related problems, here's a story to chew on. When Windows altered system clocks around the world this Spring to account for Daylight Savings Time, many players of Hasbro's ROLLERCOASTER TYCOON found that they could no longer play previously saved games. To remedy the situation, Hasbro had to issue a RCT time-zone utility to customers. ■

UPCOMING EVENTS CALENDAR

June 10-11, 1999

PC Data's 1999 Trends
The W Hotel
San Francisco, Calif.
Cost: $295 to $495
http://www.pcdata.com/conferences

June 26-27, 1999

Develop! 99
Olympia Conference Centre
London, England
Cost: early bird rates available
http://www.unmf.com

June 30-July 2, 1999

MedPi '99 Software
Congress Center and
Auditorium of Monaco
Monte Carlo, Monaco
Cost: variable
http://www.unmf.fr/anglais/salons-a.htm

INDEPENDENT GAMES FESTIVAL
ANNUAL GDC INDEPENDENT
March 17, 1999
Fire and Darkness by Singularity Software
$10,000
Ten Thousand Dollars and no cents
Grand Prize

Singularity Software won Best Overall Game and the Audience Award for Fire and Darkness*. The grand prize was a check for $10,000.*

The First Independent Games Festival: Recognition Without a Million-Dollar Budget

by Ben Sawyer

It was a moment of innocence, perhaps naïveté, that demonstrated how new and exciting the first-ever Independent Games Festival was: as the development team from Vicarious Visions accepted their Best Audio award and began to leave the stage, they forgot to take their trophy with them. The event's emcee, Alex Dunne, tapped one of the departing developers on the shoulder and reminded him. "Here, you're supposed to take this with you," Dunne said, handing the team its award. And who can blame these winners for not fully comprehending that their independently developed game, Terminus, had just garnered a metallic plaque and $1,000 check? They were too busy thanking whomever it is that game developers thank.

So started the first Independent Games Festival (IGF), which was held March 17, at the Game Developers Conference in San Jose, Calif. The event grew out of an editorial that appeared in this magazine last September, and from its inception, the process of pulling together the IGF took everyone involved into uncharted territory. Hollywood may not know much about developing games, but you have to admit that Tinseltown has got that awards thing down pretty well.

The event began with an introduction from GDC Director Jennifer Pahlka, who described the launch of the IGF as a "labor of love" for everyone involved in its production. Attendees piled into the awards area, a large crowd began to build, and the 15 IGF finalists awaited to see what game would take home the evening's top award for Best Overall Game — worth $10,000 and publishing consideration by G.O.D. Games. To Pahlka's left on stage was an easel with one massively oversized check, with the requisite black drape hanging over it to conceal the winner's name. When Dunne came on stage and looked over at the check, he remarked, "I kind of feel like Ed McMahon tonight."

A SUNDANCE INSPIRATION. How did the festival reach this point? Like many crazy ideas, it began over beers at a San Francisco bar. More than a year ago, a friend of Dunne's returned from the Sundance Film Festival, and after describing the event, this magazine's editor felt like it was natural event for the game industry to adopt. Months later, after sitting on the idea for a while, he wrote an editorial asking "Where's our Sundance?" (Game Plan, September 1998). The feedback from the industry was encouraging (well, nobody called Dunne crazy to his face at least), so preparations were made to launch the event at the GDC. By mid-October, with the help of its primary sponsor, game publisher Gathering of Developers (G.O.D.), as well as support from AMD, Microsoft, Disc Makers, and GT Interactive, the event was underway. When I asked G.O.D. CEO Mike Wilson why his company decided to get involved, he simply stated that "it was something we couldn't afford not to do."

The announcement of prize money for the six awards ($1,000 each for Best Programming, Best Art, Best Audio, Best Game Design, and Audience Award, plus $10,000 for the Best Overall Game), not to mention visibility at the show, established the awards and energized the underground world

Terminus *from Vicarious Visions took the awards for Best Audio and Best Programming.*

Ben Sawyer is the co-founder of Digitalmill, Inc. (http://www.dmill.com). He is the co-author of the Game Developer's Marketplace *and more than ten other computer books. His latest top-secret project is handling the business development for a foundation-funded simulation product aimed at the higher-education market. He can be reached at bsawyer@dmill.com.*

of independent developers worldwide. In fact, entries were received from as far away as Pakistan, Russia, and Poland. With entries starting to come in, and sponsors and support in place, a panel of jurors was invited to help narrow all the entries down from nearly 100 submissions to 15 finalists, each of whom would be invited to demonstrate their games at the event.

By the time the awards were about to be handed out, a crowd of nearly 200 people had assembled. Singularity Software's FIRE AND DARKNESS took the first honor, garnering the Audience Award which was determined by popular ballot at the GDC. Vicarious Visions took the second award, Best Audio, for TERMINUS (a space combat RPG that was one of several titles to show that indies were very capable of putting out quality real-time 3D work). All the better for the company's apparent inexperience in accepting awards, the developers at Vicarious Visions had another chance to practice their acceptance style when they later won the Best Programming award. This time they remembered to take their award with them, and entered the record books as the first game ever to be a two-time IGF winner.

Andrew Leker of Mind Control Software accepted the award for Best Game Design for RESURRECTION. This game lay at the intersection of turn-based strategy and role-playing, and featured an interesting 3D landscape on which two opponent attempt to fan out their pieces and conquer the map. The Best Art award went to Poland-based Techland Software for their game, CRIME CITIES. Andrew Beard, who accepted for Techland, proclaimed upon accepting the award, "I'm pleased for Eastern Europe!"

THE ENVELOPE, PLEASE. Finally, the time came for the big check — $10,000 for the overall winner. FIRE AND DARKNESS, the stunning 3D real-time strategy game from Singularity Software, which had earlier earned the Audience Award, was proclaimed the overall winner. The Singularity Software team, made up of a group of friends attending Carnegie Mellon University and their younger high school friends from McLean, Virginia, took home the big award. The game combines an advanced 3D graphics engine, state-of-the-art networking technology, and a powerful combat simulation engine that *Next Generation Online* said "would make the game a legitimate competitor with Pandemic Studio's spectacular-looking DARK REIGN 2." After the ceremony, many of the other participants I spoke with admitted that they thought FIRE AND DARKNESS would win, indicating that the game was a winner even according to the team's IGF peers.

"We'd like to thank the members of the Academy," the team joked as they proudly posed for press with their poster-sized check in hand. The three representatives of Singularity Software thanked all of their peers and the people who voted for their title. In thirty short minutes these formerly inexperienced indie game developers had become pros, thanking the members of the academy. No one ever said that game developers weren't quick learners.

In the end Guha Bala, project lead for Vicarious Visions, summed up what it meant to win an award at the IGF: "We hope it means a deal!" With this statement, he seemed to speak for all of the finalists. For some of the finalists and winners, that hope may be realized. ■

IGF 2000 INFORMATION:

The second annual IGF will take place next year at the Game Developers Conference. The specific submission deadlines for the upcoming event are as follows:

- Applications for the event must be received by Wednesday, September 1, 1999.
- Finalists (those invited to the GDC) will be announced Monday, November 29, 1999.
- The IGF winners will be announced at the Game Developers Conference, San Jose, Calif., March 9-13, 2000.

Note that completed games will be required for both judging rounds, so make sure your game is ready for play by this September.

Updated information and IGF entry forms will be posted on the IGF website, at http://www.indiegames.com.

IGF WINNERS

Best Overall Game:
FIRE AND DARKNESS
by Singularity Software

Best Programming
TERMINUS
by Vicarious Visions, Inc.

Best Audio
TERMINUS
by Vicarious Visions, Inc.

Best Game Design
RESURRECTION
Mind Control Software

Best Visual Art
CRIME CITIES
Techland

Audience Award
FIRE AND DARKNESS
by Singularity Software

IGF FINALISTS

ACIDIA
by Whoola.com

BOOBIES
by Daedalon Interactive Entertainment

BFRIS ZERO-GRAVITY FIGHTER COMBAT
by Aegis Simulation Technologies

EVERNIGHT
by VR.1

FLAGSHIP: CHAMPION
by Keith Nemitz

FOOD CHAIN
by Cajun Games

JOURNEY INTO THE BRAIN
by Morphonix

MIND ROVER
by CogniToy

SEED
by Human Soft, Inc.

SLEIGHTS
by James Ferolo

V.D.
by Nothing Special Productions

IGF SPONSORS

AMD

DISC MAKERS

GATHERING OF DEVELOPERS

GT Interactive Software

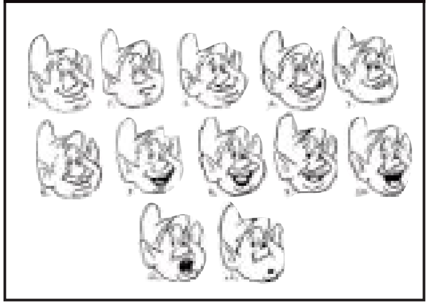
Microsoft

Read My Lips: Facial Animation Techniques

This column follows the erratic path of a professional computer graphics developer, namely me. Anyone who has ever been in a professional production situation realizes that real-world coding these days requires a broad area of expertise. When this expertise is lacking, developers need to be humble enough to look things up and turn to people around them who are more experienced in that particular area.

As I continue to explore areas of graphics technology, I have attempted to document the research and resources I have used in creating projects for my company. My research demands change from month to month depending on what is needed at the time. This month, I have the need to develop some facial animation techniques, particularly lip sync. This means I need to shelve my physics research for a bit and get some other work done. I hope to get back to moments of inertia, and such, real soon.

FIGURE 1. *The 12 classic Disney mouth positions.*



And Now for Something Completely Different

My problem right now is facial animation. In particular, I need to know enough in order to create a production pathway and technology to display real-time lip sync. My first step when trying to develop new technology is to take a historic look at the problem and examine previous solutions. The first people I could think of who had explored facial animation in depth were the animators who created cartoons and feature animation in the early days of Disney and Max Fleischer.

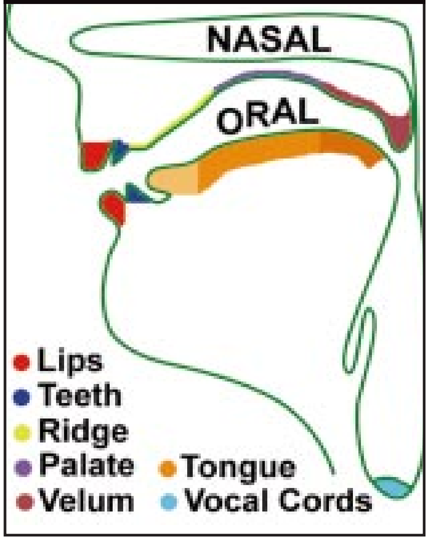
Facial animation in games has built up on this tradition. Chiefly, this has been achieved through cut-scene movies animated using many of the same methods. Games like FULL THROTTLE and THE CURSE OF MONKEY ISLAND used facial animation for their 2D cartoon characters in the same way that the Disney animators would have. More recently, games have begun to include some facial animation in real-time 3D projects. TOMB RAIDER has had scenes in which the 3D characters pantomime the dialog, but the face is not actually animated. GRIM FANDANGO uses texture animation and mesh animation for a basic level of facial animation. Even console titles like BANJO KAZOOIE are experimenting with real-time "lip-flap" without even having a dialog track. How do I leverage this tradition into my own project?

Phonemes and Visemes

No discussion of facial animation is possible without discussing phonemes. Jake Rodgers's article "Animating Facial Expressions" (*Game Developer*, November 1998) defined a phoneme as an abstract unit of the phonetic system of a language that corresponds to a set of similar speech sounds. More simply, phonemes are the individual sounds that make up speech. A naive facial animation system may attempt to create a separate facial position for each phoneme. However, in English (at least where I speak it) there are about 35 phonemes. Other regional dialects may add more.

Now, that's a lot of facial positions to create and keep organized. Luckily, the Disney animators realized a long

FIGURE 2. *Side cut-out view of places of articulation.*



Jeff Lander often sounds like he knows what he's talking about. Actually, he's just lip-sync'd to someone who really know what's going on. Let him know you are on to the scam at jeffl@darwin3d.com time ago that using all phonemes was overkill. When creating animation, an artist is not concerned with individual sounds, just how the mouth looks while making them. Fewer facial positions are necessary to visually represent speech since several sounds can be made with the same mouth position. These visual references to groups of phonemes are called visemes. How do I know which phonemes to combine into one viseme? Disney animators relied on a chart of 12 archetypal mouth positions to represent speech as you can see in Figure 1.

Each mouth position or viseme represented one or more phonemes. This reference chart became a standard method of creating animation. As a game developer, however, I am concerned with the number of positions I need to support. What if my game only has room for eight visemes? What if I could support 15 visemes? Would it look better?

Throughout my career, I have seen many facial animation guidelines with different numbers of visemes and different organizations of phonemes. They all seem to be similar to the Disney 12, but also seem like they involved animators talking to a mirror and doing some guessing.

I wanted to establish a method that would be optimal for whatever number of visemes I wanted to support. Along with the animator's eye for mouth positions, there are the more scientific models that reduce sounds into visual components. For the deaf community, which does not hear phonemes, spoken language recognition relies entirely on lip reading. Lipreading samples base speech recognition on 18 speech postures. Some of these mouth postures show very subtle differences that a hearing individual may not see.

So, the Disney 12 and the lip reading 18 are a good place to start. However, making sense of the organization of these lists requires a look at what is physically going on when we speak. I am fortunate to have a linguist right in the office. It's times like this when it helps to know people in all sorts of fields, no matter how obscure.

Science Break

The field of linguistics, specifically phonetics, compares phonemes according to their actual physical attributes. The grouping does not really concentrate on the visual aspects, as sounds rely on things going on in the throat and in the mouth, as well as on the lips. But, perhaps this can help me organize the phonemes a bit.

Sounds can be categorized according to voicing, manner of articulation (airflow), and the places of articulation. There are more, but these will get the job done. As speakers of English, we automatically create sounds correctly without thinking about what is going on inside the mouth. Yet, when we see a bad animation, we know it doesn't look quite right although we may not know why.

CHART 1. *American English phoneme summary chart.*

CONSONANTS			BILABIAL (both lips)	LABIOVELAR (lips and hanging down bit in the back of the throat)	LABIODENTAL (lip and teeth)	INTERDENTAL (tongue tip between the teeth)	DENTAL (tongue tip touching teeth)	ALVEOLAR (tongue tip touching ridge behind teeth)	PALATO-ALVEOLAR (tongue blade on the ridge behind the teeth)	PALATAL (tongue blade on the top of the mouth-palate)	VELAR (back of tongue near soft palate hanging down bit)	GLOTTAL (closing off in throat)
ORAL	STOPS	V	**b**uy					**d**ie			ha**g**	
	(airflow closed off)	VL	**p**ie					**t**ie			ha**ck**	**uh**-oh
	FRICATIVES	V			**v**ie	**th**y		**z**ion	vi**si**on			
	(airflow partially closed and turbulent)	VL			**f**ie	**th**igh		**s**igh	**sh**y			
	AFFRICATE	V							**j**ive			
	(combines a stop then fricative)	VL							**ch**ime			
	APPROXIMATE	C		**w**hy				**r**ye		**y**es		**h**igh
	(airflow narrowed, not turbulent)	L					**l**ie					
NASAL	STOPS		**m**y					**n**igh			ha**ng**	

VOWELS (airflow free flowing)	FRONT	CENTRAL	BACK
HIGH	b**ea**t		b**oo**t
	b**i**t		b**oo**k
	b**ai**t	b**u**t	b**oa**t
	b**e**t		
LOW	b**a**t		b**ough**t

V = Voiced
VL = Voiceless
C = Central
L = Lateral

FIGURE 3a. *Side view of the sound [b], as in "buy."*

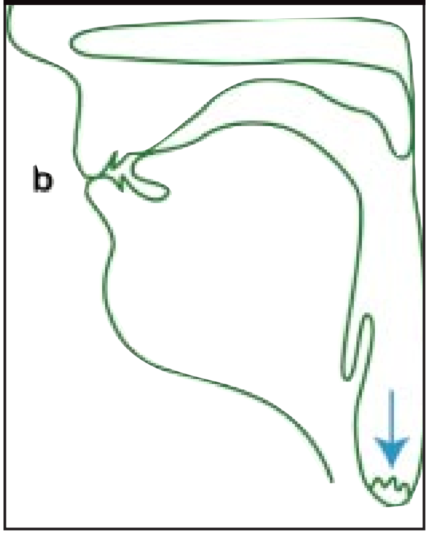


FIGURE 3b. *Side view of the sound [p], as in "pie."*

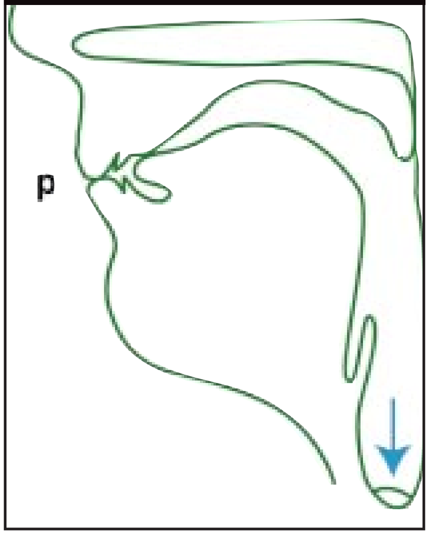
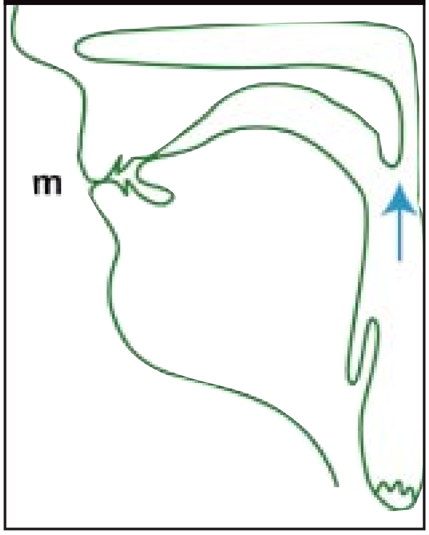


FIGURE 3c. *Side view of the sound [m], as in "my."*



With the information below, you will be equipped to know why things look wrong. Now for some group participation. This is an interactive article. Go on, no one is looking. The categories we want to examine are:

VOICED VS. VOICELESS. Put your hand on your throat and say something. You can feel an intermittent vibration. Now say, "**p**-at, **b**-at, **p**-at, **b**-at," (emphasizing the initial consonant). Looking at the face, there is no visual difference between voiced and voiceless sounds. In some sounds the vocal cords are vibrating together (**b**-voiced) and in some the vocal cords are apart (**p**-voiceless). This is an automatic no-brainer as far as reducing sounds into one viseme. Any pair of sounds that is only different because of voicing can be reduced to the same viseme. In English, that eliminates eight phonemes.

NASAL VS. ORAL. Put your fingers on your nose. Slowly say "momentary." You can feel your nose vibrating when you are saying the "m." Some sounds are said through the nasal cavity, but most are said through the oral cavity. These are also not visibly different. So again, we have an automatic reduction in phonemes. All three nasal sounds in English can be included in the oral viseme counterpart.

MANNERS OF SPEECH. Sounds can also be differentiated by the amount of opening through the oral tract. These also do not offer a visible clue, but are very important for categorizing phonemes. Sounds that have complete closure of the airstream are called stops. Sounds that have a partially obstructed closure and turbulent airflow are called fricatives. A sound that combines a stop/fricative is called an affricate. Sounds that have a narrowing of the vocal tract, but no turbulent airflow, are called approximates. And then there are sounds that have relatively no obstruction of the airflow; these are the vowels.

PLACES OF ARTICULATION. This involves where the sound is being made in the mouth. This is where the visible differences occur. There are several places of articulation (see Figure 2) involving the lips, teeth, tongue, and stuff in the back of the mouth (the palate, velum, and glottis) for the consonants. Vowel placement is based on the relative height of the tongue and whether the tongue is more front or back in the mouth. A differentiating factor not listed in Chart 1 is lip rounding. This is not associated with any particular place of articulation and will be addressed below. Whew.

As I said, there are 35 phonemes in my dialect of American English. You may have more. Chart 1 is a summary of these phonemes. Read the chart from the front of the mouth to the back of the mouth. Try saying each of the words that illustrate the phoneme that is in bold. Have a look in the mirror and see what is going on as well as feel what is going on inside the head. By using the distinction of voicing and oral/nasal, we have already eliminated 11 phonemes. Let's continue the reduction of phonemes into the usable visemes.

Take It to the Limit

According to the chart, there are three bilabials, which are sounds made with both lips. They are [b], [p], and [m]. According to the Figures 3a, 3b, and 3c they have different attributes inside the mouth. B and P only differ in that the B makes use of the vocal cords and P does not. The M sound is nasal and voiced so it is similar to the B sound, but it is a nasal sound. The cool thing about these sounds is that while there are differences inside the mouth, visually there is no difference. If you look in a mirror and say "buy," "pie," and "my" they all look identical. We have reduced three phonemes into one viseme.

While you're working, remember that you are thinking with respect to sounds (phonemes), not letters. In many cases a phoneme is made up of multiple letters. So, if we go through Chart 1, we can continue to reduce the 35 phonemes into 13 visemes. For the most part, the visemes are categorized along the lines of the Places of Articulation (with the exception of [r]).

Take a look at the sidebar on visemes. It describes the look of each phoneme in American English. The only phoneme not listed is [h]. "In English, 'h' acts like a consonant, but from an articulatory point of view it is simply the voiceless counterpart of the following vowel." (Ladefoged, 1982:33-4). In other words, treat [h] like the vowel that comes after it.

To see how helpful this information can be when animating a face take a word like "hack." It has four letters, three phonemes, and only two visemes (13 and 9 in the sidebar).

Say that you don't have enough space to include 13 visemes and whatever emotions you want expressed. Well, by using Chart 1 and the list of visemes in the sidebar, you can make logical decisions of where to cut. For example, if you only have room for 12 visemes, you can combine viseme 5 and 6 or 6 and 7 below. For 11 visemes, continue combining visemes by incorporating viseme 7 and 9 below. For 10, combine visemes 2 and 3. For 9, combine 8 with the new viseme 7/9. For 8, combine 11 and 13.

If I were really pressed for space, I could keep combining and drop this list down further. Most drastic would be three frames (Open, Closed, and Pursed as in boot) or even a simple two frames of lip flap open and closed. In this case you would just alternate between opened and closed once in a while. But that isn't very fun or realistic, is it?

Visemes

1. [p, b, m] - Closed lips.
2. [w] & [boot] - Pursed lips.
3. [r*] & [book] - Rounded open lips with corner of lips slightly puckered. If you look at Chart 1, [r] is made in the same place in the mouth as the sounds of #7 below. One of the attributes not denoted in the chart is lip rounding. If [r] is at the beginning of a word, then it fits here. Try saying "right" vs. "car."
4. [v] & [f] - Lower lip drawn up to upper teeth.
5. [thy] & [thigh] - Tongue between teeth, no gaps on sides.
6. [l] - Tip of tongue behind open teeth, gaps on sides.
7. [d,t,z,s,r*,n] - Relaxed mouth with mostly closed teeth with pinkness of tongue behind teeth (tip of tongue on ridge behind upper teeth).
8. [vision, shy, jive, chime] Slightly open mouth with mostly closed teeth and corners of lips slightly tightened.
9. [y, g, k, hang, uh-oh] - Slightly open mouth with mostly closed teeth.
10. [beat, bit] - Wide, slightly open mouth.
11. [bait, bet, but] - Neutral mouth with slightly parted teeth and slightly dropped jaw.
12. [boat] - very round lips, slight dropped jaw.
13. [bat, bought] - open mouth with very dropped jaw.

Art Issues

These viseme descriptions are enough to realistically represent speech. However, the use of individual visemes is more an artistic judgement then a hard rule. When speaking, people tend to slur phonemes together. They do not clearly articulate each phoneme all the time. Also, the look of a viseme can change depending on the visemes that surround it. For example, the Disney guidelines describe the use of a slightly different viseme for B, P, and M if they precede the ea sound as in beat.

This dependency on surrounding sounds is called co-articulation and makes viseme choice more complicated. This is one reason that the automatic phoneme recognition software in some packages doesn't always provide realistic results. Smooth interpolation between viseme keyframes can help, but this alone may not be good enough. In many cases, it requires an artistic judgement for which viseme really looks best. In computer animation, realistic looks are all that matter. So, when you work, put in the viseme that *looks* best.

Emphasis and exaggeration are also very important in animation. You may wish to punch up a sound by the use of a viseme to punctuate the animation. This emphasis along with the addition of secondary animation to express emotion is key to a believable sequence.

In addition to these viseme frames, you will want to have a neutral frame that you can use for pauses. In fast speech, you may not want to add the neutral frame between all words, but in general it gives good visual cues to sentence boundaries.

So What Do I Do with This Stuff?

So far, I have been discussing issues that only seem important to the artists working on the facial animation. If the only use of facial animation in your project is for pre-rendered cut scenes, this may be true. However, I believe facial animation will become an important aspect in real-time 3D rendering as we take character simulation to the next level. This requires a close relationship between the art assets and engine features. As a technical lead on a cutting-edge 3D project, you will be required to create the production pathway that the artists will use to create assets. You will be responsible for deciding how many visemes the engine can support and the manner in which the meshes must be created. Having a clear understanding of what goes into the creation of the assets will allow you to interface more effectively with those creating the assets.

However, even with the viseme count I am still not ready to set the artists loose creating my viseme frames. There are several basic engine decisions that I must make before modeling begins. Unfortunately, I will have to wait until next month to dig into that. Until then, think back on my 3D morphing column ("Mighty Morphing Mesh Machine," December 1998) as well as last year's skeletal deformation column ("Skin Them Bones," Graphic Content, May 1998) and see if you can get a jump on the rest of the class. ■

FOR FURTHER INFO

• Culhane, Shamus. *Animation from Script to Screen.* New York: St. Martin's Press, 1988.
• Ladefoged, Peter. *A Course in Phonetics.* San Diego: Harcourt Brace Jovanovich, 1982.
• Maestri, George. [digital] *Character Animation.* Indianapolis: New Riders Publishing, 1996.
• Parke, Frederic I. and Keith Waters. *Computer Facial Animation.* Wellesley: A. K. Peters, 1996.

Acknowledgements:

Special thanks go to my partner in crime, Margaret Pomeroy. She was able to explain to me what was really going on when I made all those funny faces in the mirror. When she was studying ancient languages in school I am sure she never imagined working on lip-syncing character dialog.

by Mel Guymon

To Build or Not to Build

That is the question that many development houses are pondering today. And with ever-increasing frequency, developers are turning to outsourcing for much of their digital content creation needs. Some of the biggest names in the industry have brokered deals where the bulk of their modeling needs are handled out-of-house, and the tactic is paying off in shorter time to market, and a smoother production pipeline.

This month, we'll take a close look at some of the major players in the market, and look at some examples of how their products have evolved alongside the gaming industry.

Why Are We Seeing the Shift?

There are three main reasons for the recent trend in outsourcing. First, the hardware we deal with has become extremely capable, and over the past few years we've seen a shift from 2D and 2.5 D games to fully 3D, immersive environments. This refocusing of artistic demand is necessitating a massive amount of 3D work to fill the environments of today's games. And it's not uncommon to have four or five modelers dedicated just to building environmental models and architecture. The second factor is the size of the talent pool of good 3D artists. There just aren't enough of us out there to meet the constantly increasing demand for quality 3D work. In many cases, as you will see shortly, developers have no option other than to outsource. Finally, the quality level of the outsourcing community has risen dramatically within the last few years. Modeling houses are no longer the black sheep of the digital artist community. Recent Hollywood exposure as well as big name entertainment titles have helped these teams to solidify their respective reputations for providing quality work in a timely manner. Outsourcing has become a more than viable option.

Who Are the Major Players?

Although there are myriad small companies dedicated to providing modeling services for both the film and entertainment industries, there are a select few who have risen to dominate the field.

VIEWPOINT DATALABS. Widely recognized as the industry leader in digital content creation, Viewpoint is fast becoming the Microsoft of modeling. Originally known for its extensive database of wireframe meshes, the company has been rapidly expanding its operations to include custom modeling and animation services, and very recently, the development of a texture database for 3D models. Viewpoint's client list reads like a who's who in film and video games. Most notably, the digital lizard in the recently released film *Godzilla* and the entire cast of GT's ODDWORLD: ABE'S ODDYSEE stand as impressive tributes to the team's custom modeling capabilities. And with the recent acquisition of the distribution rights for the REM Model Bank, Viewpoint's library of digital content remains unparalleled in the industry. Figure 1 is a highly detailed model of the the statue of Abraham Lincoln in the Lincoln Memorial, and demonstrates the high fidelity of artchitectural accuracy for which Viewpoint is well-known. Figure 2 shows a rhinoceros model from the REM Model Bank, the distribution rights to which Viewpoint now owns. Note the painstaking attention to detail in the texture work. This is what has set the Model Bank apart from its contemporaries.

ZYGOTE. Formed back in 1994 by former Viewpoint employees, this Utah-based company was founded by a group of industry professionals who have maintained their niche in the area of organic modeling. Although Zygote's offerings and experience don't match Viewpoint's, it is still a substantial player. Its clients have included some of the biggest names in the industry, from DreamWorks and Digital Domain to

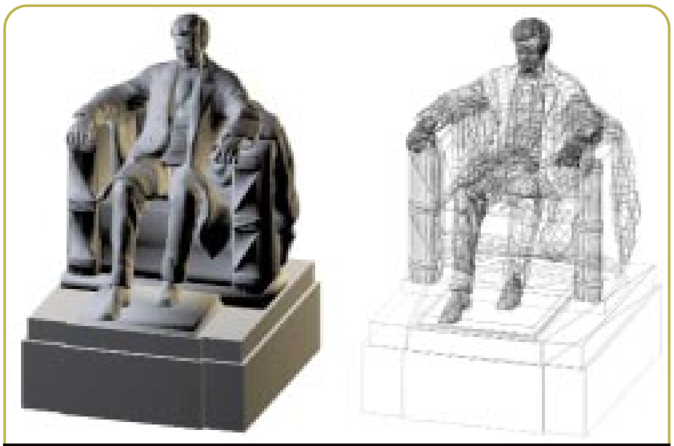


FIGURE 1. *This detailed model of Abe Lincoln demonstrates Viewpoint's high-fidelity architectural accuracy.*

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

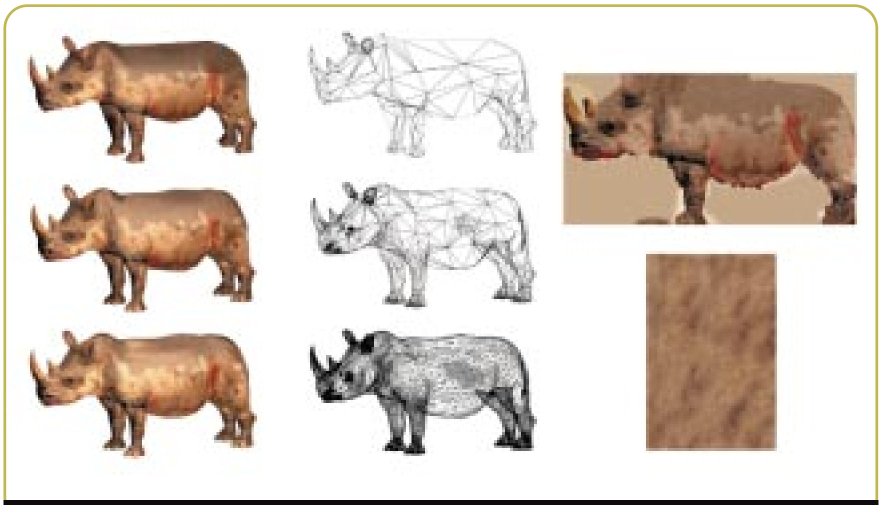


FIGURE 2. *The painstaking attention to detail in this Rhino is what sets Model Bank apart from its contemporaries.*

Accolade and Pixar. With its expertise in organic modeling techniques, it is probably the second-largest supplier of digital content in the industry.

Zygote recently developed of a line of fully textured and animatable humanoids. The models, created specifically for use with 3D Studio Max and Character Studio, come attached to a biped skeleton, which is weighted and ready for animation. An example of this is shown in Figure 3, in which a nicely modeled human is displayed with its corresponding wireframe and biped skeleton. Note the organic blending characteristics of NURBS models; the lack of texture work is made up for in the detailed nature of the geometry. Other relative newcomers to the scene include Paraform and 3D Café.

Weighing the Options

Whether to go with an outsourcing scheme is a decision to be made carefully, no matter what the situation. But it can prove to be the best and sometimes only option, depending on the circumstances. And you don't always have to depend on the stock digital libraries that Zygote and Viewpoint offer. Both have also demonstrated a robust capability for custom modeling in the gaming industry, which is where they get most of their exposure. To get a better feel for the custom modeling process, we interviewed some of Viewpoint's most recent clients, and found that, without exception, they viewed the entire experience as a positive one.

CASE STUDY 1: GAUNTLET: LEGENDS. This remake of the 1980s arcade hit was recently released by Atari. Steve Caterson, the project's art director, went to Viewpoint in the latter stages of production to get high-resolution versions of the in-game characters created. According to Steve, off-loading these tasks at the back end of the project enabled his team to focus all of their creative energies on completing the product. Atari got a lot for its outsourcing dollars as well, since the models were subsequently used for a number of marketing and promotional images, cabinet graphics, screens graphics, and full-motion videos.

For LEGENDS, Steve was able to work out a pipeline with the Viewpoint team that proved extremely efficient. Initially, concept sketches were sent off to Viewpoint's modeling teams, with any questions resolved via telephone or e-mail. Viewpoint's modelers began building the models and sending orthographic renders of the characters back to Steve, who then made changes to the images in Photoshop to indicate the direction he wanted the modeling team to take (Figure 4). This back-and-forth iteration continued through the texturing stage until the final models were completed (Figures 5 and 6). Eventually, as the relationship and communication evolved between Steve and the Viewpoint team, they were able to work out artistic changes via telephone only.

In retrospect, the cost of the process, although substantial, paid big returns in quality and speed. The Viewpoint team was able to get the content created much faster than the in-house team could have. And the Viewpoint team displayed a willingness to accept changes and rework and edit the models until the content was perfect. In Steve's words, "With other contractors I've worked with, when you ask for 'A' they give you just barely what you ask for, but no more. And when you ask for changes, they look at you like you're trying to kill their firstborn. With Viewpoint, the modelers and animators were totally enthusiastic about what they were doing. I had animators calling me up late at night to say they had stayed an extra few hours to add in cool features that we didn't even know we would need. With Viewpoint, I got what I asked for, and then a whole lot more."

CASE STUDY 2: CIVILIZATION: CALL TO POWER. This latest offering in the groundbreaking CIVILIZATION franchise is being published by Activision. CIVILIZATION: CALL TO POWER offers innovative features such as stacked combat and battle view. The game's time frame runs from 4000 BC to AD 3000, giving players the opportunity to build far into the future and enjoy new units, wonders, and technologies. But the increased functionality and game play came at a high development price. Each unit, city improvement, and civilization advance would be represented by a pre-rendered, spinning 3D object. Mark Lamia, the game's producer, was looking at more than ten man-years of modeling work just to create the 3D versions of these

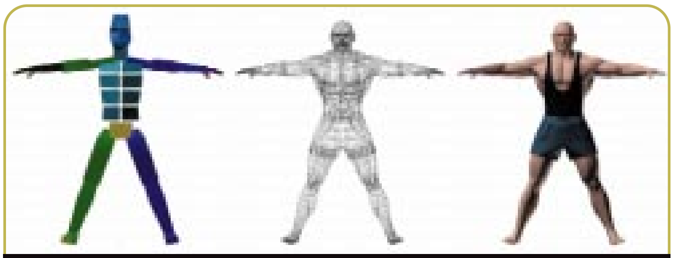


FIGURE 3. *An example of Zygote's animatable humanoids.*

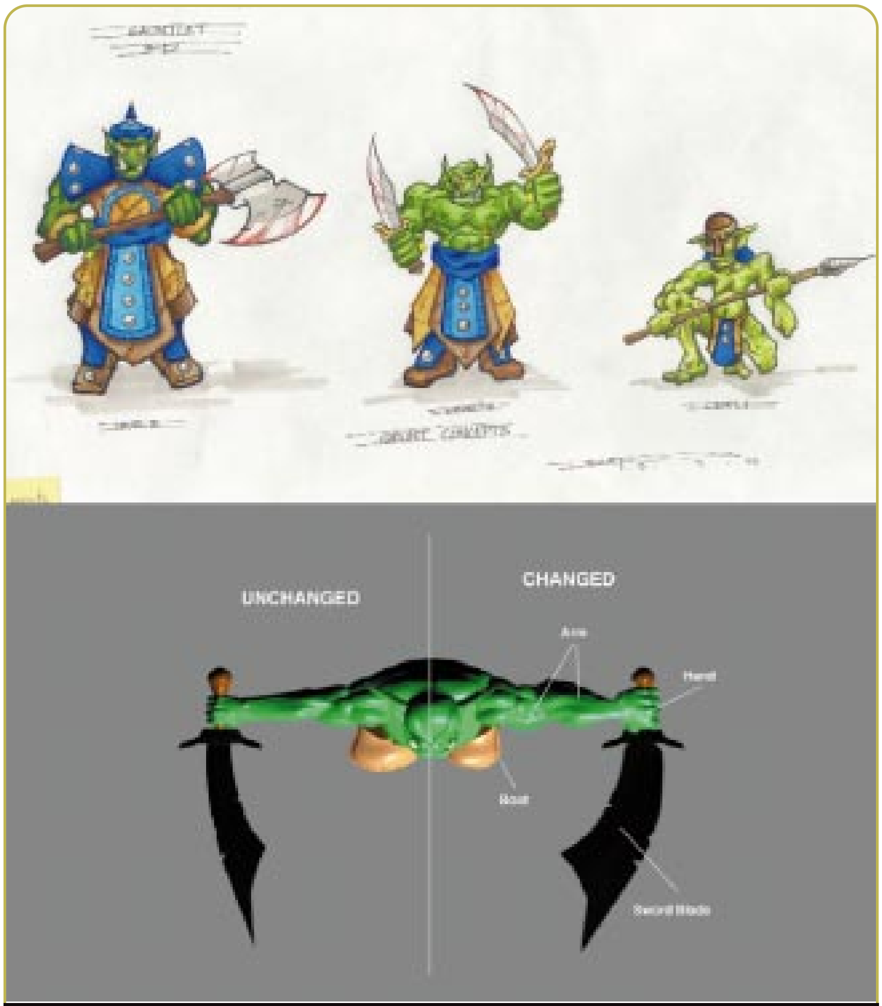


FIGURE 4. *The art director for Atari's* GAUNTLET: LEGENDS *used the Viewpoint team to turn concept pieces and Photoshop edits into highly detailed models.*

objects. From a budgetary standpoint, this task, if done in-house, would have cost the company well over $500,000. Mark needed a solution that was faster and cheaper than what he could do with his own team. Here again, outsourcing solved the problems that the in-house resources just couldn't handle. However, this was a much bigger problem than that faced by the Atari team. Instead of generating content for marketing publications, Viewpoint's team of modelers needed the resources to actually build the game. There was far less room for error — if the ploy failed, the product wouldn't ship. Faced with the challenge of getting all this artwork remotely produced under budget and on time, Mark worked closely with his point of contact at Viewpoint to organize the project around milestones which separated the hundreds of models into manageable chunks.

Aside from the sheer bulk of the work, the development process was very similar to Atari's situation. Starting with black-and-white concept pieces, and eventually moving to color comps, the Activision team put together the resource material necessary for Viewpoint to complete the task. This process turned out to have side benefits since, as Mark put it, "Having to direct someone remotely meant that the concept material had to be clear and concise, with no room for misinterpretation. It really forced us to be organized and to examine the different aspects of each object in extreme detail — something that, if we did the work in-house, we would probably have been more lax about doing." The project further benefited from the use of several stock models from the Viewpoint catalog. "Since the game takes place in large part over a real-world time frame," Mark added, "several of the architectural objects in the Viewpoint library were directly applicable to our project."

The Viewpoint team produced several hundred animating sprites, each one a rotating view of an in-game object, in a little under ten months. And it cost a fraction of the projected in-house strategy. The game was completed on time and under budget on the resource side of the house, and everyone benefited from the experience.

Tips to Remember When Outsourcing

In hindsight, both Mark and Steve had several recommendations for a team looking to contract work to an outside production facility.

1. Always speak your mind about

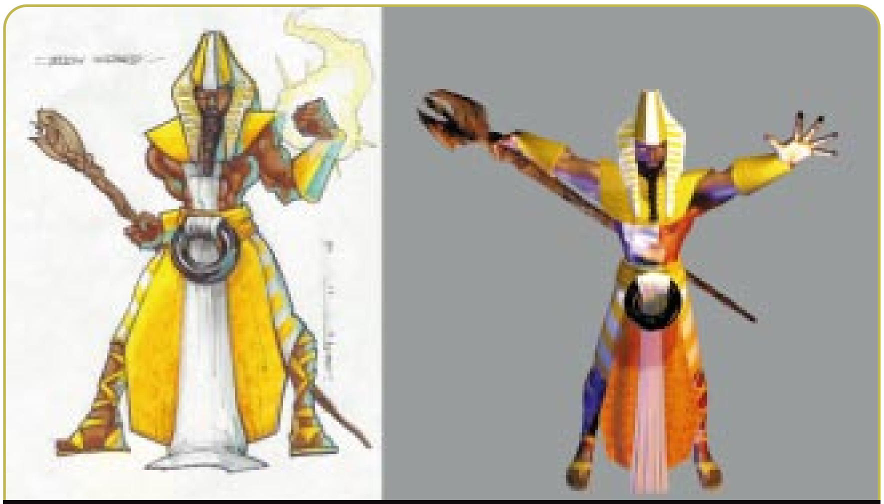


FIGURE 5. *A concept sketch and one of Viewpoint's finished* GAUNTLET *models.*

what you want. Don't mince words, and be tactful yet blunt. If you don't like what you see, the contractor probably won't fix the problem on his or her own. You are the one paying for the service, and as the customer, you are always right.

2. Prepare, prepare, prepare. It's always better to give the contracting team too much information than not enough. The level of ambiguity in your art direction is directly linked to the level of uncertainty in the finished product. If you don't know what you want, the contractor can't know either.

3. Engage the contractor in as much communication as possible. Agree on milestone points for each model (at least two, one at the modeling stage and one at the texturing stage, but the more the better), and plan plenty of time for iterations, especially at the beginning of the process. Once you have developed a rapport with your contractor, the process will go much faster and require less footwork on the front end.

4. For large projects, designate someone in-house as the asset manager, responsible for maintaining a database showing the status of models in the approval process. This person should be responsible for up- and downloads between you and the contractor, and should keep tabs on where each piece of content fits into the process. This can be an extremely time-consuming task, and should not be left to chance, as contractors may or may not take this responsibility upon themselves.

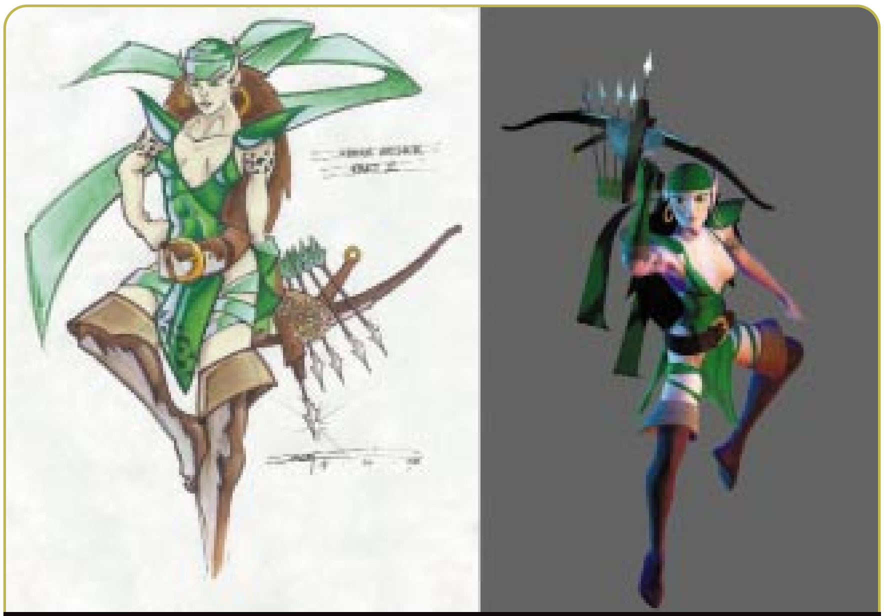


FIGURE 6. *Another concept sketch and finished model for GAUNTLET*

Redefining Real Time?

Researching this article, I came across very few instances where custom modeling was done for real-time 3D assets. The feedback I got seemed to indicate that while the high-resolution modeling and rendering were tasks which the art directors felt could be effectively managed out of house, the generation of low-polygon real-time content was something better left to the in-house teams. The reasons for this trend varied, but the underlying impression was that the real-time content could be developed rapidly in-house, and since real-time modeling techniques differ largely from the high resolution modeling methods used to generate cutscene characters, the teams felt more comfortable doing the work themselves.

Currently, a run-of-the-mill 3D engine will put out anywhere from 2,000 to 6,000 polygons on-screen at any given time. Consider that in a standard character-based game, about half of this polygon budget is spent on characters, meaning anywhere from 1,000 to 3,000 polygons in character models alone. There are projects currently in development with target on-screen polygon counts of 50,000 and higher. At a similar ratio, upwards of 25,000 polygons will be devoted to character models. This means that many of the high-resolution models generated by companies like Zygote and Viewpoint will have direct application for real-time 3D games, and the hardware is only getting better. It seems obvious then, that with the constantly increasing capability of gaming platforms, and the correspondingly complex scene requirements for 3D content, outsourcing will become a much larger aspect of game development. ■

3Dlabs: Pioneering 3D Isn't Always Profitable

Since 1993, 3Dlabs has been pioneering and evangelizing 3D graphics on the desktop. Even before it had a product offering for the mainstream desktop, as it was on the cusp of launching its workstation graphics product, it was active among game developers and in the general PC market.

In fact, 3Dlabs has always managed to maintain a high profile in the industry, luring Creative Labs, Intel, and Texas Instruments into its embrace, and at one point having practically every major graphics board vendor in the world dealing with it. Today, the company may not be so hot. It faces its biggest challenge yet, one that may make or break this pioneer of desktop 3D. That challenge is to win the hearts and minds of game players as well as the game development community.

A History Lesson

There is no doubt that 3Dlabs's pedigree is a good one. The company began life as Du Pont Pixel, a subsidiary of Du Pont, the large, well-known technology conglomerate. Du Pont Pixel was specializing in supplying GL-based 3D graphics tools and applications on SPARC, and was a founding member of Sun's Open Graphics Initiative, as well as one of the first licensees of OpenGL.

In March 1994, Du Pont Pixel's senior management took the assets that they needed and formed the nucleus of 3Dlabs using assets they had acquired from a previous company. The initial research which led to the development of the company's Glint technology actually began in 1993 as a small development effort within Du Pont Pixel. Management at 3Dlabs was eager to take this technology into the Wintel market, which was still considered a highly risky maneuver. In 1993, the idea that the PC was going to replace the workstation market was still heresy.

In late 1993 and early 1994, a small group of engineers within Du Pont Pixel undertook initial architectural design and product definition. The company substantially expanded this development effort and significantly increased the development team, culminating in the completion of the Glint 300SX processor in the fourth quarter of 1994. Glint got off to a rocky start. It was the first product to hit the PC workstation market that met the standards of application vendors, but it lacked simple features such as support for certain texture mapping functions. Despite misgivings about 3Dlabs's silicon, almost every graphics board vendor with a yen for the PC workstation market signed on with the company. By 1996, 3Dlabs owned 90 percent of the PC workstation graphics market.

In connection with the acquisition of assets from Du Pont, 3Dlabs agreed to pay Du Pont Pixel two percent of its total revenues each year that the company's total revenues exceed $70 million until the year 2000. Very shortly thereafter, on April 18, 1994, 3Dlabs's management completed the buyout of Du Pont Pixel from the Du Pont Corporation and entered into a license agreement with Creative Labs, under which Creative licensed from 3Dlabs certain Glint technology. For this license, Creative paid 3Dlabs an initial license fee of $1 million. In July 1995, Creative paid the company a nonrefundable advance royalty of an additional $1 million in connection with the development of certain products and reductions in future minimum royalties. Moreover, Creative agreed to pay minimum annual royalties through the year 2000.

In return, 3Dlabs deRaveloped the Gigi chipset for use in Creative's 3D Graphics Blaster. Creative also paid the company approximately $1 million in contract engineering fees for the development of Gigi. 3Dlabs retained exclusive rights to the intellectual property included in Gigi, and Creative had the exclusive right to sell Gigi until 1998 along with the right to sublicense. The term of the license was seven years.

At the time, many people thought Creative was going to sew up the 3D graphics market the way it had the audio market, but neither the technology nor the support of game developers was there. However, in 1995, Creative also licensed the Permedia technology from 3Dlabs in consideration for paying a portion of the Permedia development costs. In addition, 3Dlabs agreed to incorporate certain features into the Permedia chip specifically for Creative, and Creative cross-licensed certain technology to 3Dlabs, mostly on the API and driver front.

By the time 3Dlabs had its initial public offering in 1996, Creative and Intel both had shares in the company, which was a measure of the value each

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

(In thousands of dollars)	1995	1996	1997	1998
Revenues	6,594	19,695	69,129	42,721
Operating Expenses	7,992	16,904	52,712	68,547
Income Before Taxes	-1,358	3,091	17,882	-30,753

3Dlabs's revenues and profits for 1995–1998. How hard will they get hit?

company placed on the 3Dlabs technology. Intel wanted to see 3Dlabs's 3D efforts go into its platform, and 3Dlabs wanted to be tightly bound to the Intel platform in the way it had been to Sun's in its early days. Creative was to provide the company mass market exposure, but without the burden of building a consumer and mainstream graphics business. I don't think that at the time there were many people outside of the industry who didn't associate the PC 3D market exclusively with 3Dlabs. You have to give 3Dlabs a lot of credit for its evangelizing efforts. The company could so easily have chosen to focus on the high-end at the exclusion of all else, but it was ambitious. Or was it hubris?

The Need for Speed — Game Speed

Permedia should have opened up the doors for 3Dlabs in the mainstream. But it didn't hit the market in its first iteration, and missed out on performance and some essential features. By the time Permedia 2 came out, the consumer market was awash in 3dfx and the promise of Rendition. In fact, Permedia 2 became 3Dlabs's worst enemy, taking a chunk of its Glint board business at much lower costs. So, if you look at 3Dlabs in 1998 you find a company that cannibalized its own workstation graphics business with a lower-cost, lower-profit-margin product. In addition, the company's efforts to raise the performance bar on its high end with its next-generation Glint Gamma were thwarted by competitors Evans and Sutherland, Intergraph, and a small company called Dynamic Pictures. To make matters even worse, the company was not getting much mileage out of its technology in the mainstream, where S3, ATI, Matrox, Nvidia, and 3dfx were doing their thing, and where Intel was driving prices down with its chips.

When Evans and Sutherland bought one of its main customers, Accelgraphics, 3Dlabs acquired one of its competitors that was already a vertically integrated board vendor, Dynamic Pictures. In effect, the high-end graphics market did what it has always done for the industry: it trickled down technologies and business directions. You can trace the merger of 3dfx and STB back to the point when Evans and Sutherland acquired Accelgraphics, and caught the industry flat-footed. The chip and board vendors knew that the industry was going to consolidate with the coming of Intel, and the successful paths trodden by ATI and Matrox, but no one wanted to make the first move. Once the floodgates opened, however, everyone started making their moves. Of course, 3Dlabs's maneuver was part forward-looking and part defensive. The company was still determined to get into the mainstream graphics market, but could not see a way of doing so with a board business. Actually, the problem may have had more to do with the fact that 3Dlabs couldn't afford to buy a board vendor like Diamond Multimedia, and thereby retain its autonomy.

As a result, Permedia 3 now spearheads 3Dlabs's assault on the desktop. It launched at CeBIT, Europe's largest computer show, and simultaneously at the GDC. But it's very late — by six months or a year, depending on whom you talk to. And it may not be a match for Voodoo3, TNT 2, or even Savage4 in the market. In fact, 3Dlabs has an uphill battle to get Permedia 3 consumer recognition. Most board vendors are committed to taking TNT 2 into the high-end gaming segment, and beating up on 3dfx. S3 provides the vendors with a budget-priced chip, and a good one at that. Where does Permedia 3 fit if it's not significantly faster than TNT 2?

Permedia 3: There's the Rub

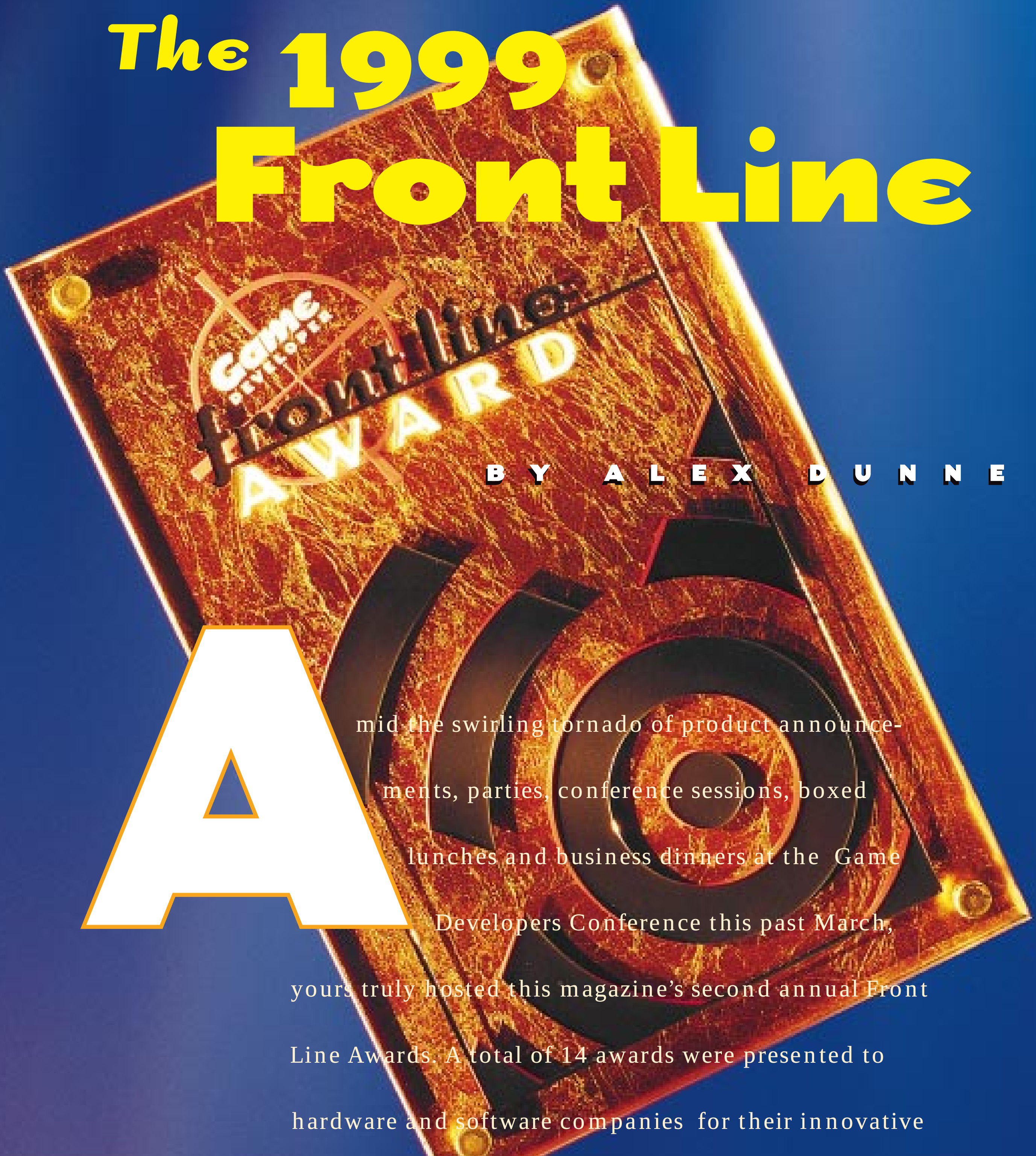
On the surface, Permedia 3 is not a bad chip. Whether it can take on the likes of Voodoo3 and TNT 2 is subject to some debate, but it's also not certain whether it deserves to be pigeonholed into the same market as Permedia 2. Ultimately for 3Dlabs, Permedia 3 must make some effective headway into the consumer space, and that means winning over the game players. Without this crucial demographic behind it, Permedia 3 loses any luster it might have, and 3Dlabs knows this. It doesn't help that 3Dlabs's fortunes have taken such a hit in the past eighteen months. The company is almost ripe for takeover, but partners for the company are few and far between. A good run in the market by Permedia 3 would change all that dramatically.

Permedia 3 has one unique selling point: virtual textures. It's worth noting that Matrox used the functionality of its bump mapping feature on the G400 in the same vein when they launched it at the GDC. Will it be enough, coupled with all of the other standard 3D features of the Permedia 3? I'm not sure that it will.

As of this writing, 3Dlabs didn't have any significant brand-name board vendors signed on for Permedia 3. Part of the reason, as I've said, is that the company doesn't want to see its flagship product slotted in between TNT 2 and Savage4. But 3Dlabs needs some glory, and quick. I don't believe the company will shuffle off its mortal coil anytime soon, but it will be subject to the slings and arrows of misfortune if it doesn't boost the sex appeal of the Permedia 3.

You can't count 3Dlabs out yet. It blazed a trail, and brazenly miscued on major product initiatives, yet it remains a strong 3D brand, and the company has the ability to muster up some impressive engineering resources. It also has some strengths in supporting game developers. It just needs sex appeal. ■

The 1999 Front Line



BY ALEX DUNNE

Amid the swirling tornado of product announcements, parties, conference sessions, boxed lunches and business dinners at the Game Developers Conference this past March, yours truly hosted this magazine's second annual Front Line Awards. A total of 14 awards were presented to hardware and software companies for their innovative contributions to the field of game development in 1998, and one product was inducted into the *Game Developer* Hall of Fame.

Photography by Carter Dow

The Front Line Awards are this publication's annual nod to the products and companies that made the lives of game developers easier. It's how we say thanks to the people and companies who make great game development products.

Admittedly, there were times over the past year when many of us judging products for these awards discussed the difficulty of pinpointing which game technologies were the most innovative. After all, the rate of change in

Awards

the industry is phenomenal, and it's not easy to proclaim specific products as "winners" in this regard. However, despite these inherent difficulties, we feel that the hardware and software on the following pages are worthy of praise and recognition.

Products Big and Little Alike

Years ago, William McGowan, then CEO of MCI Communications, was quoted as saying, "significant progress doesn't come from the formal planning process of an American corporation — it comes from a couple of guys doing something that hasn't been set down on a list." I can't think an industry that proves this sentiment more than game development. Much of the interesting work in the field of game development technology is being done by smaller or less well-known companies such as SN Systems, Darkling Simulations, and NxN Digital Entertainment. These companies understand the industry, see where gaps in technology are, and then create products that fill those voids. The combination of industry comprehension and pragmatic product development is powerful.

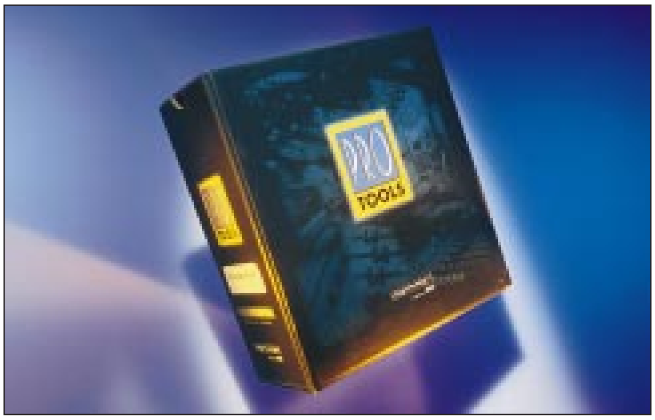
Of course, some of the stalwarts of the industry were recognized as well. The elegance of Maya earned an award for its creator, Alias|Wavefront. Diamond Multimedia and Creative Labs were recognized for their contributions in the realm of consumer graphics and audio hardware, respectively. And Digidesign's Pro Tools, the perennial favorite of sound designers and composers, was inducted into the *Game Developer* Hall of Fame.

A Multitude of Thanks

I'd be remiss if I didn't take this opportunity to offer heartfelt thanks to our distinguished panel of judges this year. These individuals, all of whom work in the industry, made these awards possible. We couldn't have done it without them. They are: Andrew Boyd, Chuck Carr, Jeanne Collins, Ron Fosner, Mel Guymon, Mark Haigh-Hutchinson, Tom Hays, Jim Hedges, Stefan Henry-Biskup, Rob Hubbard, Mike Kelleghan, Jeff Lander, Spencer Lindsay, Dominic Perricone, Wallace Poulter, Greg Pyros, Todd Siechen, Paul Steed, Dan Teven, Ben Waggoner, Rob Wyatt, and Halstead York.

Now onto the winners...

Hall of Fame Award



Pro Tools

Digidesign

Every so often, I come across an article about a game development team that did a hit on the cheap, and the team's sound person did all of the sound effects using a $200 sound card and a stereo sample editor. I then chuckle about how tough life was before I got set up with Digidesign's Pro Tools.

For those who are required to create large quantities of professional-quality audio for A-list games day in day out, it makes good economic sense to spend a few thousand dollars on some good gear. At the majority of large game development houses, not to mention film and TV sound design shops, this gear includes Pro Tools.

Pro Tools debuted as a successor to Sound Tools, a highly successful stereo hard disk recording system that used Sound Designer II as its software front end. Pro Tools added the ability to use four tracks (expandable to 16), as well as non-destructive editing features such as fades. For a few years, Pro Tools with Sound Designer II comprised a power pair, letting users easily open data in both programs, apply plug-ins, edit to the sample level, and do multi-track mixes with impeccable quality.

Game Developer front line HALL OF FAME 1999

Today's Pro Tools 24 Mix seamlessly scales from massively zooming in on a waveform to nail a click, out to a 64-track mixing session with gobs of in-line DSP power. It's great for building and revising complex game play effects. Pro Tools has earned its place in the Hall of Fame by setting the standard by which its competition is judged, and continuing to raise that standard each year since its introduction.

Pro Tools joins past Hall of Fame inductees Watcom C/C++, Borland C++, The Miles Sound System, Bounds-Checker, SoftICE, 3D Studio 1.0, Deluxe Paint, DeBabelizer, Sound Forge 1.0, Cool Edit 1.0, Cakewalk Pro for Windows, SoundBlaster 1.0, *The Mythical Man Month*, and *Computer Graphics: Principles and Practice.* *— Tom Hays*

Programming Libraries, APIs, Engines

NetImmerse

Numerical Design Ltd.

Rendering in 3D is hot right now. To get a head start, many companies are licensing game engines for their development. However, many engines are too general or too specific to a game genre and may not be well-suited to an individual project.

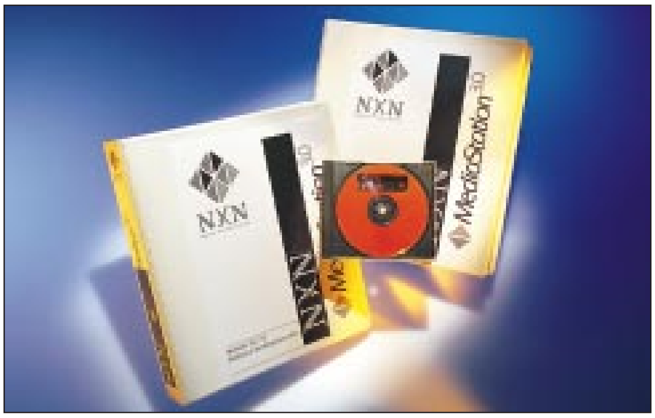
Numerical Design Ltd.'s NetImmerse is a true general rendering scene API. It takes up where pure rendering libraries such as Direct3D and OpenGL leave off. NetImmerse provides scene management, collision detection, animation support, and many other features that make rendering a game environment possible. Many such libraries could be so general that they would not be cutting-edge. However, NDL has a strong technical background and has built in many features that developers really want these days: particle systems, skeletal deformation of skinned characters, continuous LOD, and many others. There is also strong integration and support for 3D Studio Max.

The engine supports OpenGL, Glide, and Direct3D making it easy to integrate into most current 3D projects. Another wise choice NDL has made is to include full source code and well-documented API reference manuals to licensees. This allows developers to modify portions of the engine easily to their own project, as well as debug strange behavior. This open attitude is very welcome and gives NDL a clear advantage over other game libraries.

For the game development community at large, perhaps NetImmerse's greatest advantage is that it allows both new developers and those wishing to jump-start their 3D technology easy access to the cutting edge. — *Jeff Lander*

FINALISTS: Fastgraph 5.0 for Windows (Ted Gruber Software); Motivate (The Motion Factory); RTIME 3.0 (RTIME)

Programming Utilities



MediaStation 3.0.10

NxN Digital Entertainment

Bigger teams, longer development cycles, more content, more interdependencies: these trends make it ever harder for us to track the state of our projects. Kudos to NxN for recognizing an emerging product niche, and delivering a "complexity management" tool that's designed specifically for game developers.

It's hard to describe MediaStation adequately because it does so many things, and because it's readily customizable. It's a build environment. It's a version control system, optimized for media asset production. It's a multi-user project tracking database. It's a dessert topping.

The best thing about MediaStation is that NxN really understands the problems of modern game development. MediaStation gives your whole development effort structure without becoming intrusive. You can integrate your other tools into its IDE, or you can use the MediaStation SDK to give a custom tool (like a level editor) a more powerful back end. You can store meta-information with assets, which helps when you need to create localized versions, demos, and versions for multiple platforms from the same pool of files. And of course, you can ensure that everyone has access to the right version of a file at all times, that only authorized persons can modify files, and that the right files end up on the master CD.

The user interface is well thought out, the product is scrupulously documented, and it's easy to get up and running. It's hard for me to imagine a game development team that wouldn't benefit from using MediaStation — I've worked on several that would have killed for it. — *Dan Teven*

FINALISTS: Code Co-Op 2.0 (Reliable Software); IPEAK Graphics Performance Toolkit (Intel); Observation Architecture (Intel); VTune 3.0 (Intel)

Programming Environments

SN64 PRO-DG 1,0,0,4

SN Systems

Sometimes, the best innovations come from a system's interface. As a whole the SN64 PRO-DG system is great, and now by adding a visual interface, SN Systems has really simplified the developer's working process. The interface's visual approach allows you to drag and drop items, simplifies setting up project files, removes some of the intricacies of makefiles, and makes it easier to add files to existing projects, along with many other benefits. This is especially helpful to new developers unfamiliar with the make process.

Despite these simplifications, the interface remains customizable enough for experienced programmers. Built-in scripting, macros, and redefinable keyboard shortcuts allow programmers to extend its functionality. Another great aspect of the SN64 PRO-DG is its ability to update directly over the Internet. The text editor is also one of the features which makes the SN64 such a truly integrated environment — you really never need to leave the program.

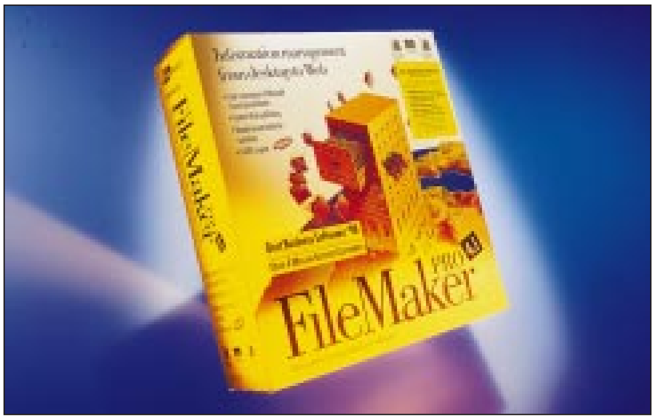
Although there was no manual, the in-program help was informative. The system's approach is general enough to support any future target platform, and for this it's to be commended (although this also means it can't handle some of the more subtle aspects of a specific development environment). It costs a little more than other development systems, but its feature set exceeds that of all of its competitors.

In short, the SN64 PRO-DG is pretty much essential if you're doing N64 development. I suspect other console development might be going forward, so keep an eye on SN Systems.

— *Mark Haigh-Hutchinson*

FINALISTS: CodeWarrior Pro 4.0 (Metrowerks); Intel C/C++ Compiler 3.0 (Intel); Visual C++ 6.0 (Microsoft)

Production Utilities



FileMaker Pro 4.1

FileMaker

Producing games is hard work. There are so many things to track: graphics, sound files, bugs, hours, and more. Can game developers find one low-cost database package to do it all? FileMaker Inc. (formerly Claris) has provided the answer with FileMaker Pro 4.1. It comes complete with tools to develop entry screens, reports, and even the ability to publish them on the web.

The interface is so intuitive and online help is so extensive, most computer literate people can develop a usable database structure without ever reading the manual. What is particularly amazing is how many of FileMaker's built-in features could only be achieved by writing extensive scripts with other database products, such as the ability to set up a screen where data entry or modification of each field is dependent upon the user's login name.

Every area of production has something they need to track in a database. With relative ease, each group can easily build their own data tracking tools to suit their needs and publish that data to the company intranet to share with other departments. Since game products vary greatly, standard bug reporting systems are rarely robust enough to handle games. FileMaker is the answer. A bug tracking tool and reporting system can be built on-the-fly in less than a day, and can last the whole project long.

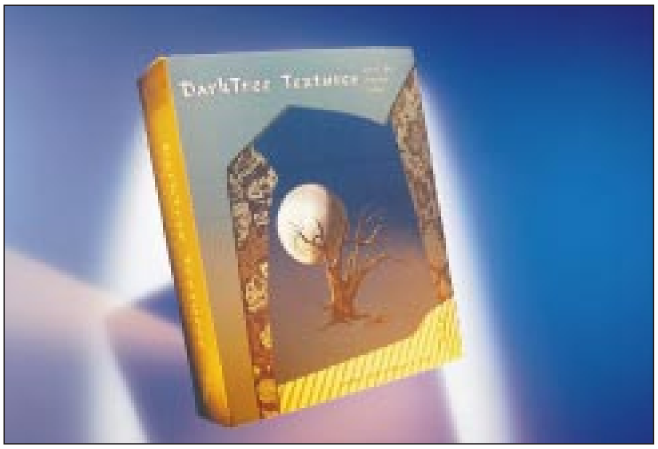
FileMaker works for all game production users because it is truly cross-platform compatible. Network-stored databases can be accessed by both Macintosh and Windows clients. FileMaker Inc. also offers exceptional multi-license discounts.

— *Jeanne Collins*

FINALISTS: MSDN (Microsoft); Norton Ghost (Symantec); Project 98 (Microsoft)

Game Developer front line AWARDS 1999

Image Editing & Manipulation



DarkTree Textures

Darkling Simulations

DarkTree Textures takes the power of procedural textures and lays it out in an easy-to-use and highly flexible format. Procedural shaders have always been a very useful tool for rendered images, providing subtle control and endless variation for surfaces. DarkTree also provides tools to make textures that do tile, a big plus for games use.

The layout of the tree construction is easy to handle. All of the modules arranged on the left of the screen can be dragged out to the schematic-like construction area. The range of the modules provided greatly exceeds that found in most 3D packages. Many of the modules provide unique methods for combining the other elements. The ability to see the shader plainly not only makes the tool easy to use but also makes the reusability of complex trees as the basis for others much more feasible. Your tree is never static. If you decide you want to add a new module in between some already laid out, it's easy to move components around. A wide array of options allows the user to visualize the tree in useful ways, and a large selection of useful samples is also included. The package's library tool is very handy for keeping track of your creations.

Dragging files directly to the renderer, the ease of naming files, and support for batch rendering all make the product user-friendly and well thought out. There's even a plug-in version available for 3D Studio Max, Animation Master, and Lightwave.

— Stefan Henry-Biskup

FINALISTS: **KnockOut 1.0 (Ultimatte); Painter 5.5 (MetaCreations); Photoshop 5.0 (Adobe)**

3D Modeling & Animation Environments

Maya

Alias|Wavefront

I have used many different 3D packages over the years, but few have had the effect on me that Maya has in such a short time. Upon first glance at the Maya toolset and its capabilities, it's easy to feel overwhelmed at the variety and complexity of tools at your disposal, but I have yet to see another application deliver such a wide range of functionality in such a usable package.

The first thing that becomes apparent in Maya is its innovative object manipulation paradigm. After only a few moments with it, I wondered why every 3D program doesn't use the same or a similar technique. The ability to work easily in a single perspective view while maintaining the control to rotate, scale, and transform in any locked axis without using a hot key is wonderful.

Maya also has a fantastic customizable interface that is easily tunable to the individual artist's needs. These simple revolutionary features will save you a great deal of production time. Maya still has a few rough edges, but the features I have struggled with in other programs such as inverse kinematics, constraints, skinning, deformations, and NURBS are all very easy to use and implement with Maya — and they work.

It should be noted that all this great functionality comes to us in Maya's infant first version. Version 2.0 promises many refinements, improvements, and increased usability to an already strong package. All these wonderful features combined with the product's recent price decrease make Maya the clear winner.

— Todd Siechen

FINALISTS: **3D Studio MAX 2.5 (Kinetix); Lightwave 3D 5.6 (NewTek); Rhinoceros (Robert McNeel & Associates); Softimage|3D 3.8 SP1 (Softimage)**

3D Modeling & Animation Plug-Ins

Surface Tools

Digimation

As a digital artist, I've been modeling for years in polygons and have tried many different ways of manipulating them in a fluid and organic way. I've tried patch modeling, NURBS packages, and lofting. All of these methods have their advantages, but Surface Tools gives me the most freedom to work with for the price.

With Surface Tools, Digimation offers an excellent modification to the existing patch topology tools already in 3D Studio Max. The once unattainable (or at least very difficult) organic surface is now a breeze to create and modify. By creating Bézier splines to define the basic shape and form of a model, a modeler can quickly generate very tunable patch sets that can then be converted to a number of different resolutions. It's like modeling with just the edges of patches. The learning curve is relatively shallow, and once you get the hang of it, Surface Tools is very easy to move around in. Depending on how you set your work up, the quick sketch feel of Surface Tools allows you to make a lot of iterations while always letting you go back to a good old version. I have used Surface Tools extensively in modeling everything from cars to mountains.

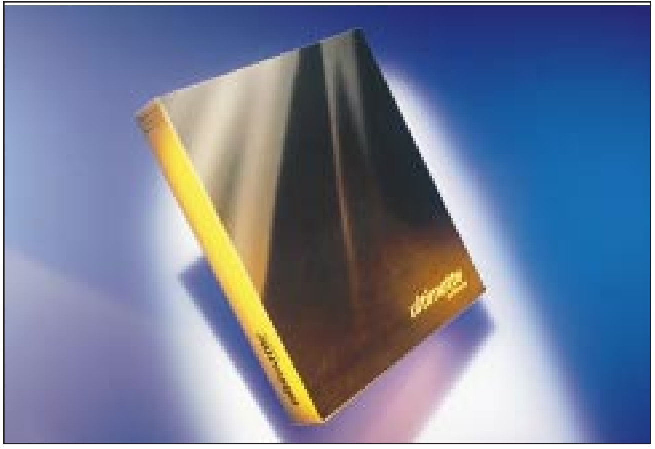
While I was at Atari, my team was responsible for generating the new cars for RUSH 2049. I had the art team use Surface Tools because we could get excellent control over the smooth surfaces. We were also able to generate both high-resolution models for cutscenes and then just crank down the resolution for the real-time versions.

All in all, Surface Tools is one of the coolest tools in my arsenal.

— *Spencer Lindsay*

FINALISTS: Character Studio 2 (Kintetix); MetaReyes 4.0 (REM Infográfica); ProMotion (LambSoft); Surface Suite (Sven Technologies)

Video Editing & Compositing



Ultimatte Software

Ultimatte Corp.

The Ultimatte chroma-compositing workflow has existed in the post-production arena for many years. Indeed, Ultimatte blue and green are colors recognized throughout the industry. With the Blue ICE-accelerated version of their After Effects plug-in, Ultimatte brings the power and speed of a multi-million dollar post-production environment into the reach of small, independent shops.

Game Developer Front Line Awards 1999

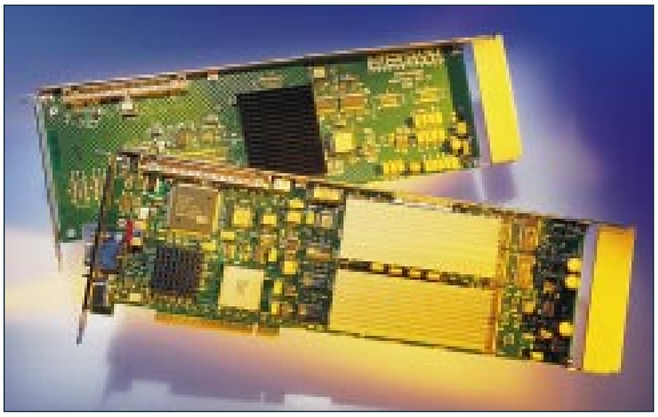
The Ultimatte plug-in's interface is extremely simple and intuitive for trained After Effects users. The manual is extremely informative, but even without it everything is just where you think it should be. The more advanced features are easy to access for advanced users, and easy to hide for beginners. Absolutely beautiful chroma-keying is only a few mouse-clicks away. Matte clean-up is more difficult, but even a newcomer to compositing can easily get results far better than they could with just about any other tool on the market, no matter what the price.

Ultimatte has delivered the holy grail: an extremely useful and rock-solid piece of software that extends the capabilities and professionalism of anyone who uses it. It's not so much a revolutionary product, but ICE'd Ultimatte is robust, thoughtful, and mature. If you need to get professional quality compositing done today, get yourself After Effects, a Blue ICE card, and the Ultimatte plug-in. It offers that greatest of gifts — the chance to go home early and do it with a clear conscience.

— *Halstead York*

FINALISTS: Automasker AE2.0 (Automedia); effect* (Discreet Logic); ICEfx2 for After Effects (Integrated Computing Engines); MediaStudio Pro 5.2 (Ulead Systems)

Professional Graphics Cards



Wildcat 4000

Intergraph

The Wildcat 4000 is a great scalable video card for high-end graphic systems. Up to four cards and a geometry accelerator can be used in a single machine. If required, each card can drive a separate monitor or work in unison with the others. With four cards, the memory limit is 256MB of frame buffer and 1GB of texture memory. For the first time on the PC, the whole OpenGL pipeline is implemented in hardware. If your application uses OpenGL's advanced features, you'll be pleased to know that anti-aliasing, stencil buffers, overlay buffers and OpenGL accumulation buffers are all fully supported in hardware. Extensions include volumetric fog, volumetric lighting and 3D textures.

The card has a full complement of drivers, supporting OpenGL, RenderGL, DirectDraw and Direct3D in Windows NT and 95. I ran the GL conformance tests which all passed, and then tried some real applications (including 3D Studio Max and Maya), and they all performed flawlessly. QUAKE 2 also ran with no problems.

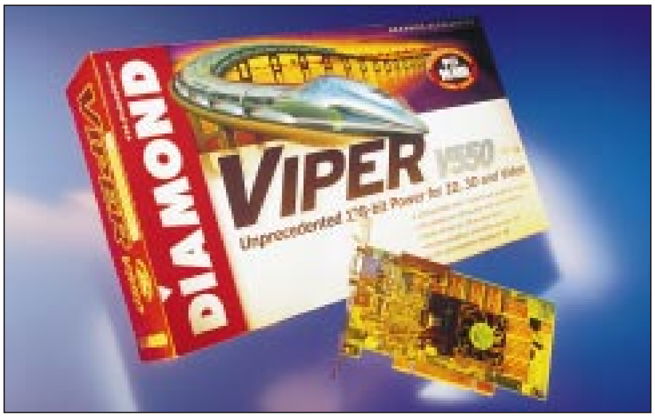
The downsides are its maximum resolution of 1280×1024 and the texture fill rate of 90 million pixels with all features enabled — which was apparent when playing QUAKE at maximum resolution. The fill rate may not be an issue for the you, however, as most things are modeled and animated in wire-frame or flat shaded; geometry processing is far more important.

User control is supported via an applet to set up multiple cards and monitors. Intergraph usually provides regular driver updates and their tech support is good. The base price for the 48MB version (16MB frame buffer and 32MB of textures) is $2995, which is significantly cheaper than the competition.

— Rob Wyatt

FINALISTS: Gloria-XXL (ELSA); Oxygen GMX (3Dlabs); TITAN II (MaxVision)

Consumer Graphics Cards



Viper V550

Diamond Multimedia Systems

Apparently Diamond has some great karma. Just as its once-close ally, 3dfx, shifted strategies and became Diamond's arch nemesis, Nvidia showed up on Diamond's doorstep with the Riva TNT. How's that for timing?

The Riva TNT chip was the biggest advancement in consumer 3D hardware last year, beating out the old champ, Voodoo2. Combined with the fantastic TNT is Diamond's know-how in the card business. Simply put, Diamond knows how to take chips and turn them into excellent consumer solutions. Last year we recognized Diamond for its fine work with its Monster 3D card, and it earned the Front Line Award again this year for the Viper V550. Available in both AGP 2X and PCI flavors, Diamond's Viper V550 features a long list of features that made for great game playing last year: full screen anti-aliasing, a triangle set-up engine, anisotropic filtering, bump mapping, a 24-bit Z-buffer, an 8-bit stencil buffer, MPEG, TV output, and 16MB of 125MHz SDRAM. The 2D acceleration is excellent as well, which means no sacrifices once you jump out from game land to take care of some business.

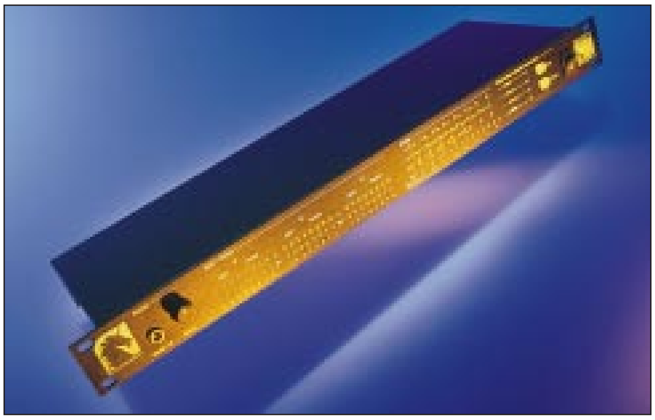
I've always had smooth setup experiences with Diamond hardware, and the Viper V550 is no exception. The 3D control panel and the desktop utilities that come with the card are great.

The card outperformed other TNT-based cards in synthetic benchmarks like Winbench 98 and 3D Winbench 98, and certainly held its own in real-world tests too. It was definitely a hardware highlight in 1998, and I look forward to what Diamond and Nvidia will bring us this year with the forthcoming TNT2-based Viper V770.

— Alex Dunne

FINALISTS: All-In-Wonder Pro AGP (ATI Technologies); Millennium G200 (Matrox Graphics); Monster 3DII (Diamond Multimedia Systems); Terminator BEAST SuperCharged (Hercules Computer Technology)

Professional Sound Hardware



MOTU 2408

Mark of the Unicorn

The Mark of the Unicorn (MOTU) 2408 hard disk recording system includes a PCI card, a single rack-space I/O unit, a 12-foot cable, software drivers for both Windows and MacOS, and a complete audio workstation software package for MacOS called Audio Desk. It has eight analog inputs/outputs with 20-bit converters, 24 channels of ADAT optical input/output, 24 channels of TDIF input/output and stereo S/PDIF in/out (with an extra stereo out). It provides word clock, ADAT Sync and Digital Timepiece Control Track sync, which achieves sample-accurate digital transfers between digital tape decks and the computer.

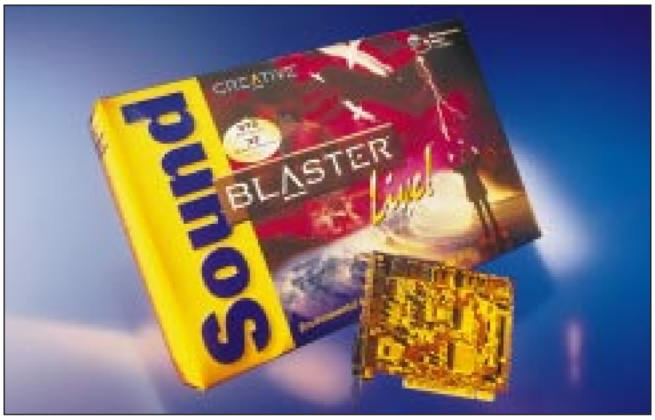
The MOTU 2408 supports stand-alone format conversion so you can transfer any format (ADAT, TDIF, or analog) to any other, up to 24 channels at a time. The 24-bit internal data path of the MOTU 2408 supports 24-bit recording via 24-bit hardware (mixer, preamp, FX processor, or other device) connected digitally. So what's the bottom line? It sounds great! I'm using it with my 400MHz PC and a Panasonic DA7 with two ADAT cards connected to the 2408 and getting 16 channels of pristine digital audio. I'm also using the analog outs of the 2408 going to the first eight channels of the DA7. All of my software applications such as Cakewalk Pro 8, Acid, Sound Forge 4.5, and Cool Edit Pro 1.1 support it. My only complaint is that it doesn't have Windows NT drivers yet.

With its excellent sound quality, features, and list price of $999, the 2408 is a solid choice for people using PCs or Macintoshes for professional audio applications.

— Chuck Carr

FINALISTS: DSP Factory (Yamaha); Layla (Event Electronics); Paris (Ensoniq); Pro Tools|24 MIX (Digidesign)

Consumer Sound Cards



Sound Blaster Live!

Creative Labs

Creative Labs's Sound Blaster Live! is the definitive sound card for game players. Notable features include 32 channels of DirectSound3D hardware acceleration, downloadable sounds (DLS) and environmental audio extensions (EAX) for reverb and echo effects. The card uses PCI bus mastering and the advanced EMU10K1, a 1000MIPS digital signal processor, which delivers audiophile-quality sound.

For games using DirectX, the card performs very well, with very little impact on things such as frame rate and graphics. This is very important, as some other cards take up a lot of the host CPU when doing DirectSound3D.

The card boasts a very good feature set considering the low cost ($99 for the value card). It supports multi-speaker output as well as EAX, which adds different reverbs and echoes according to the location or environment that the game is in. This can be very effective in first-person games such as HALF-LIFE or UNREAL. The standard card ($199) has digital output (SPDIF) and a daughter card containing standard MIDI ports.

The installation is simple, the card is reliable, and it comes with a good manual. The card also comes with an extremely useful software bundle, including a good audio customization utility. The speaker configuration application is the most useful, allowing the user to select two or four speakers or headphones.

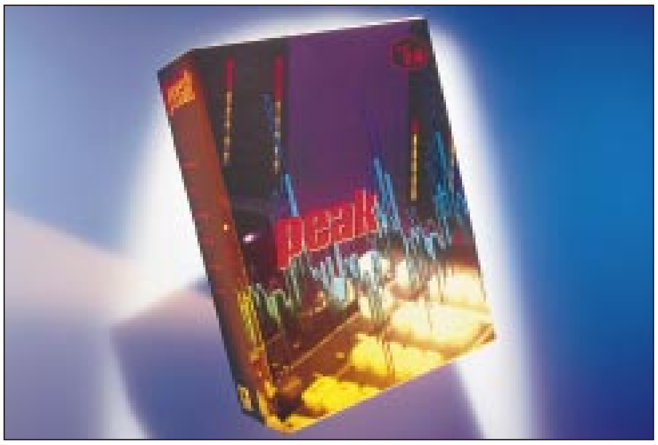
Creative plans on future driver enhancements for the Sound Blaster Live!, so it can support over 70 voices of DirectSound3D and new features such as AC-3 Dolby Digital decoding and eight-speaker support.

— Rob Hubbard

FINALISTS: EMU8710 PS (E-Mu Systems); Monster Sound MMX300 (Diamond Multimedia Systems); Waveforce 192 Digital (Yamaha)

Game Developer front line AWARDS 1999

Audio Recording & Editing



Peak 2.0

Berkeley Integrated Audio Software

The Macintosh has long been considered the standard platform for professional audio development. With products such as Pro Tools (this year's Hall of Fame inductee) as well as excellent sequencers and librarians, robust MIDI and audio support, QuickTime, and so on, Macintosh has been the platform of choice for anyone serious about audio production. Yet in all this, basic two-track editing seemed to get lost. Innovation was absent from the slow, unstable applications available. File format support was inconsistent, while interfaces were unattractive and unintuitive.

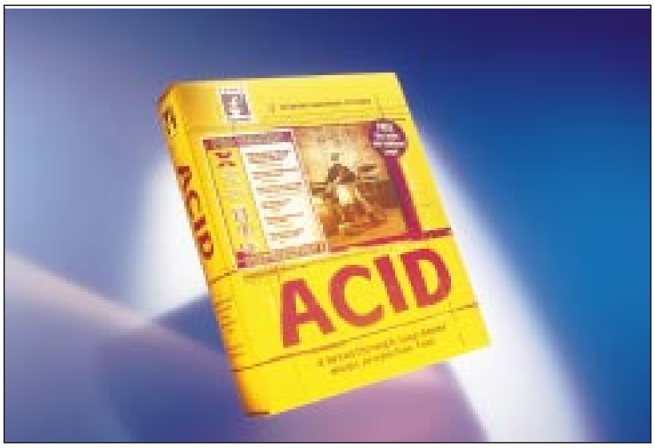
Well, there's finally a really good, general-purpose two-track audio editor on the Macintosh. Peak 2.0 is miles ahead of its predecessors in terms of features and interface, and significantly better than its other Macintosh-based competitors, too. It's easy to use, reasonably quick, sounds great, and integrates with surprising ease into a professional audio production system.

It has all the right features: non-destructive editing, a long list of supported formats, batch processing, integrated QuickTime video support, good-sounding standard DSP functions, and sampler support. Peak offers some really cool additions, too: fun effects such as "convolve" and "phase vocoder," handy real-time application of file-based Premiere plug-ins, and best of all, the ability to handle TDM and Audiosuite plug-ins.

But what really makes Peak stand out is the fact that all this quality and functionality are now available on the Macintosh, able to live on the same machine with all the other great tools you're probably already using. — *Andrew Boyd*

FINALISTS: Pro Tools 4.3 (Digidesign); SonicWORX (Prosoniq Products Software); Sound Forge 4.5 (Sonic Foundry)

Audio Composition



ACID PRO

Sonic Foundry

Acid is arguably the most innovative digital audio software to come along in years. If you use digital audio loops, you want Acid. It's that simple.

Unlike other programs that can change a loop's tempo by changing its pitch, Acid keeps the original pitch when changing tempos. Acid does an almost magical job when speeding up or slowing down tempos, and it works best when speeding up a loop from its original tempo.

Although you can record individual tracks with Acid, I liken songwriting with Acid to drum pattern creation on a drum machine. Create the parts of a song you're working on (like your drums, guitar parts, and bass lines) as two-,four- or eight-bar loops. Then add the loops in Acid to build the song. It's also great for experimenting with different arrangements by changing the chorus and verses to different places in a tune.

Doing remixes with Acid is amazing. If you want to mix your tune and you have a multi-track audio card or hardware box, Acid lets you assign your tracks to whatever output you want. If you want to add some real-time effects or compression to your tracks, go ahead — with the support of DirectX plug-ins it's a no-brainer. Sonic Foundry's Virtual MIDI Router (VMR) is the perfect "missing link" for synching Acid with a software sequencer.

I use Cakewalk for all my MIDI and non-loop-based audio and Acid for the loop-based digital audio. Perhaps this sounds like a dream scenario, but believe me, the dream is real. — *Chuck Carr*

FINALISTS: Cubase VST/24 4.0 (Steinberg); Logic Audio (Emagic); Pro Audio Deluxe 8 (Cakewalk); Studio Vision 4.1 (Opcode Systems)

Back

Im leme i g C ed S face Ge me

By Brian Sharp

Remember QUAKE? Back when it was first released, consumer-level 3D acceleration was nearly unheard of, and id's software renderer scaled in speed with the clock speed of your Pentium processor.

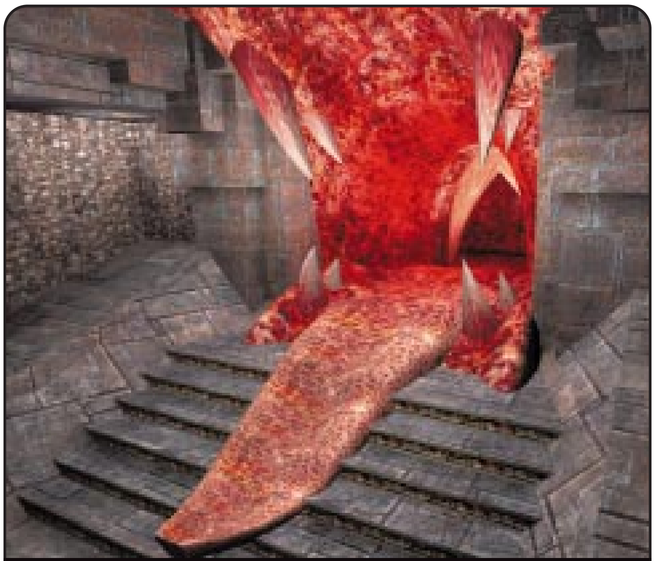
During the few years since then, though, our market has reached a point of stratification with non-accelerated Pentium "Classic" machines on the low end and the latest and greatest pixel crunchers on the high end. The range is enormous. As game developers, it's important to support high-end consumers, and yet we'd prefer not to abandon the low-end players. From this desire was a new industry trend born: scalable geometry.

Scalable Geometry

Scalable geometry is any kind of geometry that can be adapted to run either faster with decreased visual quality or slower with increased visual quality. There are a number of ways of doing this, so we'll briefly cover the more popular methods.

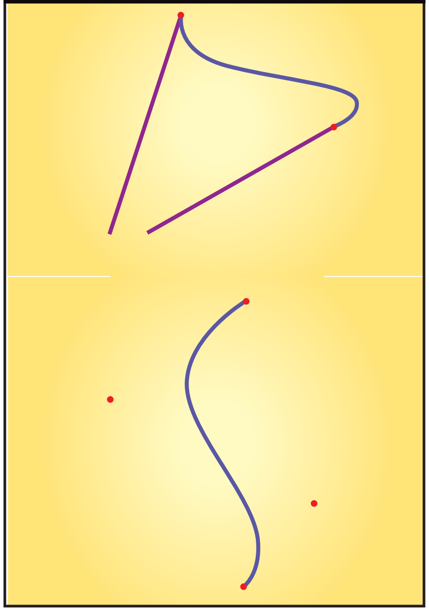
One of the earliest methods used in games to scale geometry involved hand-generated level-of-detail models. You can see this principle at work in games like BATTLEZONE and GRAND PRIX LEGENDS. In the case of a race car, artists create a very high-detail model of the car, then a lower-detail model, and then continue down to a very low-detail model. Then, at run time, factors like the speed of the machine and the distance of the car to the viewer determine which model you use each frame. One of the benefits that hand-tuned LOD models have over other approaches is that the models can have more actual polygonal detail at the higher levels, since they're created by hand. There are many drawbacks, though. For instance, the switch from one model to another can manifest itself as an abrupt visual "popping", and can therefore be distracting to the viewer. A solution to this is to increase the number of LOD models, but this exposes another drawback: it takes a lot of an artist's time to make several versions of every object.

Another method of implementing scalable geometry is in using dynamic mesh reduction techniques. Shiny's upcoming title, MESSIAH, reportedly uses a technique like this for their character animation system. The idea here is that you store one high-detail ver-


QUAKE 3: ARENA is one of the first games to take advantage of curved surfaces in a real-time 3D setting.

When Brian's not coding for CogniToy or sleeping through classes, he's writing up crazy stuff like this. Keep him busy by writing to him at brian_sharp@cognitoy.com with questions or comments, or else he might find the time to write again. Don't say we didn't warn you.

FIGURES 1 AND 2. *The top figure is a Hermite curve. Tangent vectors are magenta, endpoints are red, and the curve itself is blue. The second figure is a cubic Bézier curve. Its control points are red, and the curve is blue.*



sion of a model. Then, based on the distance of the model from the viewer and the desired framerate, you use some kind of detail reduction algorithm to generate an appropriate mesh. There are a number of ways to perform the detail reduction; if the high-detail model is stored as a polygon mesh, algorithms like the quadric error metrics described by Garland and Heckbert are perfect for the task. The advantages to dynamic mesh reduction are that many of the techniques can be very fast with some precalculation, and it can produce very good-looking results. There are, again, a number of drawbacks. It can be tricky to get texture coordinates to reduce with the mesh without making the texture slide around on the model. Also, algorithmically reducing a model from 10,000 polygons to 50 polygons still generally won't look as good as a 50-polygon model hand-crafted by an artist.

Curved Surfaces

The final method we'll mention, then, is the topic of this article. Curved surfaces are one of the most popular ways of implementing scalable geometry. There is a good reason for that, too; in games we've seen them in, they look fantastic. UNREAL's characters looked smooth whether they were a hundred yards away, or coming down on top of you. QUAKE 3: ARENA screenshots show organic levels with stunning smooth, curved walls and tubes. There are a number of benefits to using curved surfaces. Implementations can be very fast, and the space required to store the curved surfaces is generally much smaller than the space required to store either a number of LOD models or a very high-detail model.

The downside of curves and curved surfaces is that they are perhaps the most difficult of the three methods to learn and understand. There's a lot of reference material out there, but a lot of it is not easy reading, even if you know the material and are just using the books for reference. Therefore, in this article, we'll look at the basics of curves and curved surfaces. We'll cover the concept of the basic polynomial curve, and then onto two example curve representations: Hermite curves and Bézier curves. From there, we'll move onto surfaces, covering the Bézier patch. In this article, we'll take the most straightforward approach possible to rendering the curves and patches. While this does mean that our implementations will be very slow, they will hopefully be more legible for it. Next month, we'll continue our examination of patches by delving into optimization techniques to make them truly useful.

Remedial Curve Concepts

Just to be absolutely sure we all start off on the same wavelength, we'll start by reviewing some of the basic math principles that we need as a foundation for working with curves and curved surfaces. Feel free to skip this section if this is remedial.

At its core, any of the curves we'll discuss can be represented as a parametric polynomial function. Following convention, we'll use the parameter *u*. Our curves will look something like this:

$$\begin{aligned}
x(u) &= c_0u^3 + c_1u^2 + c_2u + c_3\\
y(u) &= c_4u^3 + c_5u^2 + c_6u + c_7\\
z(u) &= c_8u^3 + c_9u^2 + c_{10}u + c_{11}\\
f(u) &= (x(u), y(u), z(u))
\end{aligned}$$

Generally, we'll refer to *f*(*u*), which is the 3D point on the curve at *u*. Now, as long as at least one of c_0, c_4, and c_8 are non-zero, the curve will be a cubic curve, and cubic curves are the ones we're most interested in. After all, since we'd like to keep computation to a minimum, we'd like to use the lowest-degree curve possible (since a higher degree requires more multiplication every time it's evaluated). So, we might try using a zero-degree curve (which would be fast to compute). But a zero-degree curve is simply a point, which doesn't do us too much good.

Moving on, a one-dimensional curve is simply a line. It's pretty clear that lines are insufficient for our purposes. So, we move on to quadratic curves. These are parabolas, which might seem sufficient for representing curves and curved surfaces. Unfortunately, second-degree curves will always lie in a plane, and we're working in three dimensions, so it would be better to have a space curve, a curve that isn't confined to two dimensions or less. Therefore, our cubic curve is the curve of choice.

Representations

So all a programmer needs to do is code up a quick tool for the artists consisting of a view window and four text entry boxes for them to type in the coefficients of curves, right? Of course not — artists demand flexibile, intuitive tools, and it's clear that creating curves by typing in coefficients lacks that certain ease-of-use factor for most of us. Therefore, we need another representation for the curve that makes creation and manipulation more intuitive. We'll touch on two such representations, the Hermite curve and Bézier curve.

The Hermite curve we cover both because it's fairly common, but also because it doesn't require any specialized formulae to understand it. Then, as Bézier curves are somewhat more versatile, we'll move on to them. While we won't discuss it here, converting from Bézier curves to Hermite curves and vice versa is very straightforward and is explained in the references at the end of the article.

There are plenty of other curve representations that we aren't going to touch upon. Notably, we are not going to cover B-Splines or that family (including the pervasive NURBS), of which Bézier curves are simply a special case. I chose the Hermite and Bézier curve models as

LISTING 1. *Code to generate a cubic parametric equation from a Hermite curve's endpoints and tangents.*

```
void genCubicFunction()
{
    // Do this so we can treat each endpoint and tangent vector as a separate array.
    float* p0 = points;
    float* p1 = points + 3;
    float* v0 = tangents;
    float* v1 = tangents + 3;
    // Do this so we can treat each vector coefficient of the function as a separate array.
    float* a = functionCoeffs;
    float* b = functionCoeffs + 3;
    float* c = functionCoeffs + 6;
    float* d = functionCoeffs + 9;
    // Now, generate each coefficient from the endpoints, tangents, and the predefined
    // basis functions.
    // Note that we loop once each for the x, y, and z components of the vector function.
    for (int lcv = 0; lcv < 3; lcv++)
    {
        // a = 2p0 - 2p1 + v0 + v1
        a[ lcv ] = (p0[ lcv ] + p0[ lcv ]) - (p1[ lcv ] + p1[ lcv ]) + v0[ lcv ] +
                  v1[ lcv ];
        // b = -3p0 + 3p1 - 2v0 - v1
        b[ lcv ] = - (p0[ lcv ] + p0[ lcv ] + p0[ lcv ]) + (p1[ lcv ] + p1[ lcv ] +
                   p1[ lcv ]) - (v0[ lcv ] + v0[ lcv ]) - v1[ lcv ];
        // c = v0
        c[ lcv ] = v0[ lcv ];
        // d = p0
        d[ lcv ] = p0[ lcv ];
    }
}
```

a good starting point, because they can be represented and understood with a fair degree of ease. Once you have a firm grasp on Bézier curves, picking up one of the references at the end of this article and learning more about other curve models is much easier.

Hermite Curves

A Hermite curve is a cubic curve described by its endpoints p_0 and p_1 and the tangent vectors at the endpoints, v_0 and v_1. You can see in Figure 1 what an example curve looks like.

The question, then, is how we get the cubic equation from the points and vectors. Hermite curves are nice this way, as the derivation of the cubic is possible with just a little calculus. Let's say our cubic equation is

$$f(u) = au^3 + bu^2 + cu + d$$

Note that $f(u)$, a, b, c, and d are vectors. Then, we have that:

$$f'(u) = 3au^2 + 2bu + c$$

Then, we can express the endpoints as $f(0)$ and $f(1)$, and the tangents as $f'(0)$ and $f'(1)$.

$$\begin{aligned} p_0 &= f(0) = d \\ p_1 &= f(1) = a + b + c + d \\ v_0 &= f'(0) = c \\ v_1 &= f'(1) = 3a + 2b + c \end{aligned}$$

Solving now for a, b, c, and d, we get:

$$\begin{aligned} a &= 2p_0 - 2p_1 + v_0 + v_1 \\ b &= -3p_0 + 3p_1 - 2v_0 - v_1 \\ c &= v_0 \\ d &= p_0 \end{aligned} \qquad \text{Eq. 1}$$

Then we'll solve for what are called the "basis functions." The basis functions are simply functions of u that determine the contribution of the endpoints and tangents along the curve. So, for instance, the basis function that corresponds to p_0 determines how much p_0 contributes to points along the curve. Just by rearranging terms once again, we have the basis functions.

$$\begin{aligned} B_0 &= 2u^3 - 3u^2 + 1 \\ B_1 &= -2u^3 + 3u^2 \\ B_2 &= u^3 - 2u^2 + u \\ B_3 &= u^3 - u \end{aligned}$$

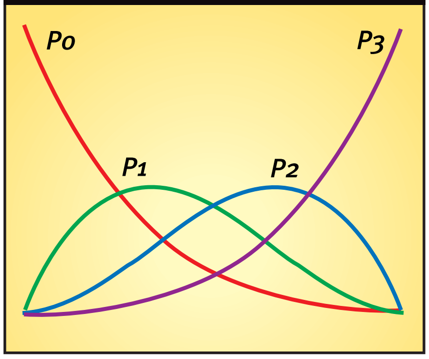
Then, we can express the curve as the sum of the basis functions times the components:

$$f(u) = B_0 p_0 + B_1 p_1 + B_2 v_0 + B_3 v_1$$

This provides us a handy way of expressing the curve. Furthermore, basis functions become far more important when we discuss Bézier curves, and so the Hermite curve provides a good introduction to the idea of a basis function.

So, as we see here, a basis function is nothing more than a function associated with a component of the curve that

FIGURE 3. *Bézier basis functions for a cubic Bézier curve. Each function is associated with a control point.*



LISTING 2. *Code to evaluate a cubic parametric equation at a given value of* u.

```
// This function simply computes au^3 + bu^2 + cu + d for a
// specific u and stores the vector result in out.
void evaluateAt(float u, float* out)
{
    // Do this so we can treat each vector coefficient of the function as a separate array.
    float* a = functionCoeffs;
    float* b = functionCoeffs + 3;
    float* c = functionCoeffs + 6;
    float* d = functionCoeffs + 9;

    // Note that we use Horner's rule for computing polynomials (which is the way
    // we nest the multiplies and adds to minimize the computation we need.)
    out[ 0 ] = ( ( ( a[ 0 ] ) * u + b[ 0 ] ) * u + c[ 0 ] ) * u + d[ 0 ];
    out[ 1 ] = ( ( ( a[ 1 ] ) * u + b[ 1 ] ) * u + c[ 1 ] ) * u + d[ 1 ];
    out[ 2 ] = ( ( ( a[ 2 ] ) * u + b[ 2 ] ) * u + c[ 2 ] ) * u + d[ 2 ];
}
```

determines the contribution of that component to points along the curve.

Implementing Hermite Curves

As handy as the basis functions are for expressing the curve, it's easier for our naïve implementation just to calculate the cubic equation of the curve by finding the coefficients using Equation 1. The code that does this is shown in Listing 1.

Then, we just run along the curve by starting u at 0 and incrementing it by some fixed amount until we reach 1. We evaluate the curve at each value of u, save each result as a point on the curve, and then render the curve as a line strip. The code to evaluate the curve at a given value of u is quite simple and is shown in Listing 2.

It's worth noting that even though the curve is recalculated fairly slowly every frame, the frame rate is still in the high hundreds (well, on my Voodoo2 graphics card, at least). Since we're doing nothing but calculating a hundred or so points along a curve every frame, the speed hit as a result of this inefficiency is not yet apparent.

Bézier Curves

Now that we understand the cubic Hermite curve and its implementation, we can move on to Bézier curves. Whereas a Hermite curve is defined by endpoints and tangents, a cubic Bézier curve is simply described by four ordered control points, p_0, p_1, p_2, and p_3. Figure 2 shows an example curve.

Our problem now is that it's not immediately clear how we define the curve based on these four points. With Hermite curves, we could use some basic calculus to get a cubic parametric equation. But even if we say that p_0 and p_3 are the endpoints, the points p_1 and p_2 seem to have little bearing, analytically, on the curve. It's easy enough to say that the curve should "bend towards" the points, but what does that give us in terms of our cubic equation? Here's where the importance of our basis functions comes in. We need to find a set of functions that blend the control points together in ways that give us the curve that we want.

To do that, of course, we need to define the properties we'd like the curve to have. We can summarize these with three qualities:

1. We'd like the curve to interpolate the endpoints. That is, we'd like the curve to start at p_0 and end at p_3. That makes curve creation more intuitive.

2. We'd like the control points to have local control. That is, we'd like the curve near a control point to move when we move that control point, but have the rest of the curve not move as much. Again, this gives us better intuitive control when crafting a curve.

3. We'd like the curve to stay within the convex hull of the control points so we can cull against it quickly if we're doing visibility culling or hit testing.

Luckily for us, there exists just such a set of functions. These functions are called the Bernstein basis functions, and are defined as follows:

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i} \quad \text{for } 0 \le i \le n$$

The parenthesized block with the n and the i is the mathematical phrasing of the binomial coefficient normally phrased "n choose i" or "n nCr i". The formula for n choose i is:

$$\frac{n!}{i!(n-i)!}$$

If we were considering general Bézier curves, we'd have to calculate that. Since we're only considering cubic curves, though, $n = 3$, and i is in the range [0,3]. Then, we further note that n choose i is the ith element of the nth row of Pascal's triangle, and so we have our values, {1,3,3,1}. So we can just hard-code that, no computation necessary.

While the Bernstein basis functions are a little odd looking at first, they're not that bad. To show that they really do satisfy our three conditions, we refer to Figure 3. This is a graph of the cubic basis functions. The red curve corresponds to p_0, the green to p_1, the blue to p_2, and the magenta to p_3. They're pretty

LISTING 3. *Code that tessellates and renders a Bézier curve as an evenly-spaced series of line segments.*

```
void UniformCurveTessellator::tessellate( const std::vector< CurvePoint >& controls,
                                          const std::vector< BasisFunction >& bases ) const
{
  // We break the curve into 100 even increments of u.
  int numSteps = 100;

  // We can multiply by this in our loop instead of dividing by (numSteps-1) every time.
  double invTotalSteps = 1.0f / (numSteps - 1);

  // Start drawing our curve.
  ::glBegin( GL_LINE_STRIP );

  for ( int step = 0; step < numSteps; step++ )
  {
    // Generate the parameter for this step of the curve.
    float u = step * invTotalSteps;

    // This holds the point we're working on as we add control points' contributions to it.
    float curPt[ 3 ] = { 0, 0, 0 };

    // Generate a point on the curve for this step.
    for ( int pt = 0; pt <= 3 ; pt++ )
    {
      // Get the value of this basis function at the current parameter value.
      float basisVal = bases[ pt ]( u );

      // Add this control point's contribution onto the current point.
      curPt[ 0 ] += controls[ pt ].getX() * basisVal;
      curPt[ 1 ] += controls[ pt ].getY() * basisVal;
      curPt[ 2 ] += controls[ pt ].getZ() * basisVal;
    }

    // Draw this point.
    ::glVertex3fv( curPt );
  }

  ::glEnd();
}
```

looking, but what do they mean? Recall that the basis functions determine the contribution of each point over the curve. Also, the horizontal axis on that graph is *u*. So, when *u* is zero, the value of the basis function for p_0 is 1, and all the others are 0. Therefore, the starting point of the curve is:

$$1p_0 + 0p_1 + 0p_2 + 0p_3 = p_0$$

Therefore, we have that the curve starts at p_0. Looking at the basis functions, it's obvious that the curve ends on p_3. Our first condition is satisfied.

As for the local control, we can convince ourselves that this holds by staring at the basis functions for long enough. It's obvious that p_0 and p_3 have local control, because as we move them, the curve moves, and they have very little influence over the rest of the curve. We can also see, then, that p_1 and p_2 have local control, since they have the most influence over the curve 1/3 of the way and 2/3 of the way along the curve, respectively. That means that if we moved p_1, it would pull the section 1/3 of the way along the curve with it, and affect the rest of the curve much less.

Then, we have our final condition: the curve must remain within the convex hull of the control points. With the Bernstein basis functions, this is true. The proof, however, is fairly complicated, and ends up dragging a bevy of new concepts into the fray. For the interested, Farin does a reasonable job of explaining this. It has to do with the fact that the Bernstein basis functions are nonnegative for *u* in the range [0,1], and also that if you sum up the values of all the basis functions for any value of *u*, the result is always 1.

Then, the formula for calculating a point on our Bézier curve is:

$$\sum_{i=o}^{3} p_i B_i^3(u) \qquad \text{Eq. 2}$$

Implementing Bézier Curves

Our approach to rendering a Bézier curve is similar to that for rendering Hermite curves. We find a series of points along the curve, and render that series as a line strip. We'll do it, once again, by evaluating the curve at even intervals of *u*. Listing 3 shows this clearly: `UniformCurveTessellator::tessellate` takes a vector of four control points and a

LISTING 4. *Code that tessellates and renders a Bézier patch as an evenly-spaced grid of triangles.*

```
void UniformPatchTessellator::tessellate( const std::vector< std::vector< CurvePoint > >&
controls, const std::vector< BasisFunction >& bases ) const
{
  // We break the patch into a numSteps x numSteps array of points.
  int numSteps = 10;
  // First, we need to make the basis functions for our normals, which just involves
  // taking the derivative of each basis function.
  std::vector< BasisFunction > bases_deriv( bases );
  for ( int i = 0; i <= 3; i++ )
  {
    bases_deriv[ i ].differentiate();
  }
  // Now we generate the points and normals.
  float* verts = new float[ 3 * numSteps * numSteps ];
  float* norms = new float[ 3 * numSteps * numSteps ];
  double invTotalSteps = 1.0f / (numSteps - 1);
  for ( int stepU = 0; stepU < numSteps; stepU++ )
  {
    // Generate the parameter for this step of the curve.
    float u = stepU * invTotalSteps;
    for ( int stepV = 0; stepV < numSteps; stepV++ )
    {
      // Generate the parameter for this step of the curve.
      float v = stepV * invTotalSteps;
      // This holds the point we´re working on as we add control points´ contributions to
      //it.
      float curPt[ 3 ] = { 0, 0, 0 };
      float curNorm[ 3 ] = { 0, 0, 0 };
      float curUTan[ 3 ] = { 0, 0, 0 };
      float curVTan[ 3 ] = { 0, 0, 0 };
      // Generate a point on the curve for this step.
      for ( i = 0; i <= 3; i++ )
      {
        for ( j = 0; j <= 3; j++ )
        {
          // Get a few basis function values and products thereof that we´ll need.
          float bu = bases[ i ]( u );
          float bv = bases[ j ]( v );
          float dbu = bases_deriv[ i ]( u );
          float dbv = bases_deriv[ j ]( v );
          float bu_bv = bu * bv;
          float bu_dbv = bu * dbv;
          float dbu_bv = dbu * bv;
          // Add this control point´s contribution onto the current point.
          curPt[ 0 ] += controls[ i ][ j ].getX() * bu_bv;
          curPt[ 1 ] += controls[ i ][ j ].getY() * bu_bv;
          curPt[ 2 ] += controls[ i ][ j ].getZ() * bu_bv;
          // Add this point´s contribution to our u-tangent.
          curUTan[ 0 ] += controls[ i ][ j ].getX() * dbu_bv;
          curUTan[ 1 ] += controls[ i ][ j ].getY() * dbu_bv;
          curUTan[ 2 ] += controls[ i ][ j ].getZ() * dbu_bv;
          // Add this point´s contribution to our v-tangent.
          curVTan[ 0 ] += controls[ i ][ j ].getX() * bu_dbv;
          curVTan[ 1 ] += controls[ i ][ j ].getY() * bu_dbv;
          curVTan[ 2 ] += controls[ i ][ j ].getZ() * bu_dbv;
        }
      }
      // Now get our normal as the cross-product of the u and v tangents.
      curNorm[ 0 ] = curUTan[ 1 ] * curVTan[ 2 ] - curUTan[ 2 ] * curVTan[ 1 ];
      curNorm[ 1 ] = curUTan[ 2 ] * curVTan[ 0 ] - curUTan[ 0 ] * curVTan[ 2 ];
      curNorm[ 2 ] = curUTan[ 0 ] * curVTan[ 1 ] - curUTan[ 1 ] * curVTan[ 0 ];
      // Normalize our normal (ouch!)
      float rInv = 1.0f / sqrt( curNorm[ 0 ] * curNorm[ 0 ] + curNorm[ 1 ] * curNorm[ 1 ] +
      curNorm[ 2 ] * curNorm[ 2 ] );
      curNorm[ 0 ] *= rInv;
      curNorm[ 1 ] *= rInv;
      curNorm[ 2 ] *= rInv;
```

Continued on page 52.

LISTING 4. *(Continued from page 50.)*

```
      // Store these.
      memcpy( verts + ( stepU + ( stepV * numSteps ) ) * 3, curPt, 3 * sizeof(float) );
      memcpy( norms + ( stepU + ( stepV * numSteps ) ) * 3, curNorm, 3 * sizeof(float) );
    }
  }
  // We render each strip of the surface out as a triangle strip.
  for ( int stepV = 0; stepV < numSteps - 1; stepV++ )
  {
    int y0 = stepV;
    int y1 = stepV + 1;
    ::glBegin( GL_TRIANGLE_STRIP );
    for ( int stepU = 0; stepU < numSteps; stepU++ )
    {
      int x0 = stepU;
      glNormal3fv( norms + ( x0 + ( y0 * numSteps ) ) * 3 );
      glVertex3fv( verts + ( x0 + ( y0 * numSteps ) ) * 3 );
      glNormal3fv( norms + ( x0 + ( y1 * numSteps ) ) * 3 );
      glVertex3fv( verts + ( x0 + ( y1 * numSteps ) ) * 3 );
    }
    ::glEnd();
  }
  // Clean up after ourselves.
  delete[] verts;
  delete[] norms;
}
```

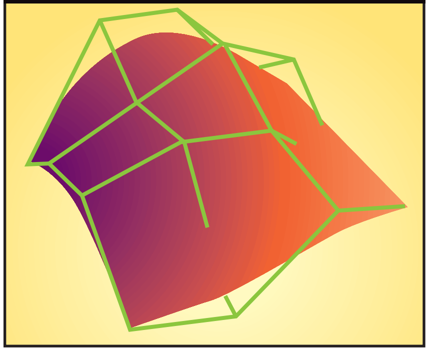
vector of four associated basis functions, and renders the curve in 100 steps.

To generate each point, it calculates Equation 2 for the input — it adds up the sum of each point times that point's basis function. For our cubic curve, this is certainly not the most optimized way to calculate the curve. However, because it's only 100 points, it's not noticeable and the demo still runs quite fast.

Surfaces: The Bézier Patch

It might seem more consistent to cover not only Bézier patches but also Hermite patches, as well. The reason we're skipping straight to Bézier patches is that we're trying to cover the curves and curved surfaces in the most intuitive order possible. Whereas it makes sense to cover Hermite curves and then Bézier curves, Hermite patches are somewhat more difficult to learn than Bézier patches.

FIGURE 4. *A bicubic Bézier patch. The green grid connects the control points.*



Since a Bézier curve was a function of one variable, $f(u)$, it's logical that a surface would be a function of two variables, $f(u,v)$. Following that logic, since a Bézier curve had a one-dimensional array of control points, it makes sense that a patch would have a two-dimensional array of control points. We'll now discuss bicubic Bézier patches. The phrase "bicubic" simply means that the surface is a cubic function in two variables — it is cubic along u and also along v. Then, since our cubic Bézier curve had a 1×4 array of control points, our bicubic Bézier patch has a 4×4 array of control points. Figure 4 shows an example of a surface.

Now, with that, we need to take our equation for evaluating a Bézier curve at some u and extend it to allow us to evaluate our patch at some (u,v). The extension is fairly straightforward. We just evaluate the influence of each of the 16 control points, yielding:

$$\sum_{i=0}^{3}\sum_{j=0}^{3} p_{ij} B_i^3(u) B_j^3(v) \qquad \text{Eq. 3}$$

We can see by inspection that our properties from the Bézier curve extend to the patches. Why? For the following reasons.

1. The patch interpolates p_{00}, p_{03}, p_{30}, and p_{33} as endpoints.

2. Control points have local control, that is, moving a point over the center of the patch will most strongly affect the surface near that point.

3. The patch remains within the convex hull of its control points.

Implementing Bézier Patches

Rendering a Bézier patch is more complicated than rendering a Bézier curve, even when doing it in the simplest possible way. With a Bézier curve, we could just evaluate a number of points and render a line strip. With a patch, we need to evaluate strips of points and render triangle strips. Also, with a patch we have to worry about lighting. After all, an unlit patch will just look like an oddly-shaped splotch of red on the screen. To see the contours, we need lighting. For our naïve implementation, that means we'll need to light each vertex. To light a vertex, we need its normal. So, for every (u,v) pair, we need to solve for the point on the surface, and then solve for its normal.

Equation 3 tells us how to find the point on the surface, but how do we find the normal? Well, we know we can take the derivative of the surface with respect to either u or v, which would yield the tangent vectors to the surface in the direction of either u or v, respectively. If we find both of those tangents, we know that they both lie in the plane tangent to the surface. Then, taking their cross product will yield a mutually perpendicular vector, the surface normal. Finally, we'll have to normalize it since it most likely won't be unit length.

So, how do we find $df(u,v)/du$ and $df(u,v)/dv$? As it turns out, we can just take the derivatives of the basis functions. That is,

$$\frac{df(u)}{du} =$$

$$\frac{d\left(\sum_{i=0}^{3}\sum_{j=0}^{3} p_{ij} B_i^3(u) B_j^3(v)\right)}{du} =$$

$$\sum_{i=0}^{3}\sum_{j=0}^{3} p_{ij} \frac{dB_i^3(u)}{du} B_j^3(v) \qquad \text{Eq. 4}$$

The same holds for the derivative with respect to *v*. Therefore, before rendering, we calculate the derivatives of the basis functions and store them. We use Equation 4 and its *v* analogue to find the tangents, and then proceed to find the surface normal. The code for the loop is shown in Listing 4.

Now, while the curves didn't slow down from our naive implementations, this patch demo shows quite painfully why optimization is very necessary. It runs at a steady 30 or so frames per second (again, on my Voodoo2), but that's just one patch. If you tried to base a terrain system on this implementation, it would be painfully slow. After all, consider the work we're doing. By default, the tessellator breaks the surface into 100 points. At each point, we're evaluating 32 cubic functions and 32 quadratic functions, then doing a vector cross-product and a vector normalization (ouch!). Then, for each point, we're asking OpenGL to light it, which is not cheap either. Plus, we're not caching any of this between frames, and we're actually allocating and then deallocating the space every frame. So we're doing a lot of work, much of it entirely unnecessary.

Nonetheless, it works. We're rendering a lit Bézier patch, and even if it is a bit sluggish, it looks pretty good. Now, if only we could do something with it...

REFERENCES

- Farin, Gerald. *Curves and Surfaces for CAGD, A Practical Guide*. New York: Academic Press, 1997.
- Garland, Michael and Paul Heckbert. "Surface Simplification Using Quadric Error Metrics." *Proceedings of SIGGRAPH* (1997): pp. 209-216.
- Mortenson, Michael E. *Geometric Modeling*. New York: Wiley Computer Publishing, 1997.
- Watt, Alan and Mark Watt. *Advanced Animation and Rendering Techniques: Theory and Practice*. New York: ACM Press, 1992.
- The full source to the Hermite curve demo, Bézier curve demo, and Bézier patch demo are available from my web site at http://www.cs.dartmouth.edu/~bsharp/gdmag/

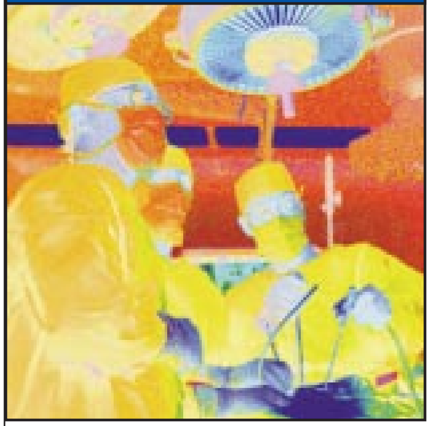
Moving from Theory to Application

There are certainly a number of loose ends. We've covered Bézier and Hermite curves and Bézier patches, but the implementations so far are entirely unoptimized and the patch demo is rather sluggish even for what little it is supposed to do.

Furthermore, we haven't seen an example of using these things in a real application. The demo code is just that — a demo of a curve or surface floating in black space. There is still a fair amount of material to cover before we can turn these into something real.

Next month, I'll cover some optimization techniques for Bézier curves and surfaces. We'll also see how to form other surfaces and objects by joining Bézier patches together, and look at some of the properties of such objects, as well as some of the problems that can arise from the new techniques. Finally, having covered all of this, I'll finish off the article with a far more interesting demo. ■

Back



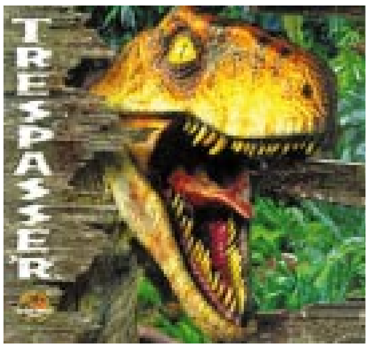
D eamW k I e ac i e TRESPASSER

By Richard Wyckoff

One seldom hears the true story of what happened at the place where the world changed. How it began. What were the reasons? What were the costs?

—John Parker Hammond

This quote from TRESPASSER's opening movie serves just as well to open the real story of a game development team's struggle to create a breakthrough dinosaur game as it does to open the fictional story of Hammond's struggle to develop a biotechnological breakthrough and clone dinosaurs.



An Ambitious Project

The parallels between the TRESPASSER project and Hammond's cloning project were numerous: ambitious beginnings, years of arduous labor, and an eventual tragic ending. Hammond's diary, as related in the game itself, dwells on the past and never attempts to explain Hammond's future direction now that he has failed so grandly. This postmortem is intended to be much more forward looking.

TRESPASSER was begun by two former employees of Looking Glass Technologies, Seamus Blackley and Austin Grossman. By the time the game was rolling, two more ex-Looking Glass employees would

Richard Wyckoff was a designer on TRESPASSER who worked previously at Looking Glass Technologies in small roles on projects like FLIGHT UNLIMITED and TERRA NOVA. He's currently applying what he learned from TRESPASSER to an unannounced project at Knowledge Adventure. Contact him at rwyckoff@loonygames.com.

join the team, and our common background was instrumental in setting the direction for the project.

The Concept

The pie-in-the-sky concept for TRESPASSER was an outdoor engine with no levels, a complete rigid-body physics simulation, and behaviorally-simulated and physically modeled dinosaurs. The underlying design goal was to achieve a realistic feel through consistent looks and behaviors. Having an abandoned island setting was a useful way to exclude anything which did not seem possible to simulate, such as flexible solids like cloth and rope, wheeled vehicles, and the effects of burning, cutting, and digging.

The game would play from a first-person perspective, and you would experience the environment through a virtual body to avoid the "floating gun" feeling prevalent in the WOLFENSTEIN breed of first-person games. Combat would be less important than in a shooter, and dinosaurs would be much more dangerous than the enemies in traditional first-person shooters. The point of the game would be exploration and puzzle solving, and when combat happened, it would more often involve frightening opponents away by inflicting pain than the merciless slaughter of every moving creature.

The original plan for TRESPASSER certainly seemed like a good one. It was very ambitious, but the team made trade-offs on implementation and execution time from the very beginning. For instance, the team wouldn't attempt to do multiple or moving light sources or QUAKE-style shadow generation in order to accommodate arbitrary numbers of moving objects and long, wide-open views. Unfortunately, there is a difference between having a plan and successfully executing it, and the product that we eventually shipped was as disappointing to us as it was to the great majority of game players and game critics.

Some have dismissed TRESPASSER altogether because it was such a visible failure. Respected industry columnists and editors use it as a reason why physics is bad, or make it the butt of their jokes ("at least it wasn't as bad as TRESPASSER!"). However, from a project perspective there were a number of successes. Before we get into the problems which ended up sinking the ship, let's look at these successes.

What Went Right

1. USE OF LICENSE. Making a new story with someone else's licensed property is often creatively stifling for designers and ultimately disappointing for fans of the original work. The *Jurassic Park* license could have been an especially limiting one, representing some of director Spielberg's and novelist Crichton's weakest work. However, TRESPASSER's Hammond diary actually contains lots of interesting tidbits about the early days of characters like Henry Wu (the scientist from the beginning of the first movie) and Dennis Nedry (Wayne Knight's character who basically caused the first disaster), which made the game world a richer environment. The player can also check out locations on the island which imply backstory which isn't explicitly told, like Henry Wu's house with its 1980s executive bachelor stylings, Nedry's office with its poster for the fictional computer game series "Swords of Kandar," and Hammond's lavish mansion. The settings and the diary itself serve to reveal much of Hammond's motivation and personal reactions to the building of Jurassic Park, creating more of a character than exists in either the books or the movies. (Crichton killed Hammond off in the first book, anyway.)

The overall plot of our game is as simplistic as most others (the character must find a way to escape a death trap), but the details revealed about the *Jurassic Park* world extend it in a way which is faithful to the originals. Although it is quite likely that the next *Jurassic Park* movie to be released will make TRESPASSER non-canon, for now it stands as the only real extension of the series published. If there are such things as *Jurassic Park* fanatics and they were able to look past the game play flaws, they hopefully enjoyed our development of the world.

2. ART AND MUSIC. On an individual basis, the models created for TRESPASSER rank with the best-looking work done for computer games. We limited the largest texture size to 256×256 pixels, and at model import time textures were converted to 8-bit paletted images. But artists worked with their models using 24-bit art in 3D Studio Max, applying the textures using any mapping methods that Max supported. We had the standard limits on visible polygons, so most models were made with as few as possible: dinosaurs ranged from 300 to 500 polygons and trees from 50 to 120, for

Concept art for TRESPASSER.

TRESPASSER

DreamWorks Interactive
Los Angeles, Calif.
http://www.dreamworksgames.com

Release date: December 1998
Intended platform: Windows 95/98/NT
Project budget: Estimated $6-$7 million
Project length: 32-36 months
Team size: 39 Including a full-time staff of 7 engineers, 5 game designers, and 10 artists.
Critical development hardware: 266MHz Pentium II with 128 or 256MB RAM
Critical development software: 3D Studio Max 1.2 and 2.5, Photoshop 4.0, Microsoft Visual C++ 6.0, Visual SourceSafe 5.0

example. But this still gave our artists more complexity than was standard at the time.

Many of TRESPASSER's artists had never worked on games or done 3D modeling before, and some had never even used computers at all. This was a fairly deliberate decision, in an attempt to achieve a much higher standard of art than we were used to seeing on previous products. The number and resolution of textures we were able to support called for painting skills far beyond the average game-trained artist.

The music is one of TRESPASSER's best accomplishments. Originally, we were slated to use John Williams's score, but the cost proved to be execessive. Fortunately, our sound effects company, SounDelux, put us in touch with one of their stable of composers who specialized in "imitation" music. With very little prompting, he recorded about 30 minutes of music for us which in some parts far exceeds the rather forgettable work Williams himself did for the *Jurassic Park* movies.

Some reviews still accused us of having spotty or inappropriate music, but this was more an implementation problem than a problem with the music itself. The music was recorded as a couple dozen short sections which were scattered through the world on location-based triggers. Much like the voice-overs, more attention could have been paid to their placement so that they only played at appropriate and regular intervals. Even more desirable would have been a system with the ability to play tension or combat music loops and fade them in and out of the special-event songs to make it seem more like a continuous musical score.

3. INNOVATIVE SYSTEMS. Our artists were able to paint textures with near-total disregard for common memory-conservation practices, thanks to the texture caching system which one of the last programmers to be hired created fairly late in the project.

Textures for our game were MIP-mapped by our helper application, GUIApp, as part of the process of building level data (curved bump maps were also created at this time). A level could have a nearly unlimited amount of textures, and once MIP-maps were created, all textures and their MIP-maps were saved into a single swap file. GUIApp automatically organized the swap files into pages based on texture size, with the lowest couple of MIP levels for all textures on a set of pages which were always committed.

As the player moved through the level and objects came into view, the appropriate pages from the swap file were accessed. In any circumstances where an appropriate texture hadn't been loaded in yet, the always-committed MIP-maps could be used until the higher-resolution texture had been loaded. In theory, this could result in a frame or two where an object was textured at a lower resolution than desired, but in practice it rarely happened, even on the most texture-intensive levels.

Another major system for TRESPASSER was its audio system, which we described as "real time Foley" because of its ability to generate collision and scraping effects between differing sound materials in real time. Although the system could have used more sound material data, even with what it had it resulted in some wonderfully immersive sound effects which most other games do not duplicate — things like scraping a board down a concrete surface or hitting an oil barrel with a metal bat sound almost perfect. The system doesn't just play two stock effects but actually chooses from several samples and sets volumes based on the strength of the underlying physics collision, with very natural-sounding results.

Finally, our image caching system, which rendered groups of distant objects into single 2D bitmaps to speed rendering, while responsible for the most disturbing visual anomalies in the game, was also by its own right an amazing piece of work. Image caching allowed scenes with tens of thousands of polygons to be rendered in near-real time, and it is the first technology which has allowed outdoor scenes to have a reasonable fraction of the complexity of the real outdoors.

4. OUTDOOR LEVEL DESIGN. When we created our terrain geometry, we were deliberately trying to avoid the "marbles in rubber" look of a lot of bad fractally-generated outdoors. To this end, we decided that we needed to base our island on real-world terrain rather than build it from scratch. Luckily, we had a real-world model to go from: Costa Rica's Cocos Island, the same island which Crichton used as his inspiration for Site B. Unfortunately, no relief maps of sufficient detail existed for the island, so we ended up having our lead artist sculpt a large model of the entire island, had it laser scanned, and did all our final work on it in 3D Studio Max.

Modeling was only the first step to creating a level. After most of the terrain was established, it took significant time to populate the levels with objects. There were 10-15,000 trees, shrubs and rocks in most levels, and a few thousand man-made objects as well. Every object could be placed individually, but for time-saving reasons we generally used groups of objects for areas off the gamepath and only spent a lot of time hand-placing items in places we knew the player would go. The rolling terrain and a random-delete tool we often ran for optimization generally kept the repetition from seeming obvious.

In the end, although our levels didn't quite fulfill our personal expectations, they usually look more like real environments than previous games. Starting from a real map seemed to be the most useful tactic for this, and looking for a real-world starting point for vegetation placement is the next obvious step for outdoor games.

5. REALISTIC PHYSICS IN A FIRST-PERSON PERSPECTIVE. The first discovery we made as our physics simulation was slowly implemented was that it was an engrossing toy. When we finally got support for compound object physics (so that a bench could consist of a top and two legs instead of a single block, for instance), it was possible to spend an hour just dropping the bench onto things to see how it would catch on edges and flip and slide around. Toys do not make games, however, and trying to establish game play that would work with our simulation was our major challenge.

Due to the vagaries of our particular physics simulation and interface, we eventually arrived on game play that primarily involved knocking things over rather than stacking things up. Knocking over will probably be the first application of realistic physics to see widespread use, as it will work even with a less-than perfect physics model (such as TRESPASSER's). It is also a behavior which easily shows off the difference between realistic physics and what we usually referred to as "fake" physics — compare pushing a box off a ledge or hill in any game which actually lets you move boxes to the same action in TRESPASSER.

Since our game play was supposed to revolve around the physics, however, we needed to apply that knocking-over behavior in slightly more sophisticated ways than just using it as eye candy. The best uses that we found in the small amount of time we had involved knocking stacks of boxes down to fill holes, or unbalancing something resting on a high ledge in order to get it or use it as a step. It also quickly became apparent that building even a seemingly simple knocking-down puzzle required a much more highly refined sense of physical laws than many of us had.

What Went Wrong

It might seem as though TRESPASSER deserves more entries in its "what went wrong" section than the usual project postmortem. However, TRESPASSER's failings are actually few in number. Unfortunately, the failings that we did have were serious enough to more than outweigh our successes.

1. SOFTWARE-ORIENTED RENDERER. TRESPASSER was begun before the original Voodoo started the wave of 3D hardware popularity. The TRESPASSER engineers set about to create an engine in the old-school manner: they picked some previously-unseen rendering technologies and implemented them, ignoring any issues of compatibility with hardware cards. Our engine's two most incompatible features were its bump mapping (which used a true geometrical algorithm that could take surface curvature into account) and its image caching. Image caching was intended to allow real-time rendering of huge numbers of meshes, but it was also the system almost solely responsible for the graphical anomalies of popping, snapping trees in the game. The other primary visual artifact of the game was the frequent sorting errors, but this was a result of poorly-constructed levels which continually handed our depth-sort algorithm more polygons than its limit, and not a direct result of image caching itself.

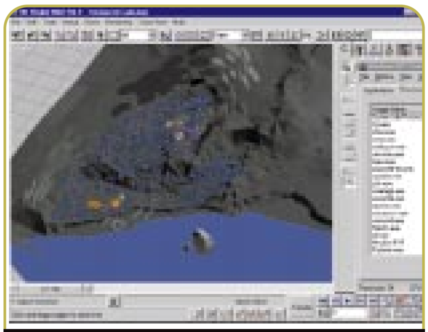
The image cache system worked by rendering distant objects into 2D bitmaps on the fly, and then updating them when the angle or distance changed enough so that the 2D representation was no longer sufficiently accurate. This may sound familiar to those who remember Microsoft's Talisman architecture, and there was a hope at one time that our game would be a "killer app" for Talisman accelerators, but resistance from key figures in the industry and 3dfx's sudden popularity pretty much put an end to Talisman.

TRESPASSER ended up slipping by more than a year, as did many games of its time. Our hardware programmer put in a valiant effort in the last half year of the project, and managed to get much more use out of 3D hardware than we initially thought possible. We ended up with a fairly unique mixed-mode renderer which drew any bump-mapped objects in software and the rest of the scene with hardware. Unfortunately, the large number of bump-mapped objects present in our game, such as all the dinosaurs and nearly every crate, meant that the fill rate advantages of accelerators were often negated.

In addition to being a costly software-only rendering method, our bump mapping was never very evident: we could have used multiple, moving light sources and had the art staff make better use of bump maps in their objects. Many of the bump maps were created by simply converting the original texture to grayscale, an artist's hack that works for rendered images and animations but not in real-time 3D. Image caching was an even bigger problem than bump mapping, because although it was the key technology that we were using to try to improve scene complexity, the game's visual quality was also the source of most people's complaints. It seems clear in retrospect that we should have made a tradeoff somewhere along the line and either dropped the physics technology and physics game play in favor of the rendering technology, or more likely dropped the rendering technology and the ability to do complex scenes in favor of the physics technology.

2. GAME DESIGN PROBLEMS. The biggest indication that TRESPASSER had game design problems was the fact that it never had a proper design specification. For a long time, the only documents which described the game play were a prose-style walkthrough of what the main character would do as she went through the game, and a short design proposal listing the keys which would be used and some rough ideas of what game play might actually be.

Our experiences on TRESPASSER made it clear that it is worse not to have a design specification at all than to have one which becomes out of date and is


Level design in Max was tolerable, not enjoyable, for the TRESPASSER team.

frequently rewritten. TRESPASSER started and finished weak in the game design, and this affected every other part of the project.

When it became clear that the technology was not going to exist to support the initial high-concept design, it would have been best to throw out all our existing notions and reinvent the game. Unfortunately, our license made it all but impossible to throw out the original TRESPASSER concepts. The only major deviations from the original idea were the change from constructive, stacking-based physics puzzles to destructive, knocking-over puzzles, and an attempt to make combat more prevalent in order to shore up the weakness of the destructive physics puzzles. Since no part of the TRESPASSER code was written to be good at doing first-person shooter game play, this attempt to make shooting a more important feature only ended up flaunting some of the weaker points in the game, like the lack of an inventory system and the slow frame rate.

3. TOOLS PROBLEMS. TRESPASSER was built entirely in 3D Studio Max. There was no level editor, only the generically-titled GUIApp, which was the game with a debugging shell — not really a tool at all. Our level creation procedure consisted of arranging 20–25,000 meshes in Max, using dummy meshes to represent game play objects like triggers, and typing trigger code into these objects' "object properties" buffer.

There were two unexpected and incredibly severe drawbacks that we discovered only after it was far too late to change our method of building the game world. The first problem was that Max is basically unfit to work with more than about 5,000 objects at a time. TRESPASSER levels averaged 40MB in size, and could take a couple minutes to load on the systems our designers used (Pentium II-266 with 256MB RAM). When all objects in a level were visible, it could take from 30 to 60 seconds to respond after clicking on an object to select it, making fast work difficult (to say the least).

The second problem with the Max method was our use of the object properties text buffer. The buffer seems to be one of those features which no one ever used before, because we discovered that if more than 512 characters were typed into an object's properties buffer, Max could become unstable. If Max didn't crash outright and a file was saved with one of these bad objects, it would become unloadable. TRESPASSER's technical artist wrote many design tools in MaxScript and also coded warning scripts to guard against problems such as properties buffer overflows, but these solutions only made designing in Max tolerable — not enjoyable.

There was an additional wrinkle to the process of using Max to create levels, and this was the export step. A Max plug-in converted data into the game format, but our particular exporter caused a lot of problems. It was developed by a programmer who worked from home, an hour away from the office, and used a separate code base with unique classes and a different version of the compiler. This was also his first project in 3D, and it became the second-most delayed part of the project after physics. Until the last year of the game, there were significant bugs in the exporter which required time consuming work-arounds. Important functionality, such as the ability to export object properties, also was not delivered until very late in development, which prevented designers from implementing game play. In the end, the exporter was assigned to another programmer and rapidly brought up to usability, but it had already delayed level building significantly.

4. AI PROBLEMS. The largest problem with the AI system was that its progress was blocked by a lack of dinosaurs with which to test it. The first time a dinosaur made the transition from a separate test application into the game was in early 1998, with significant missing functionality which prevented the completion of visually-important AI behaviors, like howling and glaring. The first quadruped went in around the early summer of 1998, about four months from the then-intended ship date (as it turned out we slipped by about another month).

The dinosaur AI was a state-based system, based on the creatures' emotions. It became apparent once dinosaurs were working well enough to put into levels that the differences between the activity states were not discrete enough. Dinosaurs were governed by a set of emotions which theoretically would prompt them to pick appropriate responses at any time. However, in practice they would end up oscillating rapidly between many activities, sometimes even literally standing still and twitching as they tried to decide what to do. Making a usable dinosaur required disabling all but one or two of their activities. This allowed aggressive dinosaurs to really be aggressive, but it also meant that most dinosaurs were as single-minded as the traditional videogame monsters we were trying to one-up.

The AI system suffered from the lack of a clear game design. There are two scenes in the *Jurassic Park* movies which demonstrate quintessential dinosaur game play: the scene in *Jurassic Park* where the kids hide from raptors in a kitchen, and the scene in *Lost World* where Jeff Goldblum deals with several cautious raptors in the ruins of the town. Both of these scenes rely on dinosaurs which can be fooled by ducking behind objects and which can home in on or be distracted by localized noises. Neither of these two fundamental abilities is actually present in the TRESPASSER dinosaur AI. Instead, the dinosaurs have a simpler and more industry-standard detection radius which doubles as sight and hearing and is not blocked by objects in any way. Without a design specification calling for these kind of behaviors, though, the AI development went in directions which ended up being largely unsuitable for game play. This problem wasn't even discovered until a few months before we shipped, when there was only time to work around it rather than completely rework it.

5. PHYSICS PROBLEMS. The box model was TRESPASSER's most significant physics innovation: it was intended to be a complete simulation of any arbitrarily-sized box interacting with a number of other boxes. The approach used in TRESPASSER was what is known as the "penalty force" method. In incredibly simple terms, when boxes collide, they are allowed to intersect with each other (mathematically), and then they push each other apart until they are no longer intersecting. The penalty force model is generally believed to be an unworkable one by the few other people in the industry attempting real-time solids models, but our physics programmer believed he could make it work.

There were several notable flaws with TRESPASSER's solids model as shipped: it ended up only working well when used with roughly cube-shaped boxes with dimensions between 0.5 and 1 meter on a side, it did not model friction well, it was extremely slow, and it was not free of interpenetration even within the size constraints.

We were aware that physics would be slow, and that ten boxes at once would represent the practical upper limit, but we had not expected that so much of that physics overhead would be eaten up by the dinosaur body physics, which used five boxes in the worst cases: head, body, tail, and two feet. Although the dinosaur physics boxes executed faster because they did not interact with each other, it turned out that in common cases (like a scene displaying two raptors and the player holding a gun), the physics budget was completely consumed and there were no were no processor cycles left to handle knocking over a stack of boxes.

Physics speed was an issue, but other problems were more severe. The game design depended on boxes of sizes other than small cubes, and we ended up including many objects outside that safe range. Unfortunately, all objects outside the safe range, even large cubes, were more prone to reveal the most egregious problem with the physics system: interpenetration. At best, these interpenetration bugs completely blow the consistency of simulation we tried to set up, and at worst they make the game unplayable. If it was not clear before shipping TRESPASSER, it is clear now: no amount of interpenetration is acceptable, and preventing it absolutely should be the number one concern of any physics coder.

TRESPASSER's dinosaurs and the arm itself were inverse-kinematic (IK) systems controlled by physical models. Originally, the dinosaurs were supposed to be full physically-modeled bipeds whose physics actually knew how to use legs to stand, walk, run, and jump. They ended up with more standard game movement physics and an IK animation system very similar to Looking Glass's TERRA NOVA. Just like in TERRA NOVA, the dinosaur legs frequently stretched, bent, and popped as the IK system struggled to handle the physically impossible movements that the simplified physics generated.

This movement problem was a case of lacking realistic boundary conditions. The joints of the arm never had realistic limitations put on their rotation or even limitations on the distances between joints. A system written by a different programmer sat on top of the underlying arm system and continually tried to make sure it had not moved into an impossible position, but as a separate system it could only be partially successful at best. The arm as shipped would often go almost out of control for a few frames, stretching or spinning in impossible ways. During this time the fragile feeling of connection between the player and their character would be shattered.

The arm suffered not only from its unbounded model but also from bad design choices. Its creator intended it to be wholly context-insensitive. The fact that guns, unlike other objects, get held fairly steadily and away from the body was only a result of some key members of the test staff adding their voices to the cries which had been coming from within the team to fix shooting so that it was easy to hit a desired target. It should have been obvious from the mere fact that a 2D interface was being used to move in 3D space that an even larger amount of context sensitivity was needed. If a truly successful virtual arm is ever to be implemented, simple mouse movements will have to be translated into complicated arm movements based on what is in or near the hand, or it will be as impossible to use as TRESPASSER's arm. That there was a conscious decision to avoid context sensitivity in our project is indicative of the larger problems with physics in our project. The physics code was largely written in a vacuum and tested in separate applications and non-representational levels, and not enough attempts were made to analyze it from a player's perspective and design it to support game play in every way.

Lessons Learned

How did TRESPASSER end up shipping with the number of problems that it had? TRESPASSER was a project with management problems at all levels. It suffered from being an innovative, technologically ambitious project produced by a team with little previous management experience at a company which had not yet gained institutional experience from publishing significant, less-ambitious projects.

It has been nearly half a year since TRESPASSER shipped. In that time it has gone from a gigantic Christmas letdown to an occasionally-referenced joke. The team has mostly disintegrated. Some quit, some were let go, and those remaining were distributed across several different projects. The engine is effectively dead, and the new DreamWorks motto is "licensed technology" — probably a good idea for the company but pretty disheartening for the engineers who created last year's most innovative, if not quite best, engine. A group of Internet fans proclaimed themselves the TRESPASSER Hacking Society and have taken up the lunatic task of trying to figure out how to build a mod for an engine which was barely usable with its in-house tools.

That TRESPASSER shipped at all is a testament to the strength of the individual members of its team. In looking back at my own experience, I find that I learned a lot about game development, and I'm already putting that knowledge to work. Although the game does not fulfill the many high hopes I had for it (or even my base expectations), I am happy with the work I put into it. I also hope that every other person who contributed to the massive effort is equally proud of their work, and that sometime in the future we all have a chance to make a major project that succeeds where TRESPASSER failed. ■

Editor's Note: A longer version of this article is available on Gamasutra.com.

Back

SOAPBOX

by Chris Hecker

So You Want to Do Movies? Good Riddance!

Nothing annoys me more than game developers who long to work in a different medium. From the artist who dreams of working at ILM to the musician who's waiting for that big recording deal to come through, these developers don't realize the true potential of games as an artistic medium. Even some of the highest-profile game industry personalities seem more interested in using games as a stepping stone into Hollywood than they do in making games a first-class art form that can stand alongside movies, music, books, painting, and the other accepted media.

Why do I get so aggravated by these grass-is-greener developers? Because they don't see games as a creative medium worthy of respect. Yes, one needs a bit of imagination and optimism to see how games will someday be as mature as literature or painting. However, people also needed imagination back in 1900 to foresee how film would be the mature art form it is today. I'd bet the filmmakers who lifted movies up from *The Great Train Robbery* to today's standards didn't do it because they all secretly wanted to be playwrights, painters, and musicians. They did it because they could see the potential of motion pictures for creative and artistic expression — and wide sociological impact.

Developers often complain about the limitations of games as an expressive medium. We can't have graphics as detailed as Pixar's animations, our audio is lame compared to what you can get out of a CD on your stereo, and so on. Yes, there are loads of technical and aesthetic limitations to games. But you don't hear painters complaining that oil paints aren't water soluble, or that water colors don't create texture on the canvas. Artists who work in mature media — like the different genres of paint, film, and writing — revel in the differences between their media. They don't complain about them! Constraints and limitations are part of a medium's charm and character — not to mention its identity. If you want the characteristics of oil paints, go use oil paints. If you want a new medium that has the best characteristics of both oil and water colors, then go invent it — but don't be surprised if it doesn't quite work out.

Enough about other media. What is the defining characteristic of games as an artistic medium? The answer is clear: interactivity. We finally have an art form where the work of art itself can respond to the viewer. Everyone in our industry pays lip service to interactivity, but this ability for each audience member to create his or her own intimately personal experience is the critical reason why games will impact society at large, not just testosterone-addled 16-year-old boys. Interactivity is a completely novel artistic opportunity, and it's clear from our crude attempts to harness it that we have a long way to go before our craft matures. But it *will* happen.

In that poetically ironic sort of way, interactivity is the basis for all of our limitations as a medium as well. Films will always be more graphically detailed than games because they don't have to generate scenes in real time. That's an inherent fact of interactivity. Yes, we can always work towards more realism and higher graphical standards — but we will never catch film on that front. Rather than sending our resumes to Digital Domain, we should recognize this limitation and appreciate that in return, we're gaining something film will never achieve — a true two-way dialog with the viewer.

We have a responsibility to our nascent art to help it grow into a full-fledged medium. Someone among us, or someone who has yet to join the game industry, will develop our equivalent of *Birth of a Nation* and the world will never be the same again. As a game developer, you can either help this medium reach maturity, or hinder this process by viewing it solely as a stepping stone to other creative arenas. If you're not committed to the growth of games as an art form, then don't let the door hit you on your way out. ■

Chris Hecker hopes somebody does the game equivalent of Birth of a Nation *soon, so somebody else can do the game equivalent of* Casablanca, *his favorite movie. He can be reached at checker@d6.com.*