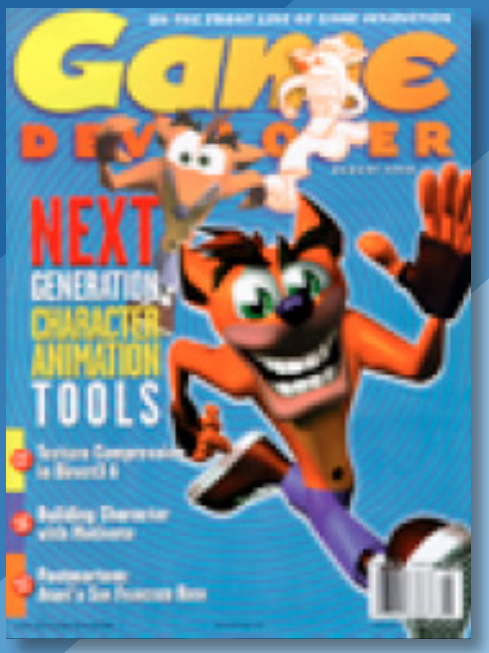
gd

GAME DEVELOPER MAGAZINE

AUGUST 1998



UBM Tech

Requiem for a Game Engine Company

If you were in the early stages of developing an Internet-based game last year, and you heard about a company that "provides game and simulation content developers with a software solution to build and deploy stand-alone and network-based applications that make full use of interactive 3D graphics," would you invest in it?

Apparently not too many people did, because Newfire Inc. — whose Catalyst/Torch products supplied the above marketing quote — died a quiet death last spring. Maybe I'm overly sentimental, but I was disappointed that the company couldn't stick it out. I really want to believe that there's room in the industry for a game engine company.

In the wake of its passing, I've been considering whether a company like Newfire can be successful. I've spoken with a number of game developers as well as employees at various tool companies on this subject, and it's been interesting to hear the different takes. While I haven't come up with a pat formula for success (like I'd still be here typing if I had), I have some observations to share.

I think the biggest hurdle that game tool companies have to clear is the "not invented here" (NIH) bias that many game developers harbor. And it seems that the more core functionality a tool attempts to provide, the more skeptically it's viewed by developers. There's a pervasive attitude — which is not entirely unfounded in my opinion — that shrinkwrapped game engines can't cut it at the core of a game. A tool company trying to sell such a product a long road ahead of it.

First, there's the credibility issue that these companies have to overcome with game developers. Developers ask, "Who are these people providing this technology, and why should I pay so much money to them instead of coding the functionality myself?" A pedigree in game development or a related field is necessary to even be considered.

Second, the solution needs to solve a problem that developers *want* a solution to. I'm assuming you, a *Game Developer* reader, are passionate about game development and enjoy working on your project. Would you want to hand off a good portion of your game's core functionality to an off-the-shelf solution? It depends, right? Management might be interested in purchasing a product if it can shave some weeks or months off of a schedule, but if the tool upsets the development team and causes some talented people to bolt, those considerations can weigh in against the investment.

Third, developers need to see a proof of concept. Epic MegaGames, id, Monolith, and others can get away with licensing their engines for hundreds of thousands of dollars because their games say more about their engines than any salesperson or datasheet could. Newfire, which was selling for a fraction of the cost of these engines, couldn't compete because there weren't any titles to hold up and say, "Here's what it can do when put into the hands of a client!"

Finally, to make matters tougher for a company like Newfire, there's the always contentious issue of pricing. The luxury pricing of the QUAKE and UNREAL engines not only provides great additional revenue for their developers, it ensures that only a top developer will be able to afford to license it, that those that do will be heavily incented to do great things with the engine, and that the engine won't be seen in too many other titles. To put this in a different light, I think Newfire would have been equally in trouble had it further lowered its price (it was already a fraction of the QUAKE and UNREAL engines) and succeeded in licensing out its engine to hundreds of companies. Would these licensees (or the gaming public) be happy with a slew of games that looked strikingly similar? I'll wager that the folks at Newfire considered these possibilities and had contingency plans to upgrade their engine or increase pricing if demand picked up, but nevertheless it points to the fragile nature of their product's economic model. ■

Game DEVELOPER
ON THE FRONT LINE OF GAME INNOVATION

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

COVER IMAGE Naughty Dog Inc.

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

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

Miller Freeman
A United News & Media publication

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

www.gdmag.com

INDUSTRY WATCH

by Alex Dunne

PORTAL SITES on the web just awoke to the luring power of games. Excite and Infoseek announced alliances with TEN to offer Java-based multiplayer parlor games. Both sites now offer Spades, Euchre, Hearts, Chess, Checkers, and various word games, with more on the way. Excite plans to use TEN's automated ranking system and host tournaments as well.

SEGA OPENED ITS KIMONO and revealed details about its upcoming Windows CE-based Dreamcast console, which launches in the fall of 1999 in America. The 128-bit console features a Hitachi RISC processor, PVRSG graphics hardware, a Yamaha 3D sound chip supporting 64 audio channels, and has built-in networking features. An interesting Dreamcast feature is its visual memory system, a console memory card that doubles as the world's smallest LCD-based portable game device.

PARRYING SEGA'S ANNOUNCEMENT, VM Labs let out some information about Project X. Project X is essentially an embedded technology which will be licensed out to consumer electronics companies for use in devices like home DVD players, digital satellite receivers, and set-top boxes beginning in 1999. Motorola has a non-exclusive license to develop, manufacture, and sell semiconductors and systems based on the VM Labs technology. Toshiba and Thompson Consumer Electronics (makers of GE, RCA, and ProScan brands) will incorporate Project X technology into products next year, and Activision, Capcom, Psygnosis, Hasbro, and Berkeley Systems are working on content.

PETER LINCROFT, a senior programmer for Totally Games who worked on LucasArts' X-WING VS. TIE FIGHTER, X-WING, and TIE FIGHTER has left to form Ansible Software in Berkeley, CA. Ansible will specialize in the science fiction genre, and

Massively Multiplayer SDK

VR•1 INC., a developer of massively multiplayer online entertainment, announced that its VR•1 Conductor SDK is now available.

The SDK, a component of the VR•1 technology suite, allows you to build massively mulitplayer games on the VR•1 Conductor platform, which powers online-only games such as MICROSOFT FIGHTER ACE. VR•1 Conductor lessens latency in online gaming by optimizing packets, setting bandwidth limits, accommodating varying modem speeds, and monitoring client and server CPU and network performance. It also facilitates network administrative functions such as game management, security, and billing. VR•1 Conductor SDK users can also take advantage of VR•1's Global Game Alliance, a strategy for partnering content developed on this platform with online gaming providers around the world. The charter members of the Alliance are six network providers that have endorsed VR•1 Conductor technology: Sony Communication Network in Japan; Bertelsmann Game Channel in Germany; British Telecom's WirePlay in England; Samsung SDS and DACOM in The Republic of Korea; and Videotex Netherlands and KPN Telcom's Online Service in the Netherlands. VR•1 expects to announce new alliance members in the coming months.

Cockpit view from MICROSOFT FIGHTER ACE, which is powered by the VR•1 Conductor platform.

- VR•1 Inc.
  Boulder, Colo.
  (303) 546-9113
  http://www.vr1.com

Polygon Decimation

RAINDROP GEOMAGIC is releasing the geomagic Decimator in July, and is showcasing it at SIGGRAPH 98.

Decimator is a polygon reduction tool designed to increase the rendering capability of any machine by greatly reducing the number of triangles in the surface mesh of a 3D model. During the decimation process, this software can improve the quality of the surface mesh while simultaneously preserving surface curvature and proximity to the original surface. The tool allows you to select regions on a model to preserve detail or reduce complexity. The interactive decimation command allows you to press a button and watch the polygons disappear, one by one, in real time. Features include the ability to import and export 3D model formats: .STL, .OBJ, .3DS, .VRML, .DXF, .WRP; selective or global real-time polygon reduction; editing capabilities; flat and smooth shading; wireframe and shaded viewing options; surface improvement with refine operations; and add-and-move-points operations.

Decimator works on Windows 95, Windows NT, and Silicon Graphics workstations. The suggested retail price is $295.

- Raindrop Geomagic Inc.
  Champaign, Ill.
  (800) 251-5551 / (217) 239-2551
  http://www.geomagic.com

ASTS

OF GAME DEVELOPMENT

Geometry Box II

GEOMETRIC COMPUTING has released the Geometry Box II SDK, a real-time 3D software development kit. The SDK contains three distinct elements: the Geometry Box Architect II Database Development Software, the Geometry Box InfiniteMotion II Real-Time Rendering and Interactivity Software, and the Geometry Box Class Libraries II.

These components are used to produce different elements of interactive software applications. The Architect software creates virtual worlds (databases) that players explore. You can place actors in the world and apply programmed behaviors to the actors. You can also use the Architect to store working versions of databases, and to create optimized, real-time ready versions of databases that InfiniteMotion can load and play back. The programming libraries are used to customize different aspects of InfiniteMotion, and to create plug-in behaviors that Architect can load. The key use for the programming library is the creation of behaviors that you can apply to actors and execute in InfiniteMotion. Geometric claims that Geometry Box streamlines the development process through its integration of tools and conversion processes with real-time rendering. Geometry Box II can be used for PC and arcade development, and has a suggested retail price of $595 (plus $20 shipping).

- Geometric Computing
  West St. Paul, Minn.
  (800) 334-8494
  http://www.geometricom.com

K6-2 with 3DNow!

AMD introduced the AMD-K6-2 processor featuring 3DNow! at this year's E3 in Atlanta.

The AMD-K6-2 processor combines 3DNow! instructions and superscalar MMX capability to increase 3D graphics performance. By improving the x86 processor's ability to handle floating-point calculations, 3DNow! technology lessens the gap between processor and graphics accelerator performance and eliminates the bottleneck at the beginning of the graphics pipeline. 3DNow! is a set of 21 new instructions that use SIMD (Single Instruction Multiple Data) and other performance enhancements to clear out the bottleneck between the host CPU and the 3D graphics accelerator card. The instruction set accelerates the front-end physics and geometry functions of the 3D graphics pipeline to enable full performance of the accelerators. This clears the way for improved 3D and multimedia performance. The DirectX 6, OpenGL 1.2, and 3Dfx Glide APIs are all optimized for 3DNow! technology.

The AMD-K6-2 processor is currently available. The AMD-K6-2/333 is priced at $369; the AMD-K6-2/300 at $281; and the AMD-K6-2 at $185, each in 1,000-unit quantities.

- AMD
  Sunnyvale, Calif.
  (800) 222-9323 / (408) 749-5703
  http://www.amd.com

Correction. The Bit Blasts section of the June 1998 issue contained an announcement of the new Miles Sound System 4.0 from RAD Game Tools. The phone number listed for RAD was incorrect. The correct number is (801) 322-4300.

plans to release an action simulation title in time for Christmas 1999.

CREATIVE TECHNOLOGY'S REVENUES and gross margins for its fourth quarter (ending June 30) fell short of analysts' expectations. Revenues for the quarter are anticipated to be about 10 percent lower than revenues for the same quarter last year. Despite the strong demand for its Voodoo2-based cards, the company said that the recent collapse of prices in the low- and mid-range 2D/3D graphics market reduced Creative's margins and sales.

THE ACADEMY of Interactive Arts and Sciences (AIAS), a sister to the Academy of Motion Picture Arts and Sciences (think Oscars) handed out its first-ever Interactive Achievement Awards at E3 in Atlanta. The biggest winner of the evening was Rare's GOLDENEYE 007, which received three awards, including "Interactive Title of the Year." Full IAA results are available at www.interactive.org.

ACTIVISION just inked some nice licensing deals. First, it signed on Marvel's X-MEN, which will debut next spring as a 3D fighting game for the PlayStation. Next it landed the rights to White Wolf's "Vampire" role playing universe, which is second only to AD&D in worldwide players. Nihilistic will turn that one into a 3D RPG for release in the fall of '99. Finally, Activision landed rights to the movie, *The Fifth Element.*

THE IDSA released an economic impact report about the interactive entertainment (IE) industry. First, it states that the computer and video game industry (a subset of the overall IE industry) racked up $5.1 billion in retail sales last year, and generated another $500 million on video game rentals. Second, the videogame and computer game industry grew over 35 percent in '97, making it the fastest growing segment of the entertainment industry — ahead of records, movies, and books. Third, the IE industry directly employs at least 50,000 US workers (up 18 percent from two years ago) and 17,000 abroad. Finally, total R&D spending in the industry reached $2b in 1997, and the average company invested 30 percent of its equity funding into R&D.

by Jeff Lander

Looking Forward with a Backward Glance at the CGDC

Another Computer Game Developer's Conference is over. I spent lots of time talking to friends, checking out the hot new products, and generally catching up on the state of the industry. This year, I returned feeling more enthusiastic than ever about the game business.

It felt to me as though games had really hit the big time. I'm not talking about when the press declared a marriage between Hollywood and the games industry a couple of years ago. That was just a brief, over-hyped flash of what was to come. What has really happened since then is that game development has become a major force in computer graphics technology and artistry.

I'm not talking about when the press declared a marriage between Hollywood and the games industry a couple of years ago. That was just a brief, over-hyped flash of what was to come.

For years, I've searched through computer graphics literature trying to improve my craft. Up until now, I've generally found game graphics between five and twenty years behind the state-of-the-art in computer graphics. When we were using Bresenham's routines to scan convert a polygon, they were trimming NURBS surfaces. When we were applying an affine texture to these polygons, they were applying specular lighting and bump mapping to an alpha-blended, perspective-correct textured micro-polygon. But we've been steadily catching up.

The Hardware

Much of our recent burst of speed has come from the consumer 3D hardware market. Affordable, high performance 3D hardware has brought us power that was previously only available in the realm of the Reality Engines and graphics supercomputer workstations. This past year, we've seen the ante upped time and time again. We are now at the point where the Christmas offerings from all the 3D card manufacturers should come very close to Brian Hook's dream in his September 1997 column ("All I Want for Christmas '98 Is a Hardware Accelerator That Doesn't Suck"). In fact, many of the offerings from the card vendors are exceeding everyone's expectations.

Let's take a look at the fall offerings in consumer 3D hardware shown at the CGDC this year.

Z-Buffering

Last year, we all agreed that a 16-bit Z-buffer was a reasonable baseline for all 3D consumer hardware for 1998. But it was hardly ideal. Many applications on the market suffer from Z-aliasing artifacts even on these 16-bit Z-buffers. When you're creating a game that requires resolution of near elements as well as distant objects, 16 bits of Z (or even W) are not enough. So, what did the card manufacturers have to say about this problem?

3Dfx was prominently displaying its Voodoo2-based boards. This board still uses 16 bits for the depth buffer. However, the buffer is now a 16-bit floating point number, and 3Dfx has added the ability to use this as a W-buffer, effectively increasing the far view Z precision.

S3 was the surprise of the show. After taking more lumps than the President last year, S3 came out with a shocker. I was one of the many who, when told that I needed to visit S3's booth, said "Yeah, right..." I was then told that I *really* needed to check it out, and boy, am I glad I did. Their Savage 3D made quite an impression. The display of TUROK running on a Savage 3D and beating Voodoo2 in performance was quite a sight. But what really interested me was the support for 24-bit depth and color buffers. As of the CGDC, these features were not yet being exercised, but I look forward to seeing them in action.

Nvidia wasn't showing the new TNT board on the show floor (it was shown to selective people behind closed doors), but the numbers they were quoting couldn't be ignored. The full-featured 24-bit Z-buffer should effectively eliminate most Z-aliasing problems.

Matrox was showing their new MGA-G200 card with a full 32-bit Z. However, they also were awaiting drivers to really

Is coming up with new technology all that Jeff Lander and the crew at Darwin 3D do at their home on the Digital Riviera? Mostly. But Jeff can also be seen helping in the rehabilitation of Marine Mammals along the So Cal coastline.

put the 32 bits through their paces. It's not clear to me at this time if this card uses the full 32 bits for Z or if the Z-buffer is 24 bits with 8 bits for a stencil, like the Nvidia TNT.

The NEC Power VR Second Generation (PVRSG) doesn't really have a Z-buffer, but it does use 32 bits of precision to handle polygon sorting. I haven't tried one of these boards out yet, so it's tough to say how that will compare with traditional Z-buffering.

Color Buffer

We've seen the 16-bit color buffer become the standard for consumer hardware this year, but it has problems. Anyone who has rendered a scene with smooth gradients has witnessed the banding problems associated with 16-bit graphics. But a hidden problem is the one associated with multipass rendering. As fill rates increase and two-pass texturing becomes more common, alpha blending will be used more and more. With only 16 bits of color precision, severe quantization errors can occur. An increase to 24 bits for color reduces this effect tremendously.

Another problem with 16-bit color buffers is that there is nowhere to store alpha information per pixel. This means that in order to render alpha-blended scenes correctly, the polygons need to be drawn in the correct order. All alpha-blend textures must be drawn after any polygons behind them. Two passes through your polygon database are sometimes necessary to get this order correct. If several alpha-blended textures overdraw each other, those polygons must be sorted in order to be drawn correctly. By storing 8 bits of alpha in the 32-bit color buffer, these considerations are eliminated.

There is good news on this front as well. The S3 Savage 3D and NEC PVRSG cards allow a 24-bit color buffer, and the Nvidia TNT and Matrox G200 cards support 32-bit buffers with full 8-bit alpha buffering. Figures 1A and 1B demonstrate the difference between a 16-bit image with 1 bit of alpha and a full 24-bit image with 8 bits alpha, both on the Nvidia RIVA 128.

Image Quality

Bilinear filtering is now the norm for 3D accelerated hardware. The Voodoo2, Matrox G200, and S3 Savage 3D all add single-pass trilinear filtering. The Nvidia TNT and NEC PVRSG add anisotropic filtering for even better image quality and less texture distortion in polygons viewed at an angle. All these cards now support per-pixel MIP-mapping. Lack of per-pixel MIP-mapping was the most noticeable graphics problem with the Nvidia RIVA 128, and thankfully, the TNT corrects this.

The display resolutions are also going up. Game developers can really start to consider much greater resolutions. Cards were displaying accelerated images at as high as 1,600×1,200 in 32 bits of color.

Texture Compression

AGP has allowed greater access to texture memory then ever before, but it's still not as fast as storing textures in chip VRAM. In order to maximize the number of textures that can stay in VRAM, some hardware cards have chosen to implement hardware texture decompression. Both the Voodoo2 and the Matrox G200 cards support forms of hardware texture decompression. S3 supports a texture decompression method that has been accepted for use in Microsoft's DirectX. However, it's unclear to me how this would become a standard, as S3 is seeking a patent on the technique. I don't see any great incentive for any other hardware company to support their standard. As a developer trying to make everything work on every card, I would hate to see each card manufacturer implement a completely different compression method. Unfortunately, right now, that's the way things are shaping up.

Texture Restrictions

It seems that restrictions on the size of textures may soon be a thing of the past. Both the Nvidia TNT and S3 Savage 3D support textures up to 2,048×2,048 textures with no size restrictions. Is anyone else out there starting to think real-time film resolution? I know I am.

Multitexture Support

QUAKE has started everyone thinking quite a bit about multipass rendering. Hardware companies are taking this very seriously. They're squeezing out the maximum fill rate possible from their chips in order to handle the overdraw necessary for multipass.

The 3Dfx Voodoo2 was first out of the gate supporting two-pass rendering of a single polygon in hardware. By allowing two textures to be combined in one pass, not only does it eliminate vertex transformations, but it also effectively doubles its fill rate. These calculations are actually done internally in 32-bit precision, eliminating the quantization errors that can occur when textures are blended in a 16-bit color buffer.

The Nvidia TNT is the second consumer card to announce two-pass rendering, with an even faster fill rate than the Voodoo2. This, combined with its deeper color buffer, should allow the

FIGURES 1A AND 1B. *A demonstration of the difference between a 16-bit image with 1 bit of alpha (1A), and a full 24-bit image with 8 bits of alpha (1B) both on the Nvidia RIVA 128.*

FIGURE 2. *Multitexture rendering in action with DirectX 6.*

creation of tremendously compelling images.

DirectX 6, which was released the week of the CGDC in beta form, contains API support for multitexture rendering. Developers can detect if the hardware supports rendering of multiple textures in a single pass. You can see a sample image of multitexture rendering in action in Figure 2.

As multipass rendering in consumer hardware becomes common, developers can start to exploit the rendering possibilities. Multipass rendering can make a difference in your application. Take a look at the images in Figure 3A and Figure 3B. Figure 3A is a scene lit without shadow maps and Figure 3B shows the maps applied. The resulting change of mood in this scene dramatically changes the player's experience. Look forward to effects such as shadow maps, environment maps, and detail textures showing up all over the place. DirectX 6 even supports what they Microsoft terms "bump mapping" by using multipass techniques. This green light from Microsoft has caused all the hardware vendors to add "Hardware Bump Mapping" to their product sheets. I'm sure this technique is nothing like what Jim Blinn had in mind when he coined the term in 1978. Still, this "embossed texture mapping" is quite effective. You can see a sample 3Dfx demo of the technique in Figure 4.

Programming API

Both Direct 3D and OpenGL were visible all over the place at the CGDC. Questions regarding card manufacturers' support for OpenGL were clearly addressed by all the drivers shown, either through a mini client driver (MCD) or full installable client driver (ICD). I can only hope that these drivers will put an end to the constant discussion of Direct3D vs. OpenGL in the development community. It seems that, finally, the choice is completely up to the developer, at least until Fahrenheit rears its ugly head again.

Geometry Acceleration

All of the next-generation 3D accelerators are moving what is called triangle setup into hardware. These are the triangle calculations that the card must make before applying the texture. The addition of full triangle setup has improved speed quite a bit, however, the cards still rely on the on-board CPU to handle geometry and lighting processing.

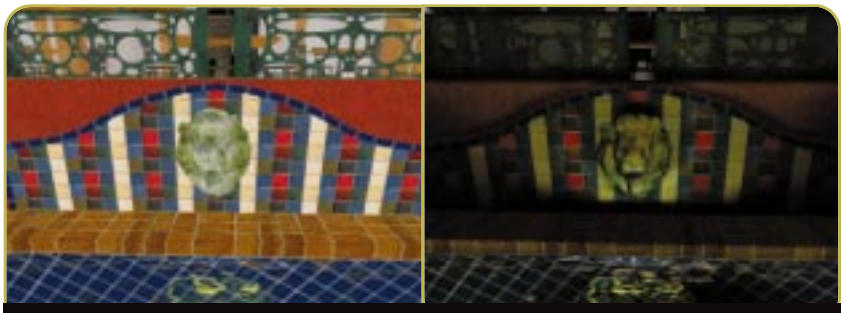
In order to speed this up, geometry and lighting will need to move to hardware as well. We saw our first glimpse of this happening at the show. While it's not a consumer card, the 3Dlabs GLINT GMX board demonstrated the complete OpenGL pipeline accelerated in hardware. The board handled full-view frustrum clipping, 16 dynamic light sources, and support for all OpenGL rendering primitives completely in hardware.

This method of allowing a coprocessor to handle all the transformation and lighting calculations is fairly controversial in PC programming circles. Questions linger about the need for a general lighting model or the transformation speed in the era of ever-expanding CPU performance. However, the concept of a transformation and lighting coprocessor is familiar ground for PlayStation developers. Certainly, the option is attractive assuming the price drops into the consumer's grasp.

The other benefit of committing the transformation and lighting model to hardware is that it opens the door to a true Phong shading model and bump mapping in 3D accelerators. These effects require that the hardware have the camera matrix and light information. The 3Dlabs GMX really addresses none of these issues, but certainly opens the door to the possibility. It should make us all consider what it is we want from the next generation of 3D hardware. Which brings me to...

The Software

Now that SIGGRAPH is here again, it will be interesting to see how SIGGRAPH compares to the CGDC. This year's game conference really pushed the envelope with discussions of new computer graphics technology. The freedom brought by the new wave of hardware will provide an unprecedented amount of bandwidth in games for new techniques. It really seemed to me that the theme of the CGDC, at least from a technology standpoint, was scalability — providing the richest, most realistic experience possible on all levels of end-user hardware.



FIGURES 3A AND 3B. *3A is a scene lit without shadow maps, 3B has shadow maps applied, both on Nvidia TNT.*

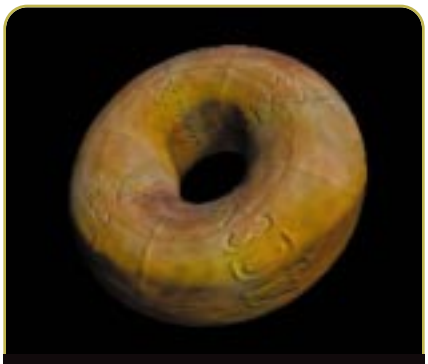


FIGURE 4. *Embossed texture mapping from the 3Dfx Voodoo2.*

Why is this necessary? Well, hardware is moving ahead so quickly, games cannot keep up. Consumers are buying new hardware to run the latest games, and there's no way for them to make the experience any better. 3D games are running on today's hardware at as high as 1,600×1,200 screen resolution, in 32-bit color, with all features turned on at 60 frames per second or more. This means that the game has failed at being able to provide a better experience for players on these new systems. It's not enough to run the game at higher resolution and color depth at faster frame rates. Games should be customizable to the point that they can drag any hardware down to its knees. This is what will give a game legs to stay on the cutting edge through more than one hardware production cycle.

So how do we do this in a real-time 3D game? Scalability. First of all, you need to represent your 3D models in such a way that they can scale to the hardware dynamically. The concept of level of detailing (LOD) in 3D models has been around for a while. Many of the games out there today have models in several LODs to aid performance. But to make the game truly scaleable, you need *many* levels of detail. In order to avoid the popping effect (when models change from one LOD to another), it's necessary either to blend between models, or to provide a form of continuous level of detail. The game community seems to accept this concept. At the CGDC, there were at least five sessions on forms of continuous LOD generation of 3D models. They seemed to fall into one of two camps — multiresolution modeling or higher-level primitives.

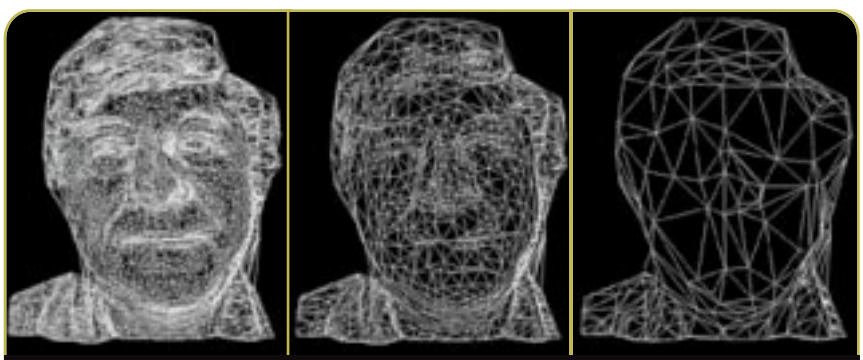
MULTIRESOLUTION MODELING. With multiresolution modeling, a mesh is made in the highest resolution and is algorithmically reduced as needed to the desired level. The work Microsoft has been doing in progressive meshes, led by Hughes Hoppe, has brought this technology to the attention of game developers. At the CGDC, Stephen Junkins of Intel gave a very good talk. He spoke about another method for creating these meshes based on the work of Michael Garland and Paul Heckbert. Intel has teamed up with MetaCreations to define an open file format for multiresolution models. It's unclear to me what they mean by "open" exactly, but a standard format that could be used by tool vendors, as well as developers, would be very helpful. MetaCreations was also demonstrating a tool that could be used to create these models. You can see a sample of several LODs generated from this tool in Figures 5A, 5B, and 5C.

It's not enough to run the game at higher resolution and color depth at faster frame rates. Games should be customizable to the point that they can drag any hardware down to its knees...

In my opinion, the biggest drawback to this technique is the lack of artistic control as the algorithm reduces the mesh. This leads to a situation where, given the same polygon budget, a talented modeler can create a much better low-polygon mesh than the algorithm can generate. However, it seems to me that these routines can be modified to allow artistic guidance in the tools. I will explore this idea more in a future issue.

HIGHER LEVEL PRIMITIVES. The other big area of discussion was "Is it time to abandon polygons?" That is, in order to create truly scaleable environments, is it necessary to create these worlds and objects out of a higher form of primitive? Several sessions were devoted to the topic of creating worlds with NURBS and other surfaces, and then converting these to polygons (tessellating) at run-time to the desired LOD. Michael "Sax" Persson of Shiny discussed a method of converting a polygonal character to a collection of primitive cylinders for MESSIAH. Creating models of higher-level primitives that can be quickly converted to polygons is tricky. This is clearly an active area for research in the game industry, and I will be looking into these techniques, as well as many others, in the coming months.

Enjoy SIGGRAPH, and we'll get back to coding some of this new technology next month. ■



FIGURES 5A, 5B, AND 5C. *High, medium, and low LOD models generated with the MetaCreations tool.*

FOR FURTHER INFO

For information on the 3D cards mentioned, contact the manufacturers.

www.3dfx.com
www.metastream.com
www.nvidia.com
www.powervr.com
www.s3.com
www.real3d.com
www.matrox.com
www.3dlabs.com

For software scalability information check out:

www.metacreations.com
www.shiny.com
http://www.research.microsoft.com/research/graphics/hoppe/

Michael Garland's web page at

http://www.cs.cmu.edu/~garland

Paul Heckbert's web page at

http://www.cs.cmu.edu/afs/cs/user/ph/www/heckbert.html

by Josh White

Advanced Texture Blending

This month, we artists aren't going to mess around with vague analogies — no, we're going to get to the heart of the matter. If we're going to claim to be experts, we must sneak onto the programmers' turf, climb down into the mechanical heart of the beast, and grasp the 3D API specification itself.

If you didn't read last month's column, this one may be tough slogging for you — that's why there's a review sidebar ("What is Texture Blending?"). Once we're prepared, we'll start with a technical introduction to texture blending procedure in OpenGL, then work through the OpenGL 1.2 specification itself, line by line. After that, we'll tackle some examples, and finish with some applications for all of this knowledge.

Texture Blending in OpenGL Simplified

In general, textures are blended in three steps: fetching the textures to blend, setting the blending method, and causing the blended texture to draw. Simple, right? Here's a little more detail about how the blending method is controlled in OpenGL. First, grab a pixel from each of the two textures we're going to blend. Add the two pixels after multiplying each one by its own translucency. Then put the resulting pixel on the screen.

The second step is where the real action is. We artists want to know how the blending method is controlled, and how the math works inside it. The players are the two input pixels (named **source** and **destination**), their personal factors (named **sfactor** and **dfactor**), and an output pixel. Here's the Sacred Blend Formula that relates these:

```
output_pixel = source * sfactor + destination *
  dfactor
```

What's sacred about this formula? Aside from the fact that it's part of the OpenGL 1.2 specification, the magic is speed. If you're working on a hardware-accelerated 3D game, the 3D accelerator chip can calculate this formula. Hardware calculation means that you can use the Sacred Blend Formula all over the place without slowing down the game. However, the calculation is being done in hardware, so it's essentially hard-wired, and so unlike most formulas, we can't simplify it.

We must use all the players in the formula because they're hard-wired, but we can work around them by making them useless. For example, if we want to add but not multiply, we set **sfactor** and **dfactor** to 1. We want to do that during an additive blend, for example. Say we have a light gray pixel (80 percent white) and a dark gray pixel (10 percent white), and we want to additive blend them. In this case, we set **sfactor** and **dfactor** to 1, and then run the formula:

```
output_pixel = 80% * 1 + 10% * 1
output_pixel = 80% + 10%
output_pixel = 90%
```

Wow, that was easy! No problem! Let's do a multiply blend. For this, we want to eliminate certain terms and multiply pixel values.

```
output_pixel = 80% * 10% + 10% * 0
output_pixel = 80% * 10%
output_pixel = 8%
```

What is Texture Blending?

Texture blending is the combination of two textures on a 3D model. QUAKE lighting, translucent textures, glowing light-sabers — these are all uses for texture blends. Figure 1 is my standard example of texture blending.

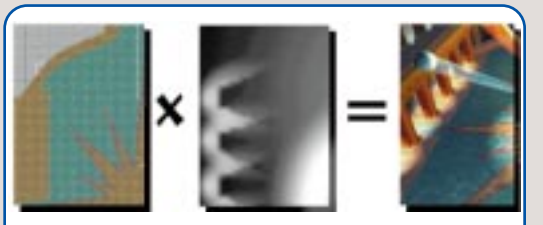


FIGURE 1. *Texture blending combines two bitmaps.*

As last month's column explained, blending is best understood by looking at what happens to a single pixel. For each pixel in an alpha blend, the computer checks if the alpha channel is white. If so, it outputs the pixel from the foreground bitmap. If not, it discards the foreground pixel and doesn't output anything, which leaves the background pixel unchanged.

Josh White runs Vector Graphics, a real-time 3D art production company. He wrote Designing 3D Graphics *(Wiley Computer Publishing, 1996), he has spoken at the CGDC, and he cofounded the CGA, an open association of computer game artists. You can reach him at column@vectorg.com.*

Devious, eh? We used the second pixel's value (10 percent) where the **sfactor** went before. Now that we're feeling cocky, let's jump into the deepest end we can find.

glBlendFunc Explained

The function **glBlendFunc** is at the heart of texture blending. If you can understand exactly how it works, then you can understand the lowest level of texture blending programming. Now we'll tour the OpenGL 1.2 specification. The first line reads

```
void glBlendFunc( GLenum sfactor, GLenum dfactor )
```

If you've never seen C code, you'll be confused by that line. The word **void** means that the function doesn't return anything; it just runs without reporting back. The stuff in parentheses tells us that the function takes two inputs, **sfactor** and **dfactor**. The word **GLenum** before **sfactor** and **dfactor** means that they are a type of data called "enumerated."

Here's how the OpenGL 1.2 specification describes this function:

> **glBlendFunc** defines the operation of blending when it is enabled. **sfactor** specifies which of nine methods is used to scale the source color components. **dfactor** specifies which of eight methods is used to scale the destination color components.

Note that this passage has introduced some interesting terminology. "Scale" means modifying a color's RGB values. If it had said "scaling down," it would have meant darkening the color.

WHAT IS ENUMERATION? Enumerated means an ordered list of names. For example, **enum color = (red, green, blue)** could define color to be one of three color-names. Enumerated data types aren't really smart; they're just dumb lists. If we set **color=red,** there wouldn't be any connection between the word "red" and RGB 1,0,0; it just stored as the first item in the enumerated list. So why not just store a number instead of a name? Names provide clarity in coding. Enumerated data types are there for the convenience of the coders.

So what are these possible enumerated types? Here's what the specification says:

> **sfactor:** Specifies how the red, green, blue, and alpha source-blending factors are computed. Nine symbolic constants are accepted: **GL_ZERO, GL_ONE, GL_DST_COLOR, GL_ONE_MINUS_DST_COLOR, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA,** and **GL_SRC_ALPHA_SATURATE.**

So, it says that alpha is being treated like a fourth color channel. That's interesting, because it's not quite how I expect artists to conceptualize alpha channels. To me, an alpha channel is more like a separate image that is associated with the colored image rather than a fourth color. This is important because it points out a difference in artistic thinking that is happening at the code level. Artists should be involved in designing this techie stuff so that they have input on the architecture in which they work.

For **dfactor,** the same types are listed, except that **GL_SRC_ALPHA_SATURATE** isn't included. It's an interesting list, and we can guess a few meanings just from the names, but it's not definitive. Let's keep going with the specification.

> [There are] eleven possible methods... Each method defines four scale factors, one each for red, green, blue, and alpha. In the table and in subsequent equations, source and destination color components are referred to as **(Rs, Gs, Bs, As)** and **(Rd, Gd, Bd, Ad).**

Note that color components are like what we call "channels" in Photoshop. That wasn't so bad, right? Don't worry, it gets harder:

> They are understood to have integer values between zero and **(kR, kG, kB, kA)**, where **ksubc = 2^msubc - 1 and (mR, mG, mB, mA)** is the number of red, green, blue, and alpha bitplanes.

Wow, that's a heck of an explanation there. It's as precise as you can get in English, but it leaves a bit to be desired if you're not a coder. Translated to casual English, it means that each color is a number between 0 and a maximum. The maximum depends on the color depth of the pixel.

WHAT'S COLOR DEPTH? Color depth refers to the idea that an image has a thickness or depth, measured in bits. More bits mean more accurate colors in the picture (and more memory used). Here are some examples: if we have 32-bit color, we've got 8 bits for red, green, blue, and alpha. With 8 bits for red, we can have 255 unique values, from 0 to 255 (as opposed to -127 to 128 or something weirder). Here's another example: if we're working in 16-bit color, we could have 4 bits for each channel, which is known as 4444. With 4444 color, we have 16 separate alpha values that range from 0 (fully opaque) to 15 (fully transparent). There's another kind of 16-bit color called 5551, which gives 5 bits for each color channel, but only 1 bit for alpha. So if we had 5551 16-bit color, our red channel would have 32 possible values (0-31), but our alpha could only have two possible values, 0 and 1.

Now I'm sure you're wondering, what about 8-bit color? It's special because 8 bits would only allow 2 bits (four colors) per channel, it uses a palette or index system instead of simply storing the RGB values. In fact, it's so special that blending can't handle it at all. The specification says so in the notes section: "Blending affects only RGB rendering. It is ignored by color index renderers."

Still following along? We're getting to the good part here.

Parameter	(fR , fG , fB , fA)
GL_ZERO	(0, 0, 0, 0)
GL_ONE	(1, 1, 1, 1)

As you probably guessed, **GL_ZERO** and **GL_ONE** are simply the minimum and maximum values. You can think of **GL_ZERO** as a fully transparent, black pixel, and **GL_ONE** as a completely opaque, white pixel. You detail-obsessed people should note that these values range from 0 to 1, not 0 to 255.

```
GL_SRC_COLOR          (Rs / kR , Gs / kG , Bs / kB , As / kA )
```

O.K., here's a real one to study. What the heck is that line of gibberish? Being artists, we'll start with some guessing. First, we'd guess about the name. **SRC_COLOR** probably means source color, right? Let's see. **Rs** is the red channel of the source image. **kR** is the maximum value for red. If we divide them, we get the red channel converted to a number between 0 and 1. For example, let's say we had a blue-purple 24-bit pixel with a 50 percent alpha. Its RGBA channels are 16, 0, 255, and 128. If we divide the red channel, 16, by its maximum value, 255, we get 0.0625 for red. Green is obviously zero, and blue is 1, and alpha is 0.5. So **GL_SRC_COLOR** would give (0.0625, 0, 1, 0.5) for our pixel. What a boring function, huh? All it does is translate our pixel's values into a 0 to 1 range. The color hasn't changed. Just you wait; it gets trickier.


```
GL_ONE_MINUS_SRC_COLOR          (1, 1, 1, 1 ) -
  (Rs / kR , Gs / kG , Bs / kB , As / kA)
```

This one's a little more interesting. It inverts the pixel, including the alpha channel. You can see that the second component is the same as **GL_SRC_COLOR**, so we have the RGBA in a 0 to 1 range, and then we subtract each value from 1. For example, our blue pixel's RGBA of 16, 0, 255, 128 gets converted to (0.0625, 0, 1, 0.5) as before. Then we subtract each part from 1 to get an inverted color: 1 - 0.0625 = 0.9375 for red, 1 - 0 = 1 for green, 1 - 1 = 0 for blue, and 1 - 0.5 = 0.5 for alpha. The result is (0.9375, 1, 0, 0.5), which is a lovely shade of yellow.

In this example, the alpha channel didn't happen to change, but normally we would find that opaque objects become transparent as well as opposite colors. This is weird from an artist's point of view; when we think "invert," we're usually thinking of color, not visibility. Again, I say this is an argument for artist involvement in low-level architecture design.

```
GL_DST_COLOR                    (Rd / kR , Gd / kG , Bd / kB , Ad / kA)
GL_ONE_MINUS_DST_COLOR          (1, 1, 1, 1 ) -
  (Rd / kR , Gd / kG , Bd / kB , Ad / kA )
```

These two functions, **GL_DST_COLOR and GL_ONE_MINUS_DST_COLOR** are the same as above, but for the destination pixel. Does it seem strange to distinguish source and destination so carefully? You might wonder, "Why not just have **GL_ONE_MINUS_COLOR** and forget about separating source and destination?" The answer is that we have more control if we can identify which pixel gets inverted.

```
GL_SRC_ALPHA          (As / kA , As / kA , As / kA , As / kA )
```

This function turns the alpha channel into a grayscale bitmap by copying the scaled alpha channel **(As / kA)** into RGB.

```
GL_ONE_MINUS_SRC_ALPHA          (1, 1, 1, 1 ) -
  (As / kA , As / kA , As / kA , As / kA)
GL_DST_ALPHA                    (Ad / kA , Ad / kA , Ad / kA , Ad / kA)
GL_ONE_MINUS_DST_ALPHA          (1, 1, 1, 1 ) -
  (Ad / kA , Ad / kA , Ad / kA , Ad / kA)
```

These three give us the variations on **GL_SRC_ALPHA**, an inverted version, and the destination-alpha equivalents.

```
GL_SRC_ALPHA_SATURATE           (i, i, i, 1 )
i = min (As , kA - Ad ) / kA
```

Now this is the tricky one. Remember that it isn't available as a destination blend; it's only possible for the source pixel. This function overwrites alpha to solid opaque, then puts **i**, which is some funky formula that seems to use alpha values, in RGB.

O.K., let's start deep inside **i: kA - Ad** is the maximum alpha minus the destination alpha, which is essentially inverting the destination alpha. The function **min()** returns the smaller of its two inputs. One input is the inverted destination alpha, and the other is source alpha. The final division scales that to 0 to 1. So we're getting the smaller of the source or inverted destination, scaled 0 to 1.

We conclude that **SRC_ALPHA_SATURATE** is used to convert an alpha channel to a grayscale, using the brightest alpha available, but we still don't know what the heck it's for. We'll need more context to determine that.

I'll spare you the accuracy issues that the specification describes next. Instead, we arrive at a far more interesting section: Examples.

Examples

Now we get to apply some of this hard-earned knowledge. Under the Specification's Examples section, we find this little passage:

> Transparency is best implemented using blend function (**GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA**) with primitives sorted from farthest to nearest. Note that this transparency calculation does not require the presence of alpha bitplanes in the frame buffer.

Recall the Sacred Blend formula we saw in the beginning of the article:

```
output_pixel = source * sfactor + destination * dfactor
```

This means that the specification is recommending that we do transparency blending like this:

```
output_pixel = source * SRC_ALPHA + destination * ONE_MINUS_SRC_ALPHA
```

Let's use our familiar example of a blue pixel with RGBA channels are 16, 0, 255, and 128, and let's blend it onto a gray background (RGBA 128, 128, 128, 255). So we know the source and destination, but what about the factors? We figure them out, just as we did earlier in this column. We saw that **SRC_ALPHA** is calculated by the formula

```
SRC_ALPHA = (As / kA , As / kA , As / kA , As / kA )
```

Plugging in our blue pixel's alpha value for **As** and using

the maximum possible alpha value for kA, we get:

```
SRC_ALPHA = (128/255, 128/255, 128/255, 128/255)
```

Dividing out the result gives us a reassuringly understandable answer: 50 percent for everything.

```
SRC_ALPHA = (0.5, 0.5, 0.5, 0.5)
```

We've got one factor done, so let's do the second one.

```
ONE_MINUS_SRC_ALPHA = (1, 1, 1, 1 ) -
  (As / kA , As / kA , As / kA , As / kA)
```

Recall that this is simply SRC_ALPHA subtracted from 1. We've already figured out that SRC_ALPHA is 0.5, so we see that 1 - 0.5 = 0.5. In other words, ONE_MINUS_SRC_ALPHA happens to be the same as SRC_ALPHA:

```
ONE_MINUS_SRC_ALPHA = (0.5, 0.5, 0.5, 0.5)
```

Now we have all the players in the Sacred Formula. With these values, we're ready to hit the calculator.

```
output_pixel = (16, 0, 255, 128) * (0.5, 0.5, 0.5, 0.5) +
  (128, 128, 128, 128) * (0.5, 0.5, 0.5, 0.5)
output_pixel = (8, 0, 128, 64) + (64, 64, 64, 64)
output_pixel = (72, 64, 192, 128)
```

Testing the Results.

Wow, we got an answer! Now we get to the hard question: is it a meaningful answer? How would we know if we screwed some of that math up? This stage is critical to cementing our knowledge of the process. Our answer is just a random number unless we believe in it. So how do we test it? Here are two ways.

First, we reality-check with an artist's eye. In general, what would you expect blue blended on a gray background to look like? I would expect it to be gray-blue, at about the same brightness, but less saturation, compared to the original blue color. After we make our prediction, we compare it to our calculated result by opening a paint program and actually creating an image filled with the calculated RGB value. Yes, the color is reasonably similar to our prediction.

Second , we'll perform the actual blend process in a paint program. To do this, we create a new image filled with our original blue color, then paste it over a second gray image with 50 percent transparency (see Figure 2).

We observe the result by using the color eyedropper tool and looking at the resulting RGB values. When I did this, I got RGB of (28 percent, 25 percent, 75 percent). If we normalize our calculated RGB pixel, we get
Red: 72/255 = 0.282
Green: 64/255 = 0.25
Blue: 192/255 = 0.75
Alpha: 128/255 = 0.50

That means our RGB values would be (28 percent, 25 percent, and 75 percent). Once again, we've double-checked our calculation and found that we're on target.

Obviously, you don't need to hand-check every pixel you blend, but if you truly understand what you're doing, you would be able to check any point, work through the math, and predict the results.

All this double-checking is very important when you start experimenting with more complicated blending than a simple 50 percent blend. Unless you understand and predict your results from blending, you're just guessing, and you won't be able to control your medium.

Getting Creative

Now that you have all the basic tools to play with blending, you're ready to mess with innovative combinations. Here's how to do it. Plug various factors such as GL_SRC_ALPHA into the Sacred Formula and figure out what the results would be. Here is an additional example, taken from a recent project I worked on:

```
glBlendFunc(GL_ZERO, GL_SRC_ALPHA); // Monochrome lightmaps
```

That formula gives us this:

```
output_pixel = (source * 0 + destination * SRC_ALPHA)
output_pixel = destination * SRC_ALPHA
```

This simply darkens the destination pixel by the alpha channel of the source. The RGB values in the source are ignored. Sounds useless, but it could allow you to hide a lightmap inside a texture that doesn't need its alpha. You could ignore the alpha and apply it as a normal texture, then blend its alpha as a lightmap on a different surface.

Granted, it's not easy to figure out a really cool new application, but if you do, it should be fairly simple to implement because you're using existing systems.

Now I know you're not going to be satisfied until we address that strange factor SRC_ALPHA_SATURATE. What's it for? Anti-aliasing, apparently. Here's what the OpenGL 1.2 specification says about it:

> Polygon antialiasing is optimized using blend function (GL_SRC_ALPHA_SATURATE, GL_ONE) with polygons sorted from nearest to farthest.. Destination alpha bitplanes, which must be present for this blend function to operate cor-

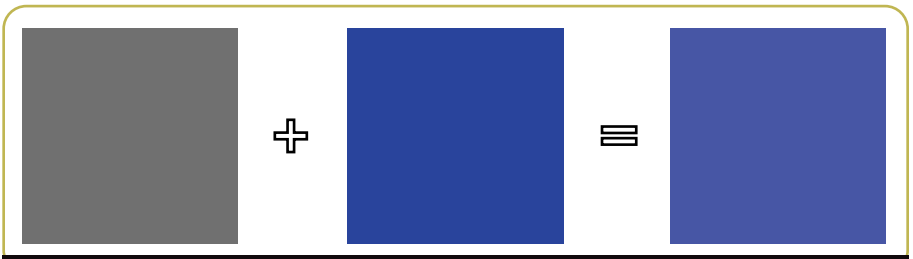


FIGURE 2. *Double-checking a simple texture blend.*

rectly, store the accumulated coverage.

What's All This For?

As you sneak back out of the belly of the beast with the knowledge safely stowed in your head and huddle in your nice artistic hobbit-holes, you may find yourself wondering, Why the heck did I bother? What can I do with this newfound knowledge?

For me, the most important reason to grasp this stuff is so I truly understand how my medium, real-time 3D, works. By knowing what's going on at a low level, I gain intuition and insight into once-mysterious problems and occurrences that I encounter. It makes my troubleshooting guesswork much more accurate.

Second, it's a power thing. I find the thought of creating my own blend modes very inspirational, and get a little wound up when I realize that I could suggest some new blend mode to my programmer, and thus actually advance the artistic front into that foreign ground of technicalities a little farther .

PREDICTING OVERFLOW. Last month, we discussed the problem of RGB limits or overflow. Pixels have a maximum brightness (100 percent) and a minimum brightness (0 percent). If we attempt to assign a value higher than the maximum, it's truncated to the maximum. This is designed into the OpenGL blending functions, as the specification states, "All scale factors have range [0,1]."

We concluded that when our pixels overflow, we lose data. There's nothing wrong with that. As long as we understand why it's happening, we shouldn't feel reluctant to have blend overflow, but few artists are aware of it.

Overflow isn't just a problem for additive (and its corollary, underflow for subtractive) blending. It can happen during the math for any of the blend modes. In fact, it's a more subtle problem for other modes because calculating the overflow isn't as simple as simply adding RGB values.

To understand and predict overflow with more complicated blending modes, we follow the same process: pick a single pixel, run the calculations (we added the RGB values in our example last month), and see if the resulting RGB values are within the 0 to 100 percent range. Of course, to do that, we'll need to know what the math is behind these blend modes. This column gave you artists the tools to actually do that yourself. ■

FOR FURTHER INFO

SGI's OpenGL WWW Center
http://www.sgi.com/Technology/OpenGL/

The OpenGL 1.2 Specification
ftp://sgigate.sgi.com/pub/opengl/doc/opengl1.2/opengl1.2.pdf

The OpenGL Site
http://www.opengl.org

by Omid Rahmat

Trends in the Entertainment Platform Market

There was a time when game developers would agonize over how much of their precious time should go into supporting the Macintosh or the PC, and over deciding among Sega, Nintendo, and Atari. During the period between 1992 and the end of 1994, the game market was in transition,

and platform issues really came to the fore. Part of this transitional period was fueled by the decline of the 16-bit consoles, the inability of 3DO and other console wanna-bes to make an impact on the market, and the uncertainty surrounding Sony's and Sega's next-generation platforms. During the same period, the industry was witnessing heightened consumer interest in the home PC, fueled in part by the promise of multimedia. Today, Nintendo and Sony tower over the console industry to such a degree that support for either platform is more of a business decision, often dictated by the developer's and publisher's relationships with either giant. The home PC market has flourished to the point that its high-end segment, the hardcore games enthusiast, is nearly the sole motivator of power systems purchases. The game development community is now in the driving seat, and platform issues tend to be more technical and resource-bound than anything to do with the fear of supporting a target machine that may not have an audience somewhere down the line

The platform has stabilized conceptually, although the vicissitudes of the PC hardware market still make for interesting research. Do I support 3D accelerated-only products? Do I go for MMX? Some might think that's about all there is to ask about the state of the entertainment platform.

Not exactly. The economics of the console and PC markets are worlds apart, and as a result of continued growth in both businesses, game developers are likely to face even more aggressive courting by the hardware and publishing powers-that-be. Everyone wants content, and while the best content makes its mark across a number of platforms, the pressure to put all your eggs in one basket is going to increase as platform vendors jockey for position by paying for exclusivity in some form or another (Figure 1).

This jockeying for position is going to take place at a macro level between the likes of Nintendo, Sony, Sega, and erstwhile allies and competitors Intel and Microsoft. Throw into the mix the desire of companies such as NEC and 3Dfx to create a 3D graphics standard, and you're pretty much up to your eyeballs in conflicting sentiments as to what constitutes a platform. Is DirectX a platform, or 3Dfx, or Sega's upcoming Dreamcast, which is based on Windows CE?

To make matters worse, the numbers for platform markets match up fairly well — in other words, you aren't going to worry so much about having an installed base of users to target with so few participants. Furthermore, it's worth noting that a PC games enthusiast is also likely to be a console owner,

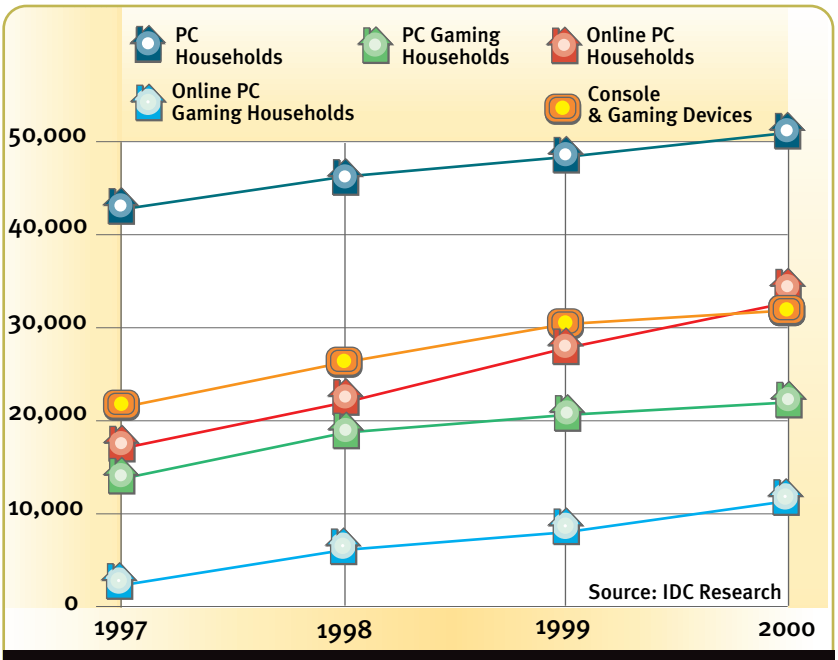


FIGURE 1. *The state of the entertainment platform.*

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.

while many general PC households, which tend to be homes with children, will also own alternative platforms because families tend to demarcate between the den PC and the television in the kid's room or the living room. Unfortunately, most of the market research on computing and game platforms tends to look at absolutes in buying and usage patterns, approaching the problem from a hardware point of view. Microsoft, Intel, Sony, and Nintendo are concerned with singular domination of the entertainment platform, although the reality is more abstract. Research on how platforms coexist is sketchy at best.

The reason that it's difficult to pinpoint or define the entertainment platform in some meaningful way based on the features of hardware or the support for a particular vendor is due to the way consumers perceive entertainment. Most of us experience entertainment, whether it's interactive or passive. We are motivated by the experiences and emotions that content creates, and aren't swayed by the promise of a platform's capabilities. There's a simpler way of looking at the entertainment platform, and one that will increasingly come to resonate with vendors of consumer goods: developers need to look at where and how people spend their money on content.

Segmenting the Platform

The segmentation of the entertainment platform is an issue that most platform vendors are loathe to deal with, preferring to concentrate on how their own platform can dominate a particular market. But as the stakes in interactive entertainment increase, so does the interest of vendors with the clout and resources to define the market. We can point to Sony's entry into the console business with PlayStation as a good example of this. The following platforms probably constitute the main targets for the games industry today:

NINTENDO. It's a mixed bag for Nintendo. N64 gets all the press, but the Super NES is still out there generating revenues, and the Game Boy is about the only significant hand-held product on the market, taking in $125 million in software sales alone in 1997 (Source: NPD). The N64 is being outsold by the PlayStation at a ratio of two-to-one in the first half of 1998 due to the N64's lack of titles and more competitive pricing by Sony. Nintendo's reliance on the more expensive cartridge storage device keeps its prices high, but developers for N64, of which there are only a select few outside of Nintendo's home-grown talent, reap extraordinary returns every time they release a title due to the pent-up demand.

SONY. PlayStation has the platform sales, internal and external title support, and the pricing. It doesn't have a MARIO counterpart, and the quality of its titles is weighed down by their sheer number. It's also less expensive to be a player in the PlayStation market. Sony is probably the only real cross-platform company in the business, having a presence in the set-top arena with WebTV, a console, and the Vaio PC business, not to mention a host of digital television and consumer electronics products. And if that wasn't enough, the company also owns a Hollywood studio and is the brand to beat in the platform business.

SEGA. Sega still makes the best arcade systems around, and the Saturn has a life of its own — just not much of one in North America. In the pipeline is Dreamcast, the Windows CE-based next-generation console. With a modem connection for the Saturn, and the collaboration of Microsoft, Sega hopes to put the game console into a more legitimate multimedia entertainment platform category — one of its own making — and in anticipation of potential rivals in the digital television set-top box category.

WINDOWS. Developers wrestle with the finer points of Windows technologies, which never quite live up to expectations. But a high-end PC, with all the 3D trimmings, is about as good a game experience as you're going to get. Still, the PC games industry suffers from a highly fragmented distribution structure. The complex value chain stalls at

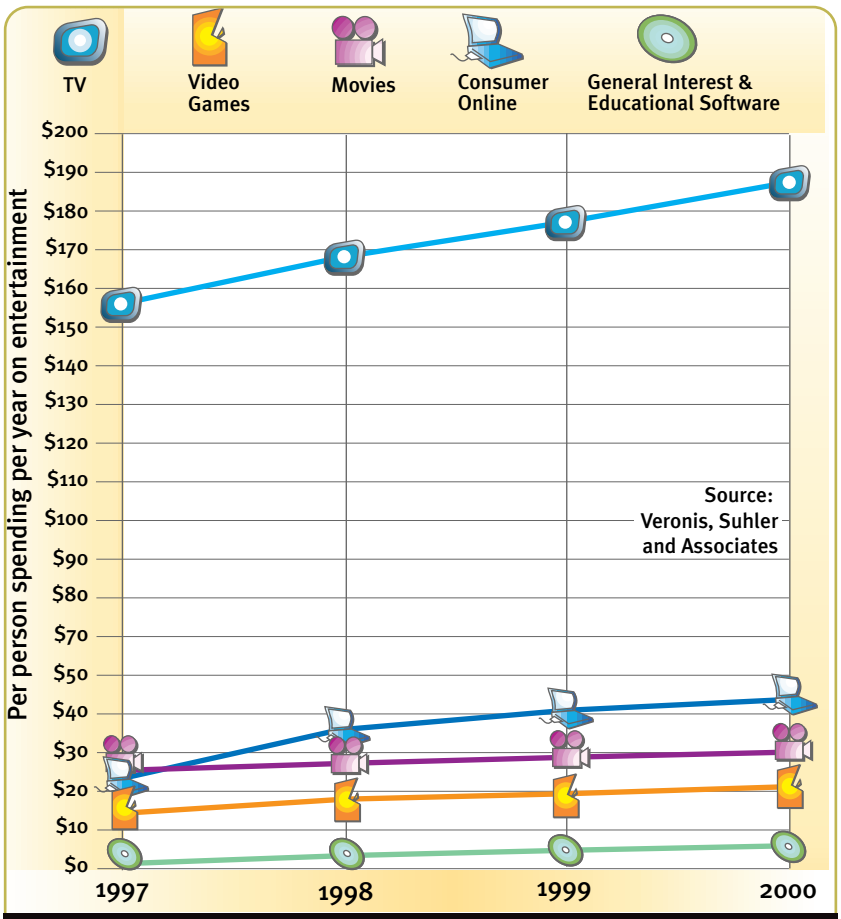


FIGURE 2. *Per person spending per year on entertainment. These figures relate to activities. Video games refers to electronic gaming in general, but not arcade. Consumer online refers to home Internet activity.*

the store-client interface every time a piece of software or hardware crashes the precious family Quicken machine, and there are no profits but for a very select few vendors. To further add to the confusion, 3D graphics vendors such as 3Dfx are trying to carve out their own subplatform category, adding to the already bewildering array of configurations in the market. Fortunately for 3Dfx, game developers have embraced the company's approach, probably out of frustration at always having to target the lowest common denominator in this market. PC games enthusiasts have plenty of money to spend on power systems (or so it seems), and they like the bleeding edge. All this favors the hardcore game mentality of the industry, and fuels intense technical rivalries between game engine developers. Some might say that today's entertainment PC is the platform that Carmack built.

Despite the existence of these platforms, the entertainment platform for games is only a small part of a much bigger picture. Television, the movies, the VCR, recorded music, and even print are all, in their own ways, competing entertainment platforms (see Figure 2).

Consumer spending on entertainment isn't dramatically growing on a year-to-year basis. It isn't a double-digit growth market. In fact, it varies and generally keeps up with inflation, or stays just ahead of it, depending on how confident the mood of society is, or how much of a need there may be for escapism. People react emotionally to content, and the measure of their reaction is how they spend the finite pool of cash that's available for their leisure.

Counting the Pieces of the Pie

As the game market continues to mature, and as the technologies that created the industry find their way into the mainstream, the entertainment platform is going to become so segmented that only highly specialized content will focus on only one segment of the market. The entertainment platform is a heterogeneous environment of PCs, consoles, and possibly set-top boxes, not to mention handheld gaming devices. The good news is that games, despite having a smaller overall audience than movies, for example, get a significant per capita spend. Compared to general software sales, games software has a significant presence in the industry.

The real threat to the platforms that games support, at least traditional games as we understand them today, is from consumer online activity. Consumer online activity is, in its own way, interactive entertainment. Granted, most online usage by consumers is devoted to e-mail, news, and information searches, but one can also argue that online entertainment is still a virgin environment, waiting to find its own style and approach. Most of the existing approaches to online entertainment have involved environments that are more comparable to television than computer games. YOU DON'T KNOW JACK is a classic example of a popular game that has successfully bridged the gap between populist game show themes from television and the more highly charged interactive environment of computer games. Does it succeed because of its obvious debt to television? Perhaps, but it does show how a smart developer can leverage content within the heterogeneous home environment.

Developers are only just beginning to realize the clout they possess. As a result, they may also be starting to get an idea of how interactive entertainment, and games in particular, fit into the big picture. Console players often grow up into PC games players. Online activity is on the up, and its entertainment potential remains largely untapped. Also looming on the horizon is the enigmatic Project X from VM Labs. Project X aims to put a powerful media processor, one capable of supporting cutting-edge 3D games, into consumer DVD players. If a solution like Project X can successfully penetrate the same market as VCRs and TVs, then counting seats is probably the last thing any developer needs to do. If you are a games enthusiast, and most developers tend to be games players as well, you'll know a good gaming machine when you see one. The trick is going to be in courting people's entertainment dollars — and not just the enthusiasts, but anyone who might be interested in games. Games in all their glory, and diversity. It's a big challenge for the industry. A very big challenge. ■

NEXT-GENERATION

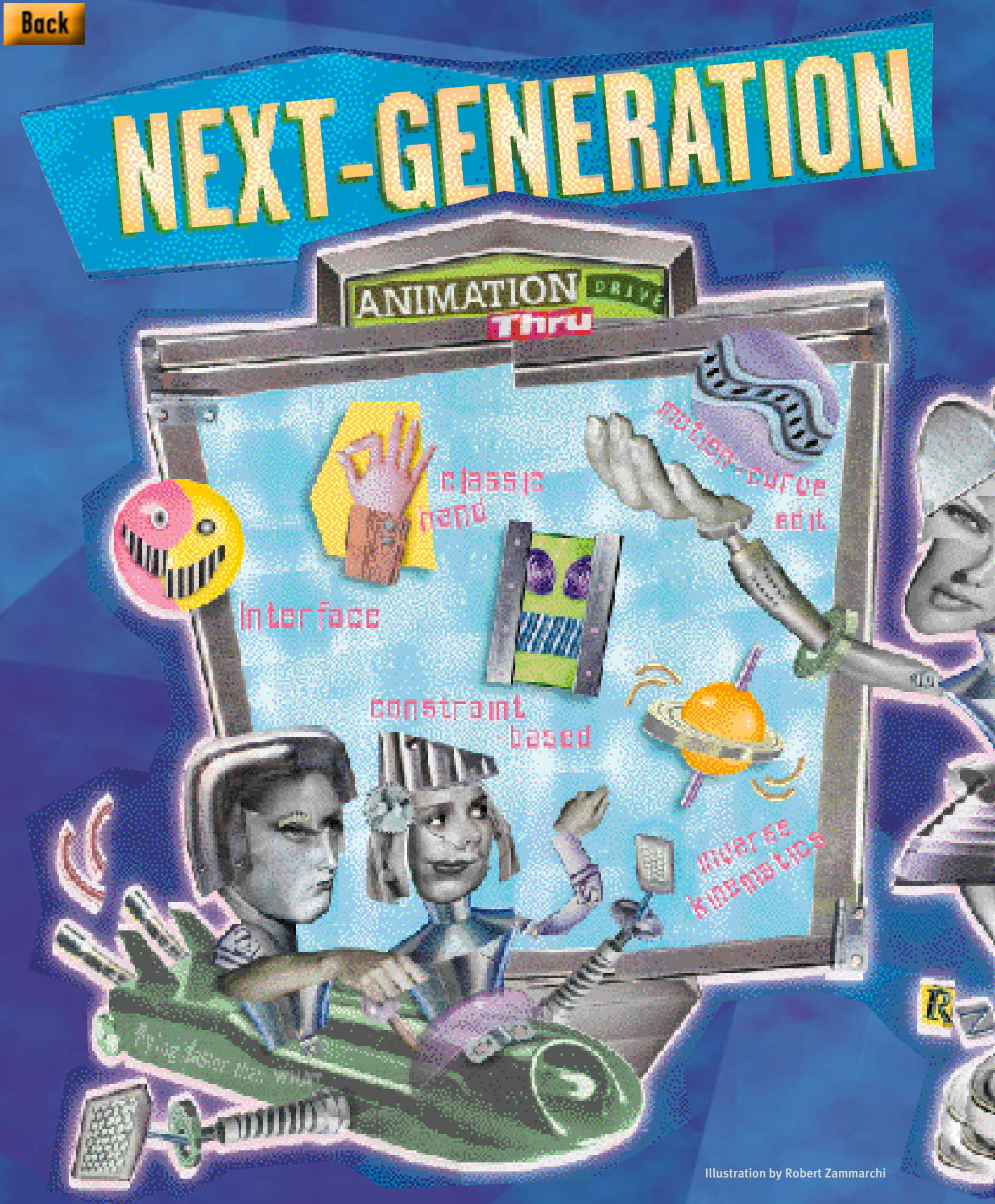


Illustration by Robert Zammarchi

CHARACTER ANIMATION TOOLS

BY MEL GUYMON

Fluid, realistic character animation is becoming the benchmark for today's real-time games. Games such as TOMB RAIDER, SOUL BLADE, and VIRTUA FIGHTER 3 have raised the bar when it comes to character motion.

As a result, today's game players are more savvy and expect better and more realistic effects. Fortunately, the tools for attaining this realism are becoming more capable, sometimes on a monthly basis. As the software gets better and better, the bar will continue to rise, and it's up to us animators to meet, if not exceed, the consumers' expectations.

Accurately deciding which animation tool to use early on can mean the difference between getting your product out by its ship date and finding yourself hitting the bricks as your team suddenly loses funding after missing it's fifth milestone in a row. With personal experience in both these categories, I can readily attest to the fact that the human element must be included when making this decision. If your game has characters — humanoid or otherwise — then it needs character animators (a touchy breed to be sure), who, aside from needing the odd bit of food tossed into their cubicles from time to time, require little maintenance if they've got a good pieceof software with which to animate.

Three premiere character animation tool vendors are releasing major product upgrades this summer. These products are Softimage's Softimage|3D 3.8, Alias|Wavefront's Maya for Windows NT, and Kinetix's 3D Studio MAX 2.0 with Character Studio 2.0. Clearly, there are other tools that have just as devoted followings as these, such as Lightwave, Electric Image, and Nichimen, but due to the fact that covering all of them would consume most of a magazine's pages, this article focuses solely on these three new releases, in the context of their real-time 3D character animation facilities.

Each of the tools in this article was evaluated on basic functionality in the following categories:

Inverse Kinemation (IK)
Constraint-Based Animation
Classical Hand-Animation
Motion Curve Editing
Interface/Ease of Use

While this is by no means an all-encompassing list, I hope it gives you enough basic information to help towards making an informed decision on which tool to use. See "Terms and Definitions" for detailed descriptions of these categories.

Mel Guymon just finished work as art director on Zombie's SPEC OPS (Panasonic).With several years experience as an animator, Mel's background includes work at Eidos, BioVision, and MSH Entertainment. Mel is currently working at Surreal Software, where he is the art lead on DRAKKAN (Psygnosis).

Kinetix 3D Studio MAX 2.0 with Character Studio 2.0

Used as the weapon of choice in the development of some of the industry's top games (such as TOMB RAIDER II, BLADE RUNNER, and DIABLO II), Character Studio is easily finding its niche within the games industry. It's the second Windows NT release of the company's character animation system, which supports motion capture, editing, and blending capabilities, plus traditional keyframe animation capabilities. The new version also has new skin deformation tools and Character Studio's footstep-driven technique. The tool has the ability to import motion capture files, and it comes with 150 motion capture files that you can edit and personalize.

Boasting over 100,000 users worldwide, MAX itself is one of the least expensive and most widespread animation tools used in the industry today. It may be surprising then to find out that it's also one of the most functional tools available and, with the addition of the new version of Character Studio 2.0, offers just as much basic functionality as it's more prestigious (and expensive) cousins.

I queried various user groups on all three platforms, and the feedback I got on Character Studio (CS) was that, while the interface left a lot to be desired, hand and IK animation using Biped was excellent. After trying out the new version of the product, I tend to agree. I think, however that the interface woes stem more from the button frenzy inherent to MAX, and less from any problems with the CS interface.

MOTION FLOW MODE. Figure 1 shows the new Motion Flow mode for CS2. I've been waiting a long time for a tool such as this, and it looks like CS2 has finally come through. In Motion Flow mode, .BIP files are combined using velocity-interpolated transitions to create longer character animation. First, you add clips and reference them to .BIP files in the Motion Flow Graph dialog. You can then select these clips to create a script in the Motion Flow Script rollout, and use the Transition Editor to adjust transitions between .BIP files. Reminiscent of Power Animator's Metacycle function, Motion Flow provides a truly user-friendly interface for splicing together different animations. The two windows in the lower left-hand portion of the screen let you load in individual sequences and overlap the transitions between them with variable parameters. In the scene in Figure 1, the Biped character is transitioning from a spin-kick animation to a dance-step anima-

Terms and Definitions

CLASSIC HAND ANIMATION. While IK- and constraint-based systems offer solutions to most animation, taking the classic approach and animating individual bones by hand can sometimes give superior results. What separates a good animator from the rest of his or her peers is the ability to combine classical and IK animation to generate nonrobotic motion. Tweaking the motion by hand is a tedious and time-consuming process, and while a smooth interface can easily double an animator's output, a bad one can freeze the process in it's tracks. The ability to easily manipulate bones and set keyframes by hand, then, is crucial in any good animating software.

MOTION-CURVE EDITING. Both classical and constraint-based IK animation ultimately generate keyframes that define the properties of an object over time. Motion curves can be generated for everything from rotation, translation, and scaling to color, surface tension, and u,v coordinates. With a good curve editor, an experienced animator can set keyframes on the first pass and finish tweaking the animation entirely by hand using the resulting motion curves. When using motion capture, animators work almost exclusively within the curve editors; it's safe to say the your ability to use motion capture is linked directly to the functionality of your curve editor. The bottom line is that a good curve editor is critical when it comes to providing animators with a smoothly flowing interface.

INVERSE KINEMATICS (IK). Probably the most important tool available to animators today, IK is a skeletal-based system that solves for joint motion automatically. For instance, if you want to animate a character's hand to swing a sword, you simply drag the characters hand to the position you want, and the IK system will solve the joint motion of the forearm, bicep, and shoulder bones automatically. It's actually as simple as it sounds, and with a little preparation setting up your skeletons, you can get remarkably smooth motion on the first pass. Even so, raw IK data is always discernable from motion capture simply due to the smooth nature of the transitions the software generates; many of the subtleties of natural motion are lost. However, IK can get most of the work done for you, allowing the animator to spend his time tweaking the motion to get a realistic, life-like result.

CONSTRAINT-BASED ANIMATION. Constraints are inherent to both classical and IK-based animation. All of us animators can remember the first walk cycle we ever generated, and how frustrating it was to try to keep the character's feet from sliding on the floor. Constraints largely remove these headaches; they can be used to limit the rotation of a joint (that is, prevent the elbow joint of a human character from bending backwards), or to glue a characters feet to the floor (preventing the "moon walking" effect). Directional constraints can be applied to keep a character's head pointing forward during a complex attack sequence, or simply to keep the character's feet pointing forward during a walking loop. Coupled with a good IK system, constraints are probably the second most important tool used by animators today.

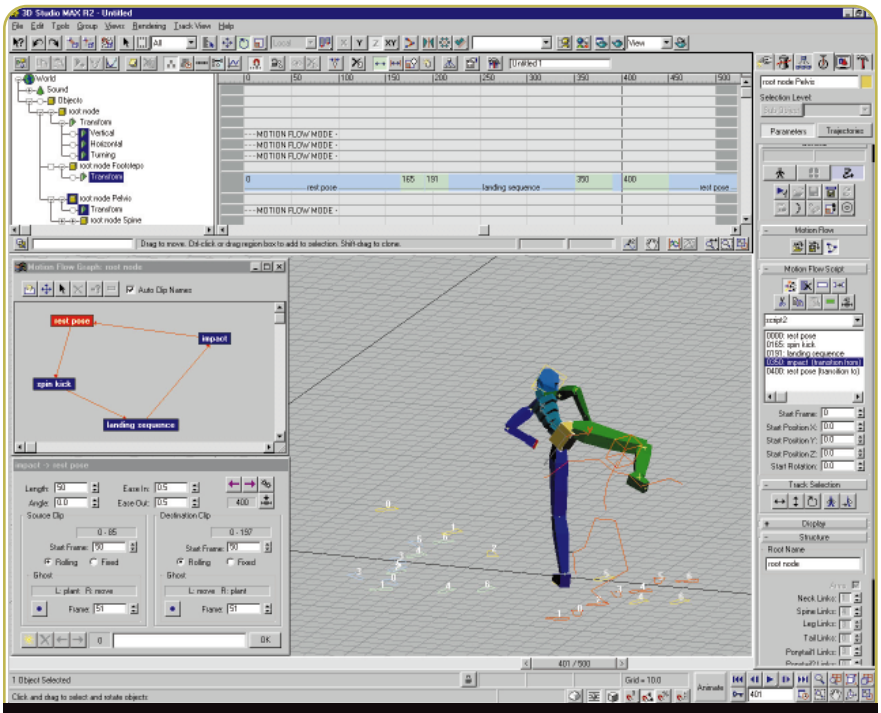


FIGURE 1. *Character Studio 2.0's Motion Flow mode.*

tion. The great thing about this is that you get a visual representation of the transition. As the time slider nears the transition point, a red stick figure appears, going through the motions for the animation to which the character is transitioning. By interactively editing the parameters for the transition, you can easily tweak the overlap to achieve a smooth result.

INVERSE KINEMATICS. Overall, MAX 2.0 has a solid IK system. I've always been a little put off by the joint-dialog interface — all that sliding up and down with the mouse gets old really quickly. But the ability to toggle IK off and on with a button is definitely a plus.

Add in CS2's functionality and it's a whole new ball game. CS2's biped feature provides an extremely easy way to set up a perfectly functional IK skeleton, with rotational constraints and expressions, giving a realistic form and balance to the skeleton. This means that if you raise a the skeleton's arm on one side, CS2 automatically shifts the weight of the other side of the body, allowing it to assume a natural pose. As a result, your skeleton's setup is pretty much done already, and it's physically accurate to boot. Although unfamiliar with the product at first, I had no trouble getting the hang of merging the footstep-based biped skeleton with a little hand-generated IK. Most of the tweaking that I usually have to do to get the correct form and balance was done automatically.

CONSTRAINTS. There are good and bad things to report on this score. Again, the interface is clunky and prohibitive, and you are limited to only two types of constraints: orientation and positional. By definition, Biped's footsteps act as combination positional and orientation constraints, allowing you the freedom to choreograph the rest of the character's motion without worrying too much about foot position or balance.

One nice facet of the program is the ability to combine constraint-based IK animation with the inherent dynamics functionality in MAX. For instance, say you're creating a death sequence for your character, and you want her to crumple to the ground like a marionette whose strings have just been cut. You simply apply motion dynamics to positional constraints set up on the characters hands, hips, and head, and your character drops like a rag doll. You can even set it up so that the hands and head bounce off the ground a few times, which is excellent.

CLASSICAL HAND-ANIMATION. This section deals largely with the interface. If it's easy to move, rotate, and scale an object, it should be easy to animate using these same basic techniques. With MAX, I found this to be the case, generally. Swapping between the various viewpoints was quick and painless, which is key when you're animating by hand. Note that MAX is set up to run off the mouse. It takes a little setting up, but with the help of MAX's keyboard shortcuts, I was able to set up a decent environment that made setting keyframes by hand pretty seamless.

MOTION CURVE EDITING. MAX's Track View Editor provides functionality comparable to the Dopesheet in Softimage|3D and the Action Windows in PowerAnimator. You can easily toggle between displaying curves for single and multiple objects, or simply display all animated objects in the scene. Changing the slope and inflection along a curve or at a keyframe is a single mouse-click away, and applying transformations such as scale and translation to entire sets of keyframes is straightforward. With MAX, you can control object rotations

3D Studio MAX 2.0 with Character Studio 2.0

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

Recommend Hardware: Intel Pentium w/128MB RAM
Tested On: Intergraph TDZ 2000
Software Price as tested: $4,000
Pros: Humanoid animation with Character Studio 2.0 is a snap.
Cons: The quirky, icon-intensive interface can be really annoying. No three-button mouse support.
Comments: Full functionality has finally been achieved in CS2, thanks to a combination of good, basic animation tools, the added bonus of a physically correct skeleton, and lots of little touches specifically tailored for character animators.

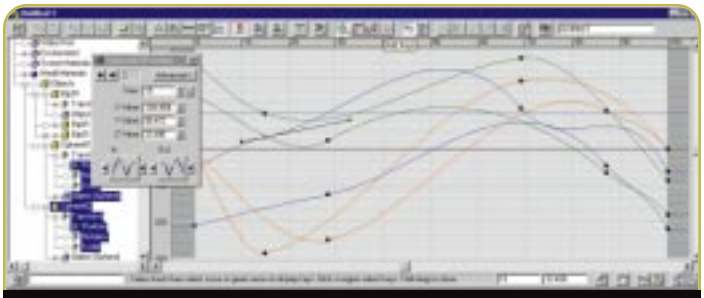


FIGURE 2. *Track View Editor in MAX.*

with several different controller types, which generally change the behavior of an object between keyframes and during transition states. I was thoroughly confused by this feature and ended up setting my defaults to Euler rotation at every step.

One thing MAX supplies that I haven't seen in other tools is the ability to interactively modify the function curves as the animation is playing back. I was able to make slight modifications to the position and slope of several keys while the animation was looping in real-time. No more starting and stopping to view the resultant modifications.

Alas, here again I found a good, solid tool that's been cluttered with a lot of "neat" extras that aren't necessarily very useful. As you can see in Figure 2, there are some thirty or so buttons just having to do with the Track View mode, and there are different keyboard shortcuts for when the mouse is in the Track View window and when it's out of it.

Overall though, it appears as though Kinetix's intent was to avoid limiting the animator choices of functionality; and I think it achieved that. I couldn't come up with any single important function that was missing from the equation, and after awhile I got used to most of the interface hang-ups that were bothering me.

INTERFACE/EASE OF USE. This is mostly a judgment call that has to be made by individual animators. When 3D Studio MAX first came out, it was vastly different from its ancestor, 3D Studio. The folks at Kinetix decided to fill up the white space in the interface with icons and buttons, which, in my view, severely clutter the working area. This is my biggest bone to pick with Kinetix. By abandoning the modular format of 3D Studio (which, like Softimage, had separate modules for modeling, animation, and texturing), it tried to fit everything into one interface. It's interesting to observe that while MAX has abandoned the modular format to allow every feature to be animated, both Softimage and Alias|Wavefront are migrating toward a more modular format, providing a cleaner, less cluttered workspace.

Another complaint I have with MAX is the lack of three-button mouse support. In my mind, the mark of a truly good interface is that you can keep your hands largely in the same spots on the keyboard and mouse without a lot of dancing around. The interface should disappear. Of all three platforms, I found myself hunting and pecking over the keyboard the most with MAX, while the combination of marking menus and hot keys in Softimage|3D and Maya allowed me to basically become one with the machine. At some point, Kinetix will realize that if you're going to force the user to rely so heavily on the mouse for input, the logical step is to support a three-button mouse.

Overall, MAX is useful. While curve editing was surprisingly fluid despite the button frenzy, I still found the interface to be overly cluttered.

Softimage|3D 3.8

Although it's probably used more widely in the film industry than by game developers, Softimage has garnered a dedicated following in the games industry. Psygnosis, LucasArts, and Squaresoft are among the many developers who have helped Softimage entrench itself in the gaming world.

Softimage's leadership in the field of character animation is widely recognized in the gaming industry. For the past several years, the mantra of the game developer has been, "Model in Alias, animate in Soft." An ever-increasing number of users are staying loyal to Softimage over time due to it's suite of solid animation and modeling tools, which fill a niche greatly needed in the gaming industry.

Of all the animation products on the market, Softimage|3D was the newest to me. However, it's fast becoming my weapon of choice for character anima-

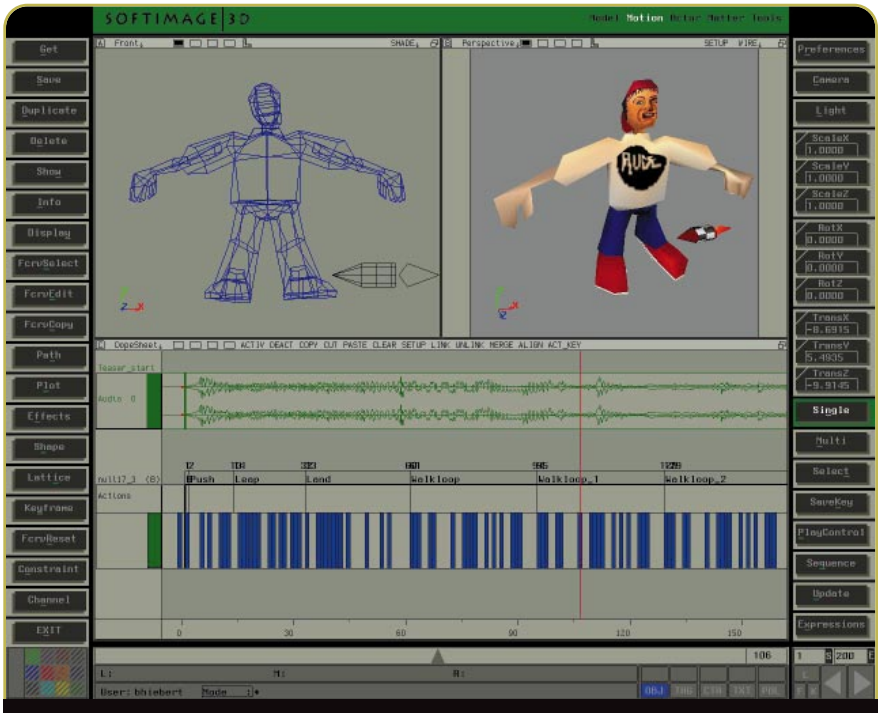


FIGURE 3. *Softimage|3D's Dopesheet and Animation Sequencer.*

tion (narrowly winning over PowerAnimator). The best part about animating in Softimage|3D is probably the most intangible. The interface simply feels right.

ANIMATION SEQUENCER. One of the new features in version 3.8 is a high-level interface for character animation (Figure 3). It allows users to work with groups of animations called "Actions," which can be managed independently. You can define groups of Actions for any character in a scene, then sequence them together on a timeline, which makes managing complex character animation much easier. Similar in function to CS2's Motion Flow mode, the Animation Sequence offers fewer options with respect to transition and overlap. In fact, at present, each Action goes in to the Sequencer in a very linear fashion, without any overlap allowed. But without a lot of extra features, the interface stays clean, allowing the work area to remain clear while retaining the necessary functionality.

In addition to the Sequencer, audio track support is now available in the Dopesheet. Similar to the Audio Track function available in 3D Studio MAX, the tool displays a waveform for the audio information, which can then be synched up with character animation to provide correct verbal cues in a sequence.

INVERSE KINEMATICS. Softimage|3D's character animation suite is built around a fundamental IK system that is as good as any on the market. Although the solution solver is not as versatile as Maya's, here again, it's simplicity is it's strength. Two basic flavors are available, the 2D and 3D chains. Skeletons can be composed of one or both types, the main difference being that where 3D chains provide IK solutions based on any axis of orientation, 2D chains provide solutions generated in a single plane of rotation. Because much of the work in IK is training your IK solver to keep only those solutions that look natural, limiting the number of solutions can help you arrive there sooner. And, because the plane of rotation for the 2D chain is itself animatable, it retains all of it's functionality.

CONSTRAINTS. In conjunction with Softimage|3D's IK toolset, users can select from myriad constraints. Besides the normal position, direction, and orientation constraints, users can also constrain bones to clusters or single points on a mesh object. For instance, say you want to animate a dragon, and you want his shoulders and pelvis to remain rigid. By creating the shoulder and pelvic joints out of polygonal objects and constraining the bones in the legs to points on these objects, you can get an amazing effect. The result is a naturally pivoting pelvic joint that mimics they way the joint works in nature.

You can even assign multiple constraints of a given type to a single object. Say you want the hips of your bipedal character to always remain equidistant between its two feet. Previously, you would have had to come up with an expression/equation that defined the relationship between the hips and the feet. Now you just assign each foot as a constraint to the hips, and the IK system automatically keeps the hips positioned between the feet. As with most features in Softimage|3D, the constraint system maintains the user-friendly tradition.

CLASSIC HAND-ANIMATION. It's no lie to say that when I sit down to animate in Soft, I'm sitting at a customizable workstation. Soft's swiftkeys allow pretty much everything to become a hot key. It's remarkably easy to set up your combination of swiftkeys so that you can keep one hand constantly on the mouse and the other hand in the same basic location on the keyboard. As in PowerAnimator, I found the perspec-

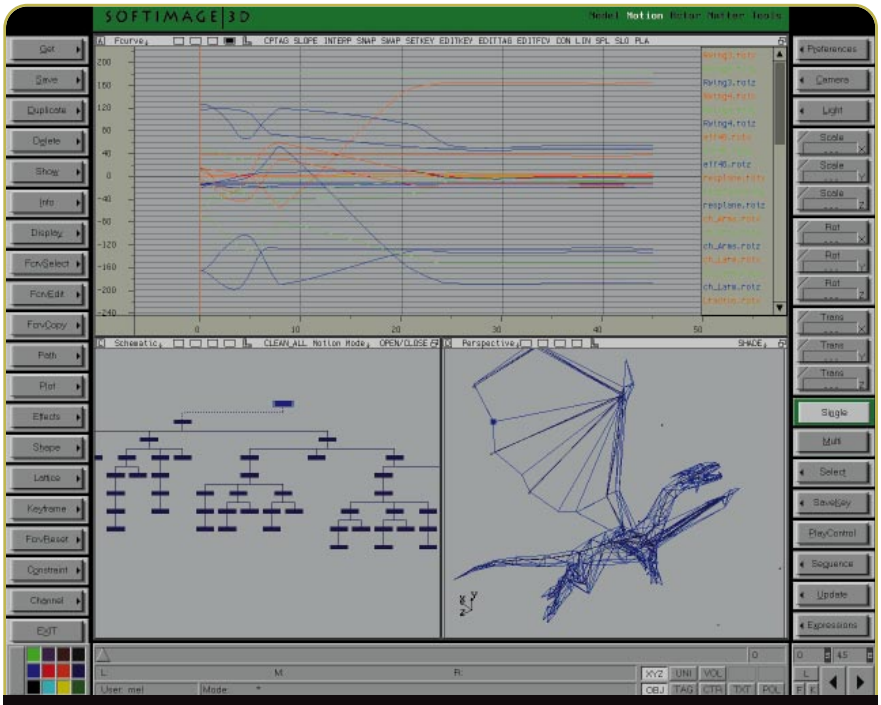


FIGURE 4. *Function Curves and Schematic Views.*

Softimage|3D 3.8

Softimage
Montreal, QB, Canada
(514) 845-1636
http://www.softimage.com

Recommend Hardware: Intel-based Pentium w/128MB of RAM
Tested On: Intergraph TDZ 2000
Software Price as tested: $7,500 for base package
Pros: Seamless, clean interface, solid IK, and basic animation tools.
Cons: Function curve editing is too simplistic, and some basic functionality is missing.
Comments: I feel this is still the best animating tool on the market for simple character animation. It has just enough tools to get the job done, without any extra fluff.

tive window extremely useful for orbiting around my characters from different viewpoints, tweaking as I went. The overall experience is very fluid. No complaints here.

MOTION CURVE EDITING. The areas I that felt were most lacking were Softimage|3D's Fcurve (Figure 4, top center) and Dopesheet (Figure 3, bottom center) windows. You can readily display all or just the selected function curves for any given object or group of objects. Changing the slop and inflection of a point is done either with a procedural effect or by moving the Bezier-like handles on the keyframes themselves. Having cut my teeth on PowerAnimator's action window, however, I often found myself trying to delete keyframes on more than one curve at a time, or trying to scale the existing curves around a point other than the origin. Neither of these actions are possible in the Fcurve window. Some of this functionality exists in the Dopesheet, but there remain a few basic tools for curve manipulation that Soft just doesn't have.

INTERFACE/EASE OF USE. Look at Figure 4; you'll notice that there are no icons. Softimage hasn't succumbed to the dreaded icon mania that is so prevalent in today's windows-based applications. The last thing you need to do at 2AM the night before a milestone is to hunt around your interface for obscure little buttons to push. Hopefully, at that point you can still read (at least phonetically), and the buttons in Softimage|3D's interface will still be readable for what they do, each one with its name clearly marked.

The schematic window (Figure 4, bottom left) acts as a functional version of PowerAnimator's SBD window, allowing you to view and alter parameters associated with object hierarchies, constraints, and animation tracks. With a fully operational skeleton in midsequence, this view tends to get a bit cluttered, but maybe that's just a comment on how an animator's mind works.

Maybe Softimage is just lucky, or maybe its developers are just good listeners. Whatever the reason, they've come up with an interface that works well. The interface seemed to disappear after just a few minutes working with the tools, and there don't seem to be that many hurdles to jump over to get to where you want to go. The bottom line is that the work area feels right.

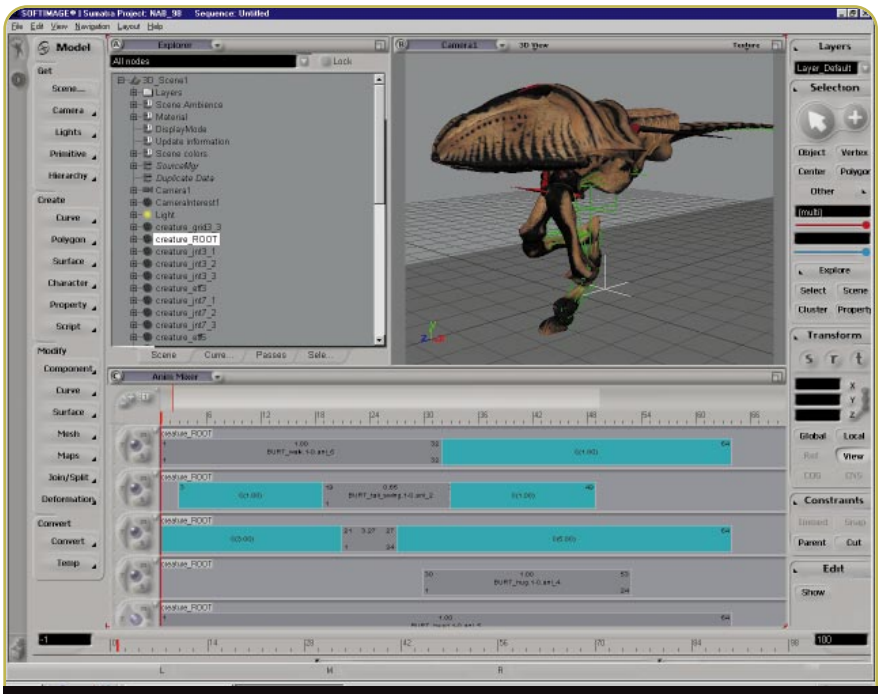


FIGURE 5. *The Sumatra interface.*

WHAT'S COMING IN SUMATRA. Sumatra is Softimage's next-generation 3D system (Figure 5), which will include nonlinear animation capabilities (NLA). NLA will allow animators to seamlessly blend animations independent of the scene timeline. For example, you can save out sequences for multiple animations and then paste them back in to form a single sequence. What is unique to NLA is that the animations will be added together graphically, allowing animators to easily view and manipulate the transitions in the curve editor. You'll have the functionality of MAX's Motion Flow Mode, with the interface clarity of Softimage|3D's Animation Sequencer.

According to Softimage, Sumatra will be have a fully-threaded architecture, which, while keeping the functional cleanliness of the current modular format, will fully integrate modeling, animation, rendering, and compositing onto a single workspace. This will put Softimage|3D on the same plane with MAX and Maya in the sense that you won't have to jump between modules to model, texture, and animate.

I was hoping to get enough information on Sumatra to be able to do a full section about it. Unfortunately, my timing was a little early. The people at Softimage told me that Sumatra will look and feel a bit different, will have several added new features, but will not lose any of the simple functionality now enjoyed by current users.

As this article goes to print, Sumatra is still several months from shipping. Until that time, the much touted nonlinear editing and seamless animation tools will have to wait. In the interim, Softimage is releasing version Softimage|3D 3.8 and Twister to prepare their customers for the new animation environment.

Alias|Wavefront's Maya NT.

Alias|Wavefront claims that in Maya, virtually everything is animatable; that any attribute of any scene component can be used to drive any other object's attributes, including position, rotation, scale, velocity, color, transparency, and so on. With a fully integrated working environment, you can set keyframes, generate path animation, and edit timing all in a single shaded and texture-mapped view.

Clearly the migration of Alias's products to the Windows NT platform is indicative of something: either to take advantage of the large Windows NT user base or simply to grab a larger

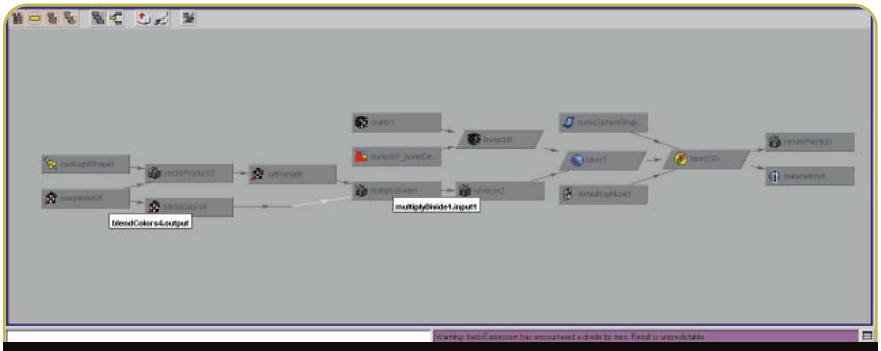


FIGURE 6. *Dependency Graph.*

share of the lower-end markets. Alias made a firm statement of commitment to game developers when it announced it's intention to ship a native Windows NT version of Maya. Full of new features and sporting a look reminiscent of an offspring of PowerAnimator and 3D Studio MAX, Maya NT appears poised to take the game developer world by storm.

MAYA EMBEDDED LANGUAGE (MEL). What sets Maya apart from other tools in the industry is its scripting language, MEL. Here's how it works: when you activate the MEL Script window, every action that you take generates a MEL script equivalent in the window. Since the entire program was written using MEL, every action that you perform has a MEL script equivalent. Say you perform a series of actions over and over, and now you want to create a button that does the same thing. In Softimage|3D or MAX, you'd have to learn MaxScript or the appropriate SDK, or depend on a programmer's talents. With the MEL Script window open, you simply perform your actions, grab the equivalent script, and drag it onto your menu bar. Maya automatically creates a button on the menu bar that contains the equivalent MEL script. Now all you have to do to execute your stack of scripts is click the button. It's very simple. If you need to modify your button's MEL script, all you have to do is drag the button into the MEL Script window, and you've got your original script back.

Compared to the poorly documented and arcane MaxScript for 3D Studio MAX, I found MEL to be far superior. What it boils down to is this: while Maya may not have every plug-in for character animation that you need, with MEL and a little patience, you can create your own plug-ins.

INVERSE KINEMATICS. One of the touted strong points of Maya NT is its powerful, multiple solution-based IK systems. Part of a continued evolution from the multi- and single-chain solutions in the PowerAnimator series, Maya boasts three different, configurable methods for arriving at an IK solution. Newest to the group, and by far my favorite, is the IK-Spline solution set. Basically, the user creates an IK chain, and then constrains the chain to a spline. As the spline is bent and deformed, the IK chain bends and deforms to keep up. It's perfect for animating long tails and neck segments, or for creating realistic motion in a whip-like tentacle.

CONSTRAINTS. Maya ships with a number of constraints for use with its IK systems. Point, Orientation, Aim, Scale, Geometry, Normal, and Tangent are all included. The real power comes in when these are used in conjunction with MEL. A major part of the design philosophy for Maya was the tiered user concept, in which a single experienced user (Technical Director) generates a digital puppet using IK, constraints, and MEL. These digital puppets can then be handed off to more junior groups of animators, who may not necessarily have the knowledge or experience to generate the required MEL scripts on their own.

CLASSIC HAND ANIMATION. Here again I think it will take some time for the interface to sink in and become widely accepted, as some of the basic functionality has become buried under new features. Still, the basic functionality is there. The Channel Box has replaced the PowerAnimator's Object Info window, displaying the properties, position, scale, and rotation of a selected object. When keyframes are set by hand, values are stored only for those objects in the Channel Box. So, for example, if you don't want to generate keyframes for an object's position, you simply remove the positional windows from the Channel Box.

Alias|Wavefront boasts that in Maya , "Everything is a node, every node is animatable, and every node can be linked to every other node." Say, for example, you want to copy the rotational information from one forearm to another. You would simply go into the dependency graph (Figure 6) and drag a connector from the rotational node of one forearm to the rotational node of the arm that you want to animate. Now both arms share identical animation.

Most of us can grasp that — it's just a modified version of cut and paste. But what if you want to do a little nonlinear editing, say to blend two animations together to get a third unique animation? Even this is possi-

Maya NT

Alias|Wavefront
Toronto, ON, Canada
(800) 447-2542
http://www.aw.sgi.com
Recommended hardware: Intel Pentium w/128mb RAM
Tested On: Intergraph TDZ 2000
Software Price as tested: $10,000 for base package
Pros: You can pretty much do everything with this program.
Cons: You can pretty much do everything with this program.
Comments: The new interface, Windows NT support, and some additional animation features that PowerAnimator doesn't have (such as a dopesheet, an interactive schematic window, and MEL scripting) make the functionality of this product astounding. All of this is partially overshadowed by the fact that to do the simple things, you have to jump through too many hoops.

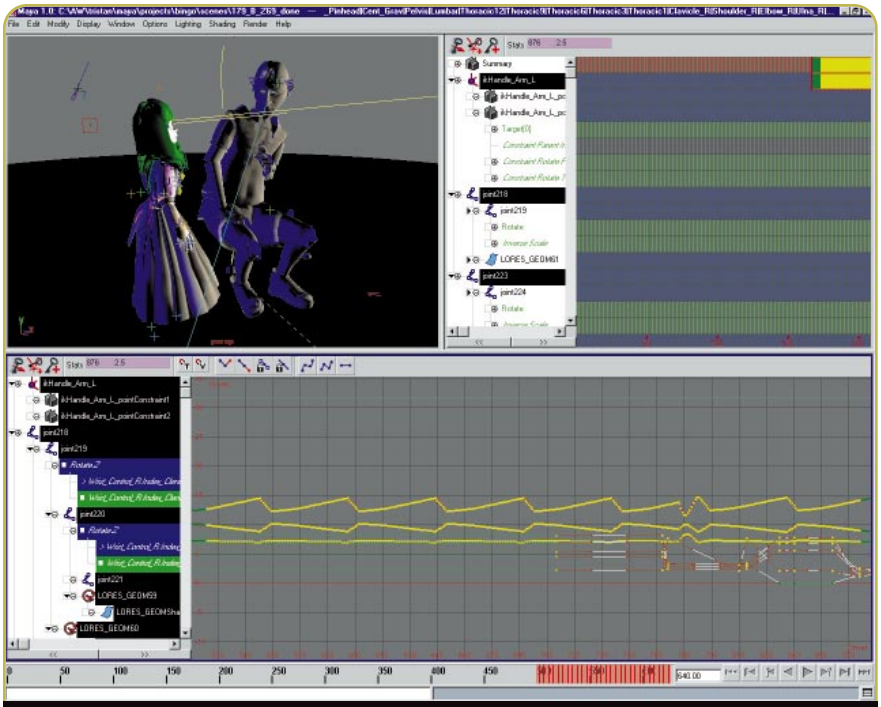


FIGURE 7. *Dopesheet and Graph Editor.*

ble. When I asked if Maya had any ability to blend animations, Brad Clarkson in Alias|Wavefront's Seattle office had a handy suggestion. By using the color blend utility node, you can link the transformation node of the first object to color 1 and the transformation node of the second object to color 2. The resulting output, which is a blend of two colors and, therefore, a blend of two transforms, is input to the transform node of object 3. There you have it; an animation that is the blended result of two completely separate animations.

Finally, Maya's expression window features a simple interface that even nonprogrammers can understand. The expressions allow you to link one attribute to other attributes using MEL scripts and mathematical expressions.

MOTION CURVE EDITING. The PowerAnimator's Action window has been replaced by the Graph Editor (Figure 7, bottom center) and Dopesheet (Figure 7, top right). The Dopesheet can also globally edit keyframes, making it fairly similar to Softimage|3D's Dopesheet. The remaining functionality that was present in PowerAnimator's Action window has been given to the Graph Editor. You can make changes to attributes in the Graph Editor and view the result in uneditable animation curves. This is useful for determining how attributes change when driven by expressions, dynamics, and set-driven key relationships.

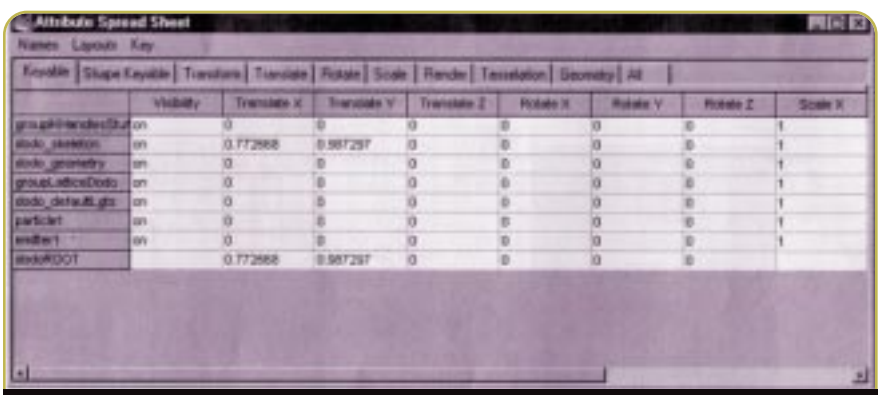


FIGURE 8. *Attribute Spreadsheet.*

INTERFACE/EASE OF USE. With added functionality, you inevitably get added complexity. This often translates into a cluttered workspace. However, Maya has striven to mitigate this complexity by using a modular toolset format. An additional display window, called the Hypergraph, allows you to examine and edit hierarchies, while the Attribute Spreadsheet (Figure 8) allows you to view and edit attributes for multiple nodes in a table layout. Potentially one of the most useful everyday tools, it's great for comparing or editing values across several nodes.

Clearly though, it's the Maya Embedded Language that makes the user interface supremely customizable. With the ability to completely customize the work area and create their own MEL-driven plug-ins, animators will soon be writing their own games without the need for programmers, producers, game designers, or publishers. (O.K., perhaps not, but the MEL Script toolset is pretty amazing!)

Maya NT promises to revolutionize the way we think of Alias workstations. With the potential for customization and the ability to run on both SGI and Windows NT platforms, Maya is affirming Alias|Wavefront's commitment to support the gaming industry, while at the same time maintaining the options for ultra-high-end graphics development.

Games such as FINAL FANTASY VII, RESIDENT EVIL, and TOMB RAIDER demonstrate the success of virtual environments populated with real-time 3D characters. As game developers try and match or exceed the standards set by these games, more and more powerful tools are required. With the increased capabilities of these three tools, in the next 18 months we will likely see a reduction in the time it takes to develop better-looking, more complex character animation. ■

Acknowledgments

Thanks to Dan Kraus, David Free, Hayley Reed, Bob Bennett, Tristan Ikuta, Alex Dunne, Alex Walsh, Franca Miraglia, Jo-Anne Panchak, Brad Clarkson, Martin Preston, Olwen Nash, Lee Sullivan, and Porl Perrot.

DirectX 6 Texture Map Compression

by Dan McCabe and John Brothers

Texture maps add visual detail to a scene without increasing its geometric complexity. However, texture memory is a relatively scarce resource, forcing game developers continually to tweak their software and artwork to fit into the limited texture memory on the graphics accelerator. Even with the availability of Intel's Accelerated Graphics Port (AGP), which lets the graphics accelerator directly access texture maps stored in system memory, bus and memory usage — as well as bandwidth — are still very limited. A solution to these problems is texture map compression, which greatly reduces not only the amount of memory that a texture map occupies, but also the bandwidth required to fetch texture data.

S3 devised a compression scheme specifically for texture maps, called S3TC, which yields benefits readily visible to programmers while maintaining the quality of artists' creations (Figure 1). Microsoft recently licensed this technology and made it the basis of DirectX 6's texture map compression. As such, this compression format should have broad hardware and software support.

Taking Advantage of Reduced Memory and Bandwidth

The speed at which texture data can be accessed generally limits sustained 3D fill rates, particularly when high-quality filtering modes, such as trilinear, are used. With a given memory or bus bandwidth, much more texture data can be read with compressed textures. The effective bandwidth with texture compression is the actual data rate multiplied by the compression ratio. So, for AGP-2x, where the maximum theoretical bandwidth is 512MB/s, the effective theoretical bandwidth with six-to-one compression is 3.0GB/s. By boosting the sustained fill rate, game performance can be dramatically improved when texturing directly from system memory over AGP or when reading a texture out of frame buffer memory. Of course, the most obvious benefit to using compressed textures is that they require less storage space. This can be taken advantage of in a number of ways.

BETTER TEXTURE RESOLUTION. Normally, texture storage requirements strain the limits of available memory. With tex-

Dan McCabe has enjoyed working on computer graphics for the last 20 years. A significant portion of that time was spent at IBM Research, where he worked on 3D rendering as well as dynamics simulation and the collision detection problem. Dan is currently a 3D architect with S3 Inc., where he is defining key aspects of the next several generations of 3D hardware. Although S3 is headquartered in Santa Clara, Calif., Dan is based in S3's Bellevue, Wash., office.
John Brothers is VP of architecture and software development at S3 and has been instrumental in the design and development of S3's recently announced Savage3D accelerator for the past two years. He is now working on future high-end 3D graphics platforms in development at S3 and is leading a great software group in putting out high performance, bug-free drivers.

ture compression, you can use higher-resolution (larger) textures, as well as a greater variety of textures at any given time. Larger textures provide more surface detail on objects and can prevent the blurriness that's often a consequence of overstretching small texture maps. Larger texture maps also allow for much more detail than would be possible without compression. And you can increase the number of texture maps in use at any one time, enabling more varied scenes.

Because the rendering surface is usually stored in memory along with your textures, you can also increase the resolution of your rendering surface with the additional memory made available by compression. Needless to say, you might not be able to increase your resolution in all aspects simultaneously, but having more memory available gives you more flexibility and options for improving your title.

MIP-MAP USE. Increased amounts of texture memory also make it easier to take advantage of MIP-maps. While MIP-maps require 30 percent more storage to house the down-sampled MIP-levels, texture compression easily frees up this extra storage. Using correctly computed MIP-maps, you can efficiently eliminate aliasing artifacts that would otherwise appear when mapping multiple texels to one screen pixel. The correct way to compute MIP-levels is with a low-pass filter, normally a box filter. Computing MIP-levels with point-sampling to get "sharper images," as one graphics chip maker has recommended, completely eliminates the intended benefit of MIP-mapping, which is to do high-quality texture antialiasing. Beware of computing MIP-levels with point sampling.

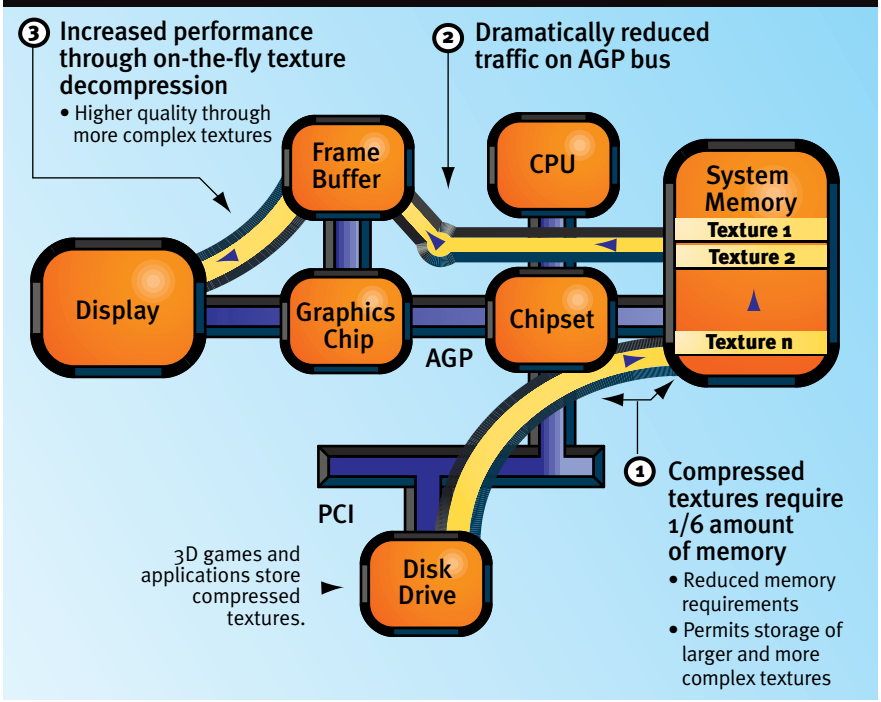
As discussed so far, texture compression can improve overall image quality without impacting performance. In fact, texture compression should actually boost performance significantly. While MIP-maps were mainly invented to eliminate texture aliasing artifacts, they also happen to increase the performance of your rendering hardware quite a bit. With MIP-mapping, texture fetches are very localized. For that reason, your renderer can use a much higher percentage of data read to generate subsequent pixels that will need texels from the same area of the texture. Also, because the texture fetches are localized, your application can read data in larger bursts with fewer page breaks in memory. Such an implementation increases the effective bandwidth over the AGP bus, system memory, or frame buffer — if the texture happens to be there. Without MIP-mapping, accesses to texture memory become random, wrecking havoc on bus and memory efficiency.

Data bandwidth, particularly texture read bandwidth, will often be the limiting factor in achieving high, sustained fill rates. Getting around these limiting factors and achieving high, sustained fill rates is what we're all really after, as high paper numbers don't do much to speed real applications.

TRIPLE-BUFFERED RENDERING. Texture compression also frees up memory that you can use for triple-buffering. If you're double buffering to synchronize buffer swaps with vertical retrace to avoid tearing, you're probably aware of the engine stalls that this method causes. Triple buffering can eliminate these engine stalls. Triple buffering can also boost frame rates, especially as frame rates increase and the cost of synchronizing buffer swaps with vertical retrace increases. While switching to triple buffering can result in a 30 percent frame rate increase, it does so at the expense of increased frame buffer requirements. But if you've compressed the texture data, you should already have this memory available.

IMPROVED SUSTAINED FILL RATE. The amount of memory that your graphics engine can transfer in a given unit of time is bounded by system design. One of the factors limiting sustained fill rates is the speed at which textures can be fetched from memory. Compressed texture maps consume less bandwidth (only 25 to 33 percent of the bandwidth required for uncompressed data) and therefore can be fetched faster. The bottom line is that compressing texture maps will result in faster rendering.

FIGURE 1. *S3 Texture compression (S3TC) is the DirectX 6 standard.*

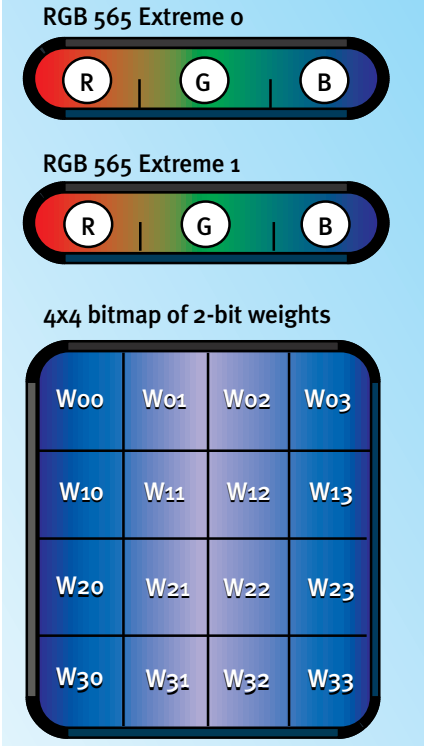


How It Works

S3TC compresses textures to a fixed size of 4 bits per texel for opaque textures or 8 bits per texel for textures requiring more than 1 bit of transparency. An image rendered with compressed textures is virtually indistinguishable from an image rendered with the original uncompressed texture maps. This image quality is the primary reason Microsoft adopted S3TC as the basis for DirectX's texture compression.

The compression scheme breaks a texture map into 4×4 blocks of 16 pixels. For texture maps with just 1 bit of transparency or none, each texel is represented by a 2-bit index in a 4×4 bitmap, for

FIGURE 2. *A 4×4 pixel block encoded as two 16-bit color extrema, and a bitmap of 2-bit interpolation weights.*



a total of 32 bits. In addition to the 4×4×2 bitmap, each block has two 16-bit colors in RGB565 format. From these two explicitly encoded colors, S3TC derives two additional colors, yielding a four-color lookup table. The system then uses these 2-bit indices to look up the texel color in this table. In total, the 16 texels are encoded using 64 bits, for an average of 4 bits per texel. The same scheme can also support 1 bit of transparency. When a 4×4 block includes transparent texels, the two encoded colors are swapped to indicate that the block has just three colors — one of the bitmap encodings (11) indicates a transparent texel. The third color is derived differently in this case. (Figure 2).

If your game makes uses of more sophisticated transparency, you can encode an additional 64 bits for transparency information in each 4×4 block. S3TC provides two mechanisms to encode complex transparency effects: it either explicitly encodes the 4 most significant bits of each pixel in the alpha channel in a 4-bit-per-pixel bitmap, or it employs a linear interpolation scheme similar to that used for color encoding. With the explicitly encoded variant, you can capture additional transparency information by dithering the alpha channel prior to truncating it to 4 bits per pixel. One great thing about this scheme is that the blocks are completely self-contained and no additional data needs to be fetched to decode the 16 texels in a block. No code book, for example, is required, as in vector quantization schemes. This advantage is important, because having to do two fetches to decode a texel is a serious performance problem. Additionally, you won't need to hassle with managing code books or palettes.

Simple Decoding Hardware

Decoding compressed texture blocks is a simple process and, therefore, very inexpensive to implement in hardware. This simplicity lends itself to very fast implementations and a straightforward approach to replicating decoding logic in order to decode multiple pixels in parallel.

S3TC's simple decoding scheme is able to achieve high-quality results by computing a linear approximation of the color space in a small block. Recall Taylor series mathematics, which states that any function can be adequately approximated over a small interval by the first two terms in the Taylor series: the constant term and the linear term. This is precisely what S3TC does in the color space of the block.

Using S3TC Texture Compression

Using S3TC texture compression in your application is very simple as it's directly supported by DirectX 6. Your artists create artwork using the same tools that they've used in the past. You perform a one-time compression as you create your distribution medium. Then, when your software run time loads the compressed texture map, a simple modification of your existing code suffices to load the compressed texture.

Although decompressing the encoded texture map is a simple matter, compressing it properly is a complex task that can be time consuming if you want the maximum possible quality. Therefore, you'll most likely be compressing your texture maps when you create them (or at least, before you place them on your distribution medium). To assist you with this conversion, S3 has made several compression tools available on *Game Developer* magazine's web site. Even if you don't use precompressed textures in your application, S3TC's fast encoder can still deliver 95 percent of the expected image quality.

If you use Adobe Photoshop to create or manipulate your artwork, you can use the S3TC Photoshop plug-in (S3TC.8BI) to extract compressed texture maps seamlessly from that graphics tool. This plug-in lets you open and save S3TC files as if they were native to Photoshop. This approach is the best one to take, especially if you're using a relatively uncommon image format for your textures.

On the other hand, if you're using an image editor other than Photoshop, you can create and view S3TC texture map files with our standalone utility, S3TC.EXE. This utility accepts a number of commonly used image formats, such as .BMP, .JPG, .TIF, and .TGA. It also allows you to create compressed texture map files with or without MIP-maps. With either the standalone utility or the Photoshop plug-in, creating and viewing a compressed texture map is straightforward and unobtrusive to your workflow.

If you don't want to compress at author time, you have several alternatives. You can compress during your game's installation, when the game is started, or when levels are loaded. These alternatives are possible, because DirectX 6 will include an API to compress your texture maps at run time. Bear in mind, however, that run-time compression isn't as fast, so we encourage you to compress off-line whenever possible. Lastly, because Microsoft will have an API to decompress textures very quickly in software, there's no danger of having problems with hardware that doesn't have built-in decompression support.

Choosing the Optimal Compression Level

Several variants of S3TC are supported within DirectX 6, depending on the level of transparency support that your application needs. Each of these formats has its own Four-Character Code (FOURCC) that you use to create the texture map surface.

Microsoft has defined five new FOURCCs. If your texture map is completely opaque, uses only 1 bit of alpha, or uses color-key transparency, you should be using FOURCC DXT1. This format is the most compact representation of the new compressed texture map formats. It lets you switch, at the block level, between fully opaque blocks and blocks with minimal transparency. Each block of 16 pixels is encoded in 8 bytes for an average of 4 bits per pixel.

If your texture maps have more complex transparency effects, you can use one of the DXT2, DXT3, DXT4, or DXT5 formats. All of these formats use an additional 8 bytes to encode transparency information, for a total of 16 bytes per block or an average of 8 bits per pixel. DXT2 and DXT3 explicitly encode alpha information by capturing the 4 most significant bits of the alpha channel. Dithering on the alpha channel can also increase the effective number of bits that are represented. You would use DXT2 when your transparency has premultiplied alpha (which Microsoft is advocating for the latest release of DirectX) and DXT3 for the more traditional nonpremultiplied alpha representation. DXT4 and DXT5 represent the transparency channel using a 3-bit linear-interpolation scheme similar to that which encodes color information. Again, use DXT4 for premultiplied alpha and DXT5 for nonpremultiplied alpha. DXT1 is by far the most useful format because it compresses down to 4 bits per texel and provides excellent color resolution and 1 alpha bit. The entire texture map must be classified by a single FOURCC code. Allocating a compressed texture map surface couldn't be simpler — do what you've been doing previously, but use the new FOURCC code instead.

Helpful for Internet-based Games

Texture-map compression is useful whenever memory size or bandwidth are an issue (all the time). One obvious application of this technology is transferring texture maps or images over the Internet. If you're creating VRML worlds for walkthroughs on the Internet, you'll want to use texture maps to provide visual detail to the objects. While the bandwidth of a local graphics system is already a concern, network bandwidth of Internet applications is an even more critical consideration. Using compressed textures in your VRML world will increase their user friendliness and increase the acceptance of VRML for your clients. And with Microsoft's Chrome project racing to completion, you can expect to see more mixing of 2D and 3D graphics on the same web page for a unified look across all graphics elements. Expect to benefit from texture compression in this environment as well.

Texture compression saves memory and bandwidth, and you'll find that taking advantage of S3TC within DirectX 6 is trivial. You can easily implement its simple decoder in hardware, so you'll likely be seeing that functionality on chips soon enough. Regardless of the hardware support in the short term, the support for fast decompression built into the DirectX 6 API makes this format a reliable solution. It will work on all platforms beginning with DirectX 6.

LISTING 1. *Handling S3TC compressed textures.*

Step 1: Load compressed texture from the file.

```
{
      DDSURFACEDESC ddsd;
      DWORD dwfilecode,dwBodySize;
      BYTE* body;
      FILE* Fp=fopen("test.s3t","rb");
      fread(&dwfilecode,sizeof(DWORD),Fp);              // Skip the filecode
      fread(&ddsd,sizeof(DDSURFACEDESC),Fp);            // Loaded the surface descriptor
      fread(&dwBodySize,sizeof(DWORD),Fp);              // Get the size of the texture
      body = (BYTE*)malloc(dwBodySize*sizeof(BYTE));    // allocate texture memory
      fread(body,dwBodySize,Fp);                        // Read the body
}
```

Step 2: Create the texture surface.

```
{
      LPDIRECTDRAWSURFACE     lpdds;
      // Assume that your DirectDraw interface is represented by lpDD
      lpDD->CreateSurface(&ddsd,&lpdds,NULL);           // Not exactly - verify

      // Fallback strategy
      // If CreateSurface fails due to lack of video memory
      // OR the DDSCAPS_NONLOCALVIDMEM flag to ddsd.ddsCaps.dwFlags
      // call CreateSurface again to create the surface in AGP memory
}
```

Step 3: Load the compressed data onto the texture surface.

```
{
      lpdds->Lock(NULL,&ddsd,NULL,NULL);
      memcpy(ddsd.lpSurface,body,dwBodySize);
      lpdds->Unlock();
}
```

Sample Code

Sample code to load and use compressed S3TC textures in DirectX is presented in Listing 1. With that code, you' re locked and loaded. Use this texture surface anywhere within your Direct3D game or application. Check the S3TC web site for more information about how to use S3TC in DirectX 6 and in the OpenGL S3TC extensions. ■

FOR FURTHER INFO

S3 Inc.
http://www.s3.com

DirectX 6
http://www.microsoft.com/directx/pavilion/default.asp

Intel's AGP
http://developer.intel.com/pc-supp/platform/agfxport/AGP_FAQ.HTM

Mo i a e 1.1: l Abo Cha ac e

by Dan Teven

We game developers are guilty of following a content formula, encouraging an arms race in hardware, and being obsessed with technical tricks. And we're in danger of burning out our audience. Once stunning graphics are taken for granted, we'll have to improve the quality of interaction in other ways. According to the Motion Factory, the missing ingredient is richer, more realistic characters (Figure 1).

Think about what it would take to create a computer simulation of yourself. With readily available technology, we could make an elaborate physical model, and we could animate it convincingly with motion capture data. We could program the character to pursue a goal and to respond to particular stimuli. If we're good enough programmers, we could even convey a fleeting illusion of intelligence, creativity, or sense of humor.

Now imagine inserting that character into a simulation of a subway station at rush hour. Will the illusion hold up? Of course not. The environment is too rich, the potential interactions too plentiful. The difficulty of managing characters forces us to set our games in controlled environments — and by doing so, we may be missing out on some intriguing designs.

The Motion Factory's first product, officially called the Motivate Intelligent Digital Actor System, makes it a lot easier to build complex characters. Motivate combines technologies used to control mechanical robots with advanced techniques for animating 3D models. Characters — both player-controlled and independent — can be assigned high-level tasks, such as "walk to the door"; the Motivate run-time engine will sweat the details.

Although Motivate would be useful for animated 2D titles, this version works with 3D characters only. Motivate doesn't limit you to humanoid or bipedal characters, but if your characters will roam freely in space, they won't be able to use the path generation feature; real-time path planning is a difficult computer science problem in just two dimensions. I guess you could say that Motivate works best with 2.5D titles.

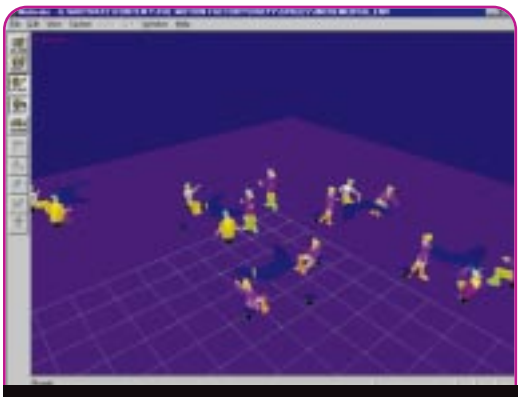


FIGURE 1. *A party scene created with the Motivate authoring tool.*

Dan Teven inflicted DOS/4GW on the game industry. For the past four years, he's lived off nuts and berries while struggling to come up with an even better acronym. He can be reached at dteven@ici.net.

Inside the Box

Motivate is a product that doesn't fit into a single niche. Its value lies in four basic areas:

- Modeling behaviors using hierarchical finite-state machines (HFSMs). This paradigm for defining player AI is very robust and scales gracefully from simple to complex behaviors.
- Editing jointed 3D models and keyframe animations. Although Motivate's Actor and Skill Editors are not its most unique feature, they are very capable.
- Real-time motion synthesis. Besides animating models with inverse kinematics (IK), Motivate can blend and overlay animations, so characters can move more realistically.
- Performing real-time collision detection and path generation. The path generation allows actors to solve simple navigational problems on their own, even under changing conditions.

The full package consists of the Motivate run-time engine in redistrib-

utable form; an authoring environment based on the engine; a Software Development Kit with the expected include files, class library, and sample code; documentation; and two days of hands-on training.

Plug-ins are used to support different video and audio renderers, to import geometry and motion data from different file formats (Figure 2), and to extend the scripting language that controls the run-time engine. You'll use the SDK to control digital actors and other Motivate classes from C++ code, and also to write new plug-ins.

Motion Factory has a multi-user runtime engine available, but I didn't get a chance to review it. Motivate should be a good fit for multiplayer games because you can use very concise commands to direct actors.

According to Motion Factory, everyone who buys the product goes through the two-day training session. Not everyone gets the same level of technical support, however; buyers can purchase three levels of technical support packages separately. Support personnel usually responded the same day to e-mail and always had an answer for me.

Motivate is hardware-locked with Hasp, so you have to attach a dongle to your parallel port before the authoring environment will start up. I hate copy protection and hardware-lock schemes because they always seem to create more problems than they solve. Sure enough, when I installed Motivate on my NT 5.0 alpha, the Hasp drivers crashed and left my system unbootable. I never got around this incompatibility; I used Motivate on Windows 98 instead.

Learning the Product

I got up to speed in the authoring environment very quickly, in part because of the training and in part because the environment is genuinely well-designed. Learning the SDK is no different than learning any other class library, so it takes time. There is a lot of interesting sample code, including examples of each type of plug-in. Project files are included for Microsoft Visual C++.

The documentation is in the midst of being made more task-oriented. The user's guide appears to have made this transition, and it's excellent; the SDK reference has not done so, and it's meager. I would have liked a more thorough overview of how the components and classes in the system interact — or at least a good index so I could be sure I hadn't missed something. I would also have liked longer discussions of error messages and of delivering a finished title using Motivate.

Behavior Modeling

Finite state machines are taught in first-semester programming classes, so they're not exactly breakthrough technology. On the other hand, using them for behavior modeling is clever, and Motion Factory has succeeded in making them both practical and accessible.

FSMs are ideal for creating characters that respond gracefully under all conditions because they force the author to plan for all possible combinations of behavioral states and events. The drawbacks to FSMs are that the number of state/event combinations can get very large and that their logic is fixed. Hierarchical finite state machines, as implemented by Motion Factory, are FSMs in which states are allowed to contain substates and procedural code can be attached to state entry, exit, and transitions (Figure 3). Also, states can contain variables (which are also visible to their substates). These properties reduce the number of states in an HFSM and allow it to adapt to changing environments. For example, you can reuse the same states for "hunting with the bazooka" that you used for "hunting with the rifle," and the transition code for the "fire" event can worry about whether you have enough ammo left — if you do, you can come back to the same state, and if you don't, you go somewhere else. (If you hit another actor with your shot, then its HFSM will handle the event; Motivate executes HFSMs for each actor simultaneously.)

HFSMs are created in the Behavior Editor using a visual interface similar to a flowchart (Figure 4). States are represented by rectangles, transitions by arrows. The hierarchy of substates is obvious because rectangles can contain other rectangles. Clicking on a state or a transition lets you edit its properties. Those properties can include procedural code written in Motivate's scripting language, Piccolo.

FIGURE 2. *The Import Actor dialog for 3D Studio Models.*

A Motivate Glossary

Let's go over a few quick definitions. A *space* is just a container for objects that can potentially interact. A space might correspond to a room or a level in your game.

Actors are the fundamental objects in a space: almost everything that happens is an actor "acting" in some way. Actors have properties such as jointed, hierarchical geometric models, skills, and behaviors. Some actors (sets or props, to continue the stage metaphor) may have no skills or behaviors.

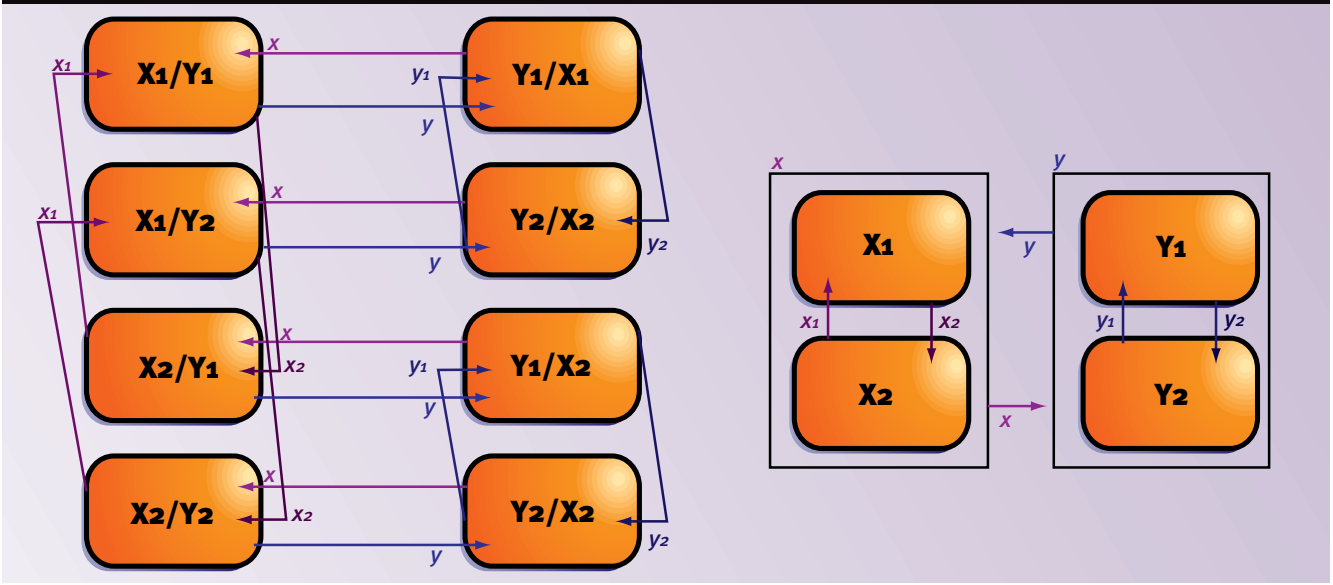
Skills are short sequences of animation describing an actor's most basic movements. Think of skills as a vocabulary; the parts of speech are locomotion (for example, go to), manipulation (grasp, touch, and place), and gesture skills. These may be combined in various ways to generate sentences (complex movements). You can also create compound skills involving more than one actor — for actions that must be tightly choreographed.

Behaviors are the "intelligence" in intelligent digital actors: goals, responses to stimuli, states of mind, and so on. Actually, they're programs implemented as hierarchical finite state machines.

Finite state machines (FSMs) are systems that can be in a fixed number of distinct states and that can change to different states only in accordance with predefined events. Because the states and state-to-state transitions are defined ahead of time, FSMs are perfectly predictable.

Inverse kinematics (IK) is a way of interpolating between keyframes in an animation. IK allows some of the nodes in a jointed model to be constrained and adjusts the positions of the others as necessary.

FIGURE 3. *Conventional (left) and hierarchical FSMs describing the same system.*



Piccolo is a typeless, garbage-collected language like JavaScript. The text editor that's built in for editing Piccolo code is very basic, but it doesn't need to be more powerful, because most of the Piccolo functions you'll write will be just a few lines long. Small functions are also easy to debug, and there's an integrated debugger that's more than adequate.

Piccolo has hundreds of predefined methods. You can view their return values and parameter lists from a help window, no in-depth information is available within the environment. Fortunately, the manuals are included on the CD in Adobe Acrobat format.

In the Behavior Editor, you can zoom in or out, so the user interface scales up to handle big HFSMs. However, the interface isn't as polished as those of the Actor and Skill Editors. It tries to cram too much information onto the screen at once, so you'll have to do a lot of scrolling in screen resolutions less than 1,024×768. There aren't enough on-screen cues to explain what all the toolbar buttons do, or what the elements of an HFSM diagram mean; a "What's this?" tool would be nice. And the few bugs I encountered while testing the product were in this area.

Despite these minor complaints, the Behavior Editor is good enough that you can build HFSMs with hundreds of states and not find yourself fighting with the tool. It's even good enough to be used by teams in which the game designer isn't a programmer. He or she could simply map out the state diagram, leaving behind comments that a programmer would eventually translate into Piccolo.

Editing Models and Animations

Motivate is not a 3D modeling tool, so you begin creating an actor by importing a model created by another package. Motivate accepts .3DS, VRML, and .DXF file formats. (You can use models in another format if you use the SDK to write a plug-in for that format.)

The models must be made of rigid links; deformable meshes aren't supported. The geometry must be segmented, so that each body part to be animated is a distinct node. Still, you don't have to define the hierarchy in advance, and you can specify the handedness of the coordinate system at load time. Motivate supports both single- and double-sided polygons; perspective-corrected, lit, and filtered textures; and opacity maps, but not bump maps or highlights (yet). Texture maps should be square and a power of two in size.

After importing the model, you can edit it. This step can include resizing the model, defining "up" and "forward" vectors, setting pivot points, constraining joint movement, and rearranging the hierarchy — up to a point. You can rearrange nodes, remove them, paste in nodes from another actor, merge two nodes into one, or split an actor into two, but you can't divide a single node into two.

The Actor Editor has two windows, with a rendered view in one and a hierarchy graph in the other (Figure 5). You can also use multiple monitors. The user interface is intuitive, with good use of

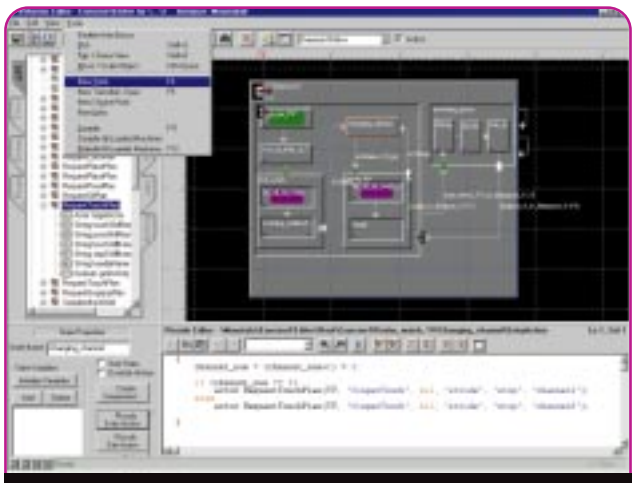


FIGURE 4. *The Behavior Editor, with a state entry action shown in the Piccolo Editor.*

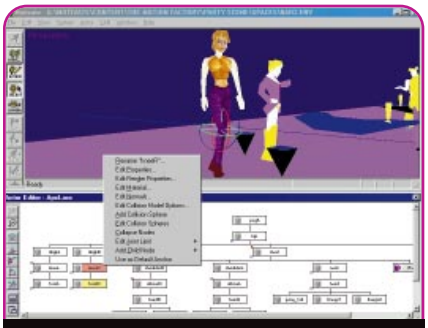


FIGURE 5. *Preparing to edit a joint limit in the Actor Editor.*

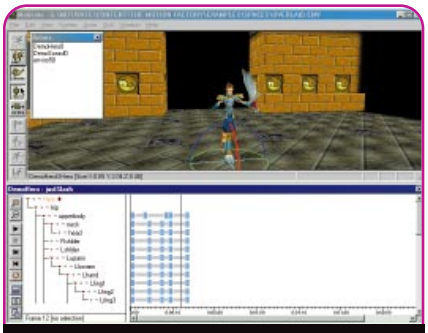


FIGURE 6. *The Skill Editor playing back a compound skill.*

context menus (right-click), keyboard shortcuts, and the ability to work in either window when it makes sense to do so. Basic operations, such as selecting an actor, are painless, even when the actor is off screen, small, or obscured.

Selecting an actor causes a transform manipulator to appear around it. The manipulator can be resized independently from the actor by pressing the [Tab] key, which helps make the manipulators unusually convenient to use. You can drag an edge to rotate an actor, or drag a handle (a "tab") to translate it. [Shift]+drag will scale the actor proportionally; [Ctrl]+[Shift]+drag will scale along just the axis associated with the tab.

To create skills for an actor, you can start from scratch, or you can import data (3D Studio animations or BioVision motion capture data). The Skill Editor is another two-window view (Figure 6). One window plots the geometry nodes against a timeline; the other lets you see the animation "live." The basic idea is that you drag the geometry into an orientation you like, associate it with a position on the timeline, and save it as a keyframe. To see the skill animated, just click the Play button or drag the time cursor back and forth to see what the actor looks like at any instant.

Like the Actor Editor, the Skill Editor is painless to use. I have next to no experience animating 3D models, but Motivate made it easy. My only complaint about the process is that Motivate doesn't come with any libraries of prebuilt skills. A basic walk skill, for instance, could be applied to any actor that used a basic bipedal hierarchy. Why should I have to create one from scratch just to get a prototype working?

Motion Synthesis

Once an actor has been assigned enough skills, Motivate is able to move the actor to account for changing goals and obstacles. This flexibility requires motion synthesis, not just motion playback. For example, if an actor has a walk skill, and is directed to go to a certain location, Motivate will loop the walk animation until the actor gets there; will adjust the animation if the actor has to climb up a set of stairs, making sure the feet touch the ground properly; and will overlay other animations, such as "chew gum," as needed. If the actor is suddenly told to run in another direction, Motivate will segue smoothly from the walk to the run.

Although Motivate makes better use of animation scripts than any other off-the-shelf game engine, and real-time IK is a powerful feature, it's still not a general solution to the problem of motion synthesis. Real-time physics simulation is the wave of the future; in another year or two, we'll start seeing games in which joint movements are computed dynamically instead of interpolated from keyframe data. Motivate lets you drive its actors with your dynamics engine, but it doesn't have one of its own.

Path Generation and Collision Detection

Motivate's path-generation capabilities are another first for a commercial game-authoring tool. The run-time engine can determine an efficient path to move an actor from point A to point B along any surface defined as a floor, avoiding both static and moving obstacles. The path finding seems smarter than that used by most games I'm familiar with, but the algorithms can take a fair amount of processor power. The computation is performed asynchronously to minimize its effect on responsiveness.

Actors can also be sent from A to B using a straight line, arc, projectile, spline, or sampled point path. Motivate uses these trajectories to provide a full complement of camera manipulation commands — and uses

Reality Check

To validate my conclusions about Motivate, I spoke with Red Orb Entertainment, one of the few companies that has publicly announced a licensing arrangement with Motion Factory. PRINCE OF PERSIA 3D is an adventure-action game set in ninth century Persia; rich in story line and character development, it should be a good fit for intelligent digital actor technology.

According to Peter Lipson, chief technologist for PRINCE OF PERSIA 3D, the team members happiest about Motivate are the animators. The lead animator was already familiar with Lightwave and 3D Studio MAX, but found Motivate easier to use. Because of Motivate, the team has been able to use animators without a lot of experience.

The PRINCE OF PERSIA team uses NDL's NetImmerse engine for rendering and Red Orb's own code for world management. They are not using Motivate's behavior management because they already had their own stuff working when they switched over. According to Lipson, the biggest integration challenge was hybridizing the collision detection code (since both NetImmerse and Motivate want to do collision detection). He called the integration "no harder than it ought to be."

Red Orb says Motivate is of higher quality than tools they could have developed in-house, and it was ready when they needed it. In short, they're very happy with the product.

its path planner to prevent the cameras from being blocked. Smart cameras are a real plus, but they're almost an afterthought on Motivate's long feature list.

For collision detection, Motivate uses three different algorithms, depending on the situation. Bounding volumes are used most of the time, with triangle-based checking used when very accurate results are required, such as when an actor moves along an irregular surface or uses a manipulation skill. You can select a hybrid algorithm (intermediate in speed and accuracy) on a skill-by-skill basis.

Integration

Motivate's diverse feature list might make you a little nervous. What if Motivate does too much, you ask? What if it gets in the way of the rendering code, or the physics engine, or the support for that cool subcutaneous joystick you saw at the CGDC?

Motion Factory admits that it had some integration problems with their release 1.0, but it has worked hard to open up its architecture since then. At present (release 1.1.2), the SDK gives you the control you'll need to adapt Motivate to your design: you can use the class library to call Motivate from your own application framework, and you can extend the authoring environment by creating eight different types of plug-ins.

The first type of plug-in is for video rendering. There are standard plug-ins for OpenGL, 3Dfx Glide, and RenderWare, with a Direct3D plug-in scheduled for this summer. The second type is for sound, although DirectSound3D is the only provided option. You can also create plug-ins for importing and exporting actors, importing motion data, extending the Piccolo language, editing custom properties, and registering callbacks for important events.

Keep in mind that the plug-ins are used to extend the authoring environment; you don't need them in order to utilize Motivate classes in your C++ code. If you use the SDK, your game can keep control of the main loop and treat Motivate like a fancy animation or AI library. You don't have to use every feature, but the interdependencies among the core components can be slightly inconvenient. For instance, animation depends on Motivate's collision detection functions; collision detection requires an up-to-date object database; and therefore Motivate needs to know about every object in the world, even those you don't intend to animate.

Performance

Compared to the cost of rendering 3D scenes, Motivate's CPU requirements are modest. However, it does need memory: the authoring environment allocated 30MB just starting up, and running a fairly basic demo took another 7MB. There is no built-in meter that tells you exactly how much memory you're using, so be prepared to spend some time adding instruments to your game to measure it. Version 1.5, to be released soon after you read this, is expected to need less memory.

And Now, the Bad News

Overall, I am extremely impressed with Motivate. The quality of the product is topnotch. Its features are genuinely useful for developing a large class of games, and you won't find them in another off-the-shelf product. It's being used for real game development, so it continues to improve (see "Reality Check"). It's fun to use.

The bad news: the price puts it out of reach of all but the largest developers.

Motion Factory offers two licensing models, each of which buys you ten developer seats for $25,000 and requires a royalty buyout of $25,000 to ship a title. In one model, the ten developer seats are for the entire development cycle of one title, and in the other, they're for the life of a single major version of Motivate. Which model you choose depends on whether you're developing several titles at once.

I won't say this product is overpriced, because it would undoubtedly cost several times as much to develop an equivalent product from scratch, but I could find a lot of other ways to spend $50,000. Has anyone else noticed that the acronym for Motivate Intelligent Digital Actor System is MIDAS?

If you're weighing a Motivate license against the cost of rolling your own, remember that you'll own whatever you develop yourself. You may be able to reuse your technology for several titles; if you do a really good job, you may be able to license it to others. On the other hand, Motion Factory *did* a really good job, and their solution is ready now. Don't forget that you'll spend considerable time developing actors, skills, and behaviors; if Motivate speeds up your character development, it will pay for itself. ■

Motivate 1.1.2

RATING (OUT OF FIVE STARS): ✪✪✪✪

The Motion Factory
Fremont, Calif. 94538
(510) 505-5151
Fax: (510) 505-5150
www.motion-factory.com

Price: $25,000 for the development kit, which includes ten developer seats. $25,000 when the title ships.

Software Requirements: Windows NT 4.0, Windows 95, or Windows 98.

Hardware Requirements: Pentium with 32MB RAM and SVGA; Pentium II with 64+MB and hardware-accelerated 3D recommended; 40MB disk space.

Technical Support: Three tiers of support available, beginning at $1,500/year.

Return Policy: 30-day free evaluation period.

Pros:

1. Hierarchical finite state machine paradigm works well for modeling character behavior.
2. Powerful animation engine: real-time inverse kinematics, motion blending, and composition.
3. A powerful authoring environment with a short learning curve.

Cons:

1. Price puts it out of reach of independent game developers.
2. Fairly high memory requirements.
3. No library of prebuilt characters and motions for prototyping.

Competitors: According to The Motion Factory, the biggest competitor to this product is the "Not Invented Here" syndrome. There are many products for 3D modeling and animation, such as Kinetix's Character Studio, but no commercial alternatives for some of the other features.

Atari San Francisco Rush: Extreme Racing

by Cameron Petty

The number of people who have had, and continue to have, an effect on the development of racing simulations at Atari is somewhat mind boggling. In March 1974, Gran Track 10, Atari's first driver, featured a shifter, a wheel, a pedal, and sound. Other notables were Night Driver and Sprint 2 in 1976 (Sprint 2 was a two-player game that was followed up by the one-player Sprint 1 in 1978). Pole Position in 1982 was actually licensed from Namco but built by Atari, and was followed by Pole Position 2 in 1983. Super Sprint in 1986 and Final Lap in 1988 round out the list, leading up to

the introduction of HARD DRIVIN' in February of 1989, which was the first truly 3D driving simulation to be seen in the arcade.

Rick Moncrief led the stalwart crew of designers that created HARD DRIVIN'. Some of these designers were members of the Society of Automotive Engineers. The result of these development efforts, as history will attest, was a large contingent of happily addicted arcade goers, who stayed that way through the release of RACE DRIVIN' in 1990 and even RACE DRIVIN' PANORAMA (with multiple, wrap-around screens) in 1991. (An entirely separate division of the company was formed to adapt the driving model and market it as a police training device. I've been told that the police forces in question reported a marked increase in successful, first-time, high-speed pursuits due to the training program.)

Eventually, Moncrief and crew, apparently not satisfied with the challenge of simulating a normal automobile, decided that their next game should feature an automobile that also had retractable glider wings. Get up enough speed down a hill, pop your wings out, and take to the air. They even had a little fan in the top of the cabinet to blow air at you when you got AIRBORNE (the name of the game). The team, known at that time as the Applied Research Group, did a fine job in the simulation, and once you learned to control it, it was loads of fun. But it suffered from two problems that proved fatal. First, it was too damn hard to fly (for which it picked up the fond nickname "Flyin' and Dyin" and was the subject of many a late night lesson in crash landing), and second, it missed out on a key trend in game development at that time: texture mapping.

At almost the same time that AIRBORNE was being tested, Atari's two main Japanese competitors in the racing game market, Sega and Namco, came out with their own entries into the 3D racing realm. DAYTONA and RIDGE RACER both stepped up the bar from the previous Japanese blit-based racing entries and featured resplendent visuals due mainly to their use of this newly emergent technology. So AIRBORNE died a quiet death, the Applied Research Group faded away, and Moncrief and some team members left Atari to pursue a more down to earth, but no less ambitious goal: creating a full-fledged, motion-platform-packing, monster-audio-blasting, driving simulation. The results can be seen now, or soon, in a number of locations. Check out http://www.smsonline.com for more up-to-date information.

I Left My Lunch in San Francisco

Meanwhile, back at Atari, gears shifted, and an internal development effort began to play catch-up to supply 3D texture mapping hardware. Two sets of hardware grew out of that effort — ZOID and TGS — neither of which ever saw the light of day. In addition, the reigns of the Atari driving simulation effort were given to producer John Ray and the "San Francisco Rush, or, I Left My Lunch in San Francisco" project was initiated to restore Atari's lost position as the king of arcade racing simulations. This is about where I came into the picture. I had just finished up work as game designer and associate producer on PRIMAL RAGE, and I was itching to get back into some 3D animation-oriented work. After a few meetings, I was accepted onto the team as

A latter day incarnation of the RUSH team, from top to bottom left side, then top to bottom right side: John Ray, Spencer Lindsay, John Geraci, Gunnar Madsen, Steve Riesenberger, Cameron Petty, Kirk Young, and Alan Gray.

associate producer and game designer. The core team originally consisted of Master Ray, a few members of the former Applied Research Group programming staff, and some art staff from another recently disbanded project called METAL MANIAX, a TGS-based, futuristic destruction derby. Marketing and sales were crying out for a DAYTONA-type game, but the team was really looking to make its own mark.

We looked at DAYTONA carefully and tried to determine why it was so much more successful than RIDGE RACER. We also tried to learn from Eugene Jarvis's CRUISIN' USA, which sold a whole lot of units for Williams by overcoming weaker graphics with its pure fun factor and a dirt-floor price point. In the end, though, SF RUSH was directly descended from HARD DRIVIN' and used a variation on the same physics model. This model not only simulated the engine and its effect on the

Cameron Petty has been alternately slaving and slacking at Atari Games since he entered the games industry in 1992. He was game designer and associate producer for both PRIMAL RAGE and SF RUSH, and has immensely enjoyed seeing Atari Games grow back into a cutting-edge endeavor. He's currently off in the woods writing a sci-fi novel, but can almost always be reached via his laptop at j.cameron.petty.91@alum.dartmouth.org

wheels and, thus, the tire patches, but it also tracked the reciprocal forces back up through the drive train. This model led to some key audio developments and enabled the sort of realistic force-feedback steering that made HARD DRIVIN' famous in the first place.

RUSH took a long time to produce — almost two and a half years. For the programming and hardware staff, much of that time was spent trying to bring up a new hardware system and create tools for it, or port between platforms. Alan Gray led the programming effort, focusing on the physics model. In the latter months of the project, John Geraci lent some key help with drone AI, among other things. Jim Petrick, Betsy Bennett, Forest Miller, and Dave Shepperd also contributed to the programming, and there were tools contributions and assorted other efforts from several programmers from other in-house teams (Bruce Rogers, Steve Bennetts, and Terry Farnham). Pete Mokris designed a new, cost-reduced force-feedback mechanism that provided nearly the same performance as that used for HARD DRIVIN' at a fraction of the cost. The hardware team is too long to list, but Andrew Dyer and Steve Correll at Williams in Chicago made key contributions.

Positive Developments

SF RUSH is, without a doubt, the most realistic simulation of San Francisco that's ever been done in a game. That's not to say that there wasn't a large dose of artistic license taken in the layout of the tracks; after all, a fun race definitely takes precedence over an authentic simulation in the arcade. Still, there were a few key elements of the production that stand out as noteworthy and contributed to the success of the game.

1. SOLID CONSTRUCTION TOOLS. The art staff and the programming staff worked extensively with folks at MultiGen. We needed a version of their MultiGen II plug-in Road Tools that generated a data structure which could be adapted to work with our driving model. As far as I know, the folks in the Applied Technology Group had built their tracks for HARD DRIVIN' by placing each polygon individually in 3D space. The scale and variety of the worlds we envisioned for SF RUSH would have made this approach prohibitive, so Spencer Lindsay, who had worked with MultiGen on METAL MANIAX, pushed through the effort to adapt Road Tools for our purposes. MultiGen had developed real-time simulation databases for the military, so the company's tools were right up our alley in terms of generating a data structure optimized for real-time polygonal display. At the time, MultiGen II was one of few software packages available that let us view our texture-mapped geometry in real time, almost exactly the way it would appear in the game. The art staff was using mainly Indigo 2 workstations, which were upgraded to the Indigo 2 Extreme at some point, and one Onyx with a Reality Engine graphics head on it. Incidentally, this was before MultiGen II ran on any platform other than Silicon Graphics. Additionally, each of the artists had a Macintosh Quadra running Photoshop 3.0 and other utilities, and a PC running 3D Studio R4, both of which were used almost exclusively for texture creation.

2. GOOD CHOICE OF SILICON. In 1995, parent company Time Warner sold Atari Games to WMS Industries. This sale provided an tangential advantage to 's development. At the time, Williams happened to be working with 3Dfx, a small start-up that had splintered off from Silicon Graphics. In a combined effort between Atari Games and Williams, the 3Dfx graphics chipset was integrated as a daughter board into a proprietary development system known as "Phoenix." Later, the 3Dfx chipset was worked into a smaller, less expensive board solution for production. The 3Dfx chip gave us access to a number of nifty tricks, including vertex shading, two sets of (animatable) texture coordinates, MIP-mapping, and bilinear interpolation.

3. LOVE THOSE LODS. We made some of the most extensive use of levels of detail (LODs) in the game community to date. RUSH was, and still is for that matter, one of few games to create an environment with a naturally expansive feel. One of the art team's mandates was to avoid having geometry pop into existence out of a void, without having to resort to a fog or other obscuring artifact. We also wanted to have reasonably detailed geometry immediately surrounding the track, however, which created a resource conflict. All of the textures in SF RUSH were drawn directly from the city itself via a perspective-correct lens on a 35mm camera. Used in conjunction with a scanner and Photoshop, this approach gave a sense of gritty realism to the environments. We wanted to have flower bushes, trees, and window boxes along the road as players jumped their cars over the length of Lombard Street, but we also wanted to let players see out over Coit Tower to the Bay at the same time. We wanted players to be able to look down the entire length of Market Street, but if they were to stop and look down a side street, they would see another vista, or at least an alley. All this, while maintaining a decent frame rate, which we defined as 30Hz, was no easy task. The solution, we found, was to extensively exploit the use of LODs.

MultiGen was once again the tool of choice (and still is, for that matter, with Creator) for its ability to implement LODs, another concept that grew out of the military simulation industry. Everything in SF RUSH has multiple LODs, and all of the LOD switch ranges are finely tuned to create a sort of animated facade. Geometry is switching in just around the corner and right under

the player's nose in SF RUSH, but they're unlikely ever to notice it, unless they look very closely or someone points it out to them. The fact that SF RUSH is a racing game aided our efforts in achieving this effect. Following a race course limits the number of routes that a player is likely to take through the city, which consequently limits the number of angles/speeds at which you can approach objects/locations in the game. Also, we spent a lot of time rebuilding sets of LODs that were too polygon heavy, in order to maintain the frame rate once the final hardware was available. In the end, an awful lot of hand tuning and elbow grease was required to get right, but I think we were able to create a good sense of expansive spaces without sacrificing too much detail.

4. SWEET, SWEET MUSIC. I've heard reports that the musical selections for the consumer releases of RUSH, which were taken directly from the coin-op version, were not appreciated by consumers. I have to apologize to all the people who feel that way, but we did that (almost) on purpose. The entire team was of the opinion that the most important thing for the game aurally was quality of the sound *effects* as opposed to the sound *tracks*. That meant that the engine sound was paramount, closely followed by wind noise, road rumble, a proper Doppler shift effect for other cars, and reverb (for tunnels and canyon-like city streets). The sound tracks were relegated to whatever time and resources remained after implementing the effects, which is why the music on an optional switch in the cabinet, and the default setting is no music at all.

The intricacy of the driving model made it possible to create an engine sound that was true to life. The torque and load parameters from the engine were used to drive an audio model that then acted upon a series of samples taken from various automotive sources. In-house audiophiles Gunnar Madsen, Chuck Peplinski, and Todd Modjeski teamed up with contractor David Riesner and the Atari Industrial Design team (Mark Gruber, Ralph Perez, and Pete Takaichi). They produced a quadraphonic sound system design for the cabinet, rounded out with a seat mounted sub-woofer, that would do justice to the game's detailed audio effects.

The one thing that really puts the SF RUSH experience over the top turned out to be something we hadn't anticipated: the audio. The audio, in combination with the rest of the elements of the game, increases your level of immersion in the experience. The audio experience is very evident in the game when you get air going over hills and off jumps. The combination of the realistic physics model and a full-weight car going well over 100 MPH makes for long jumps in which the car seems to float. Perhaps due to these intense physics, there was always a sense of disconnection from the car when it was jumping. Then we added the road rumble, got the seat-mounted sub-woofer working, and actually linked the road rumble to the car's position on or off the ground. It's an extremely subtle effect, and is more felt than it is heard, but when a player goes over a jump and the grinding rumble beneath him or her turns to a coarse whooshing sound, it really sells the fact that the car just went airborne. The audio guys, naturally, wished they had a better audio hardware with more resources to put towards the audio effort, but I think they did a fine job with what they had, given our goals.

5. CENTRALIZED PLANNING. When I first joined the team, design meetings were being held in conjunction with status meetings for the entire team and weren't particularly functional. I was the new kid on the block, and despite my best efforts, the meetings always degenerated into separate groups. Everyone argued and brainstormed energetically, but never came to any conclusions either. Can you say, "Dilbert?"

This disorganization went on for a bit until a certain key member of the team threatened to be off about his business if there wasn't a change, and at his suggestion a core design team was formed. The core team was composed of John Ray as producer, Alan Gray as lead programmer, myself as game designer, and the art lead, whomever that happened to be at any given time. I suppose it's easy for me to say, because I was included in it, but I don't think anything would have ever gotten done if we hadn't implemented the core team design meetings. Also, we made it clear that intelligent feedback and suggestions for alternate solutions were more than welcome from the rest of the team. We needed to establish initial priorities, however, and assign short-term tasks while starting to map out what was going to be a huge effort. To me, it was at this point that we actually started making a game, as opposed to developing the underlying technologies that would make a game possible.

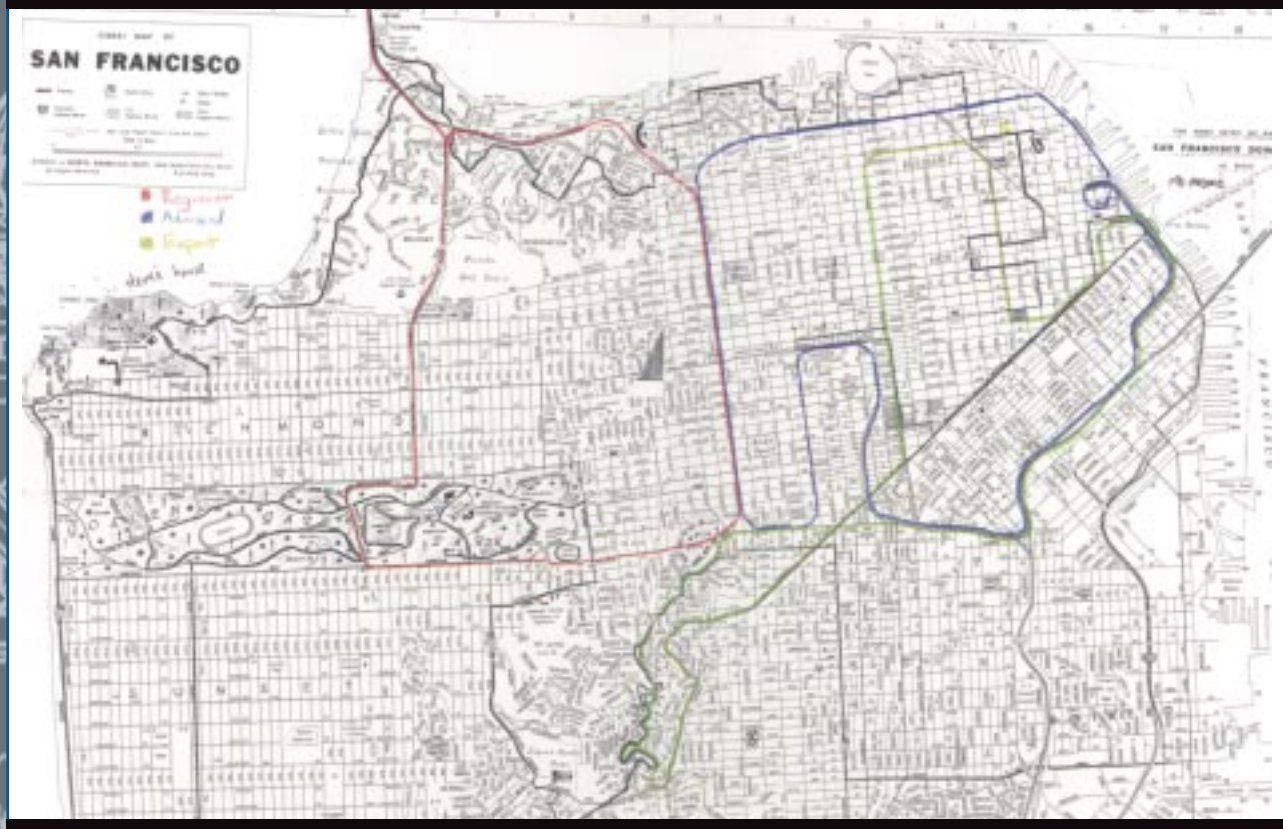
Stumbling Blocks

In spite of the fact that we finished SF RUSH on time, and that we achieved nearly all of the goals that we set for ourselves, we did encounter some significant hurdles. In general, however, we were able to learn from our mistakes, and we turned most of these impediments into advantages.

1. MOVING HARDWARE TARGETS. The development of the hardware progressed slower than we had anticipated, and the hardware itself was slower than we had hoped. Think about it: we were building some of the first consumer-level 3D hardware. The RUSH art effort, in particular, faced the inevitable problem of trying to hit a moving target by creating graphics for a hardware platform that kept changing.

The production hardware came in

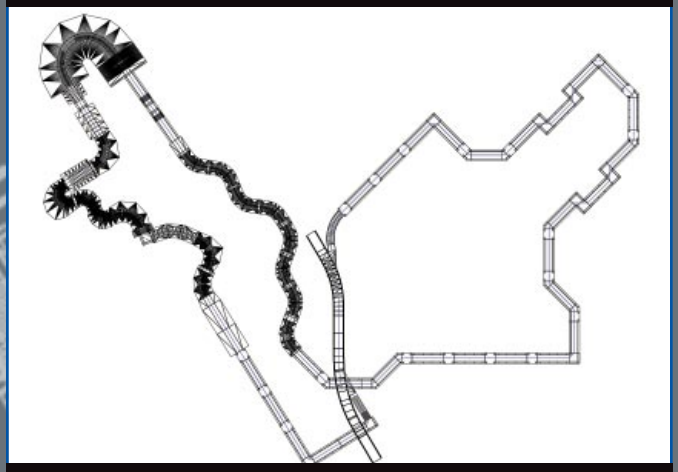
The following five images are a series showing the evolution of Track 3



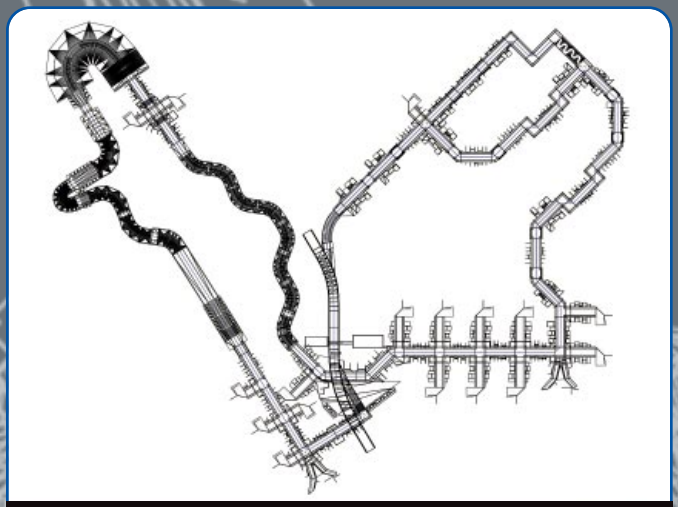
The original city map, with potential routes drawn in colored pencil

A hand drawn route interpretation with topographical info for Track 3

The following three images show three top-down orthographic views of the track during production, with all objects showing their highest LOD.



Just the road surface.



Half decorated

The final product.

two flavors: "Seattle" was a single texture-memory unit (TMU) version (used on MACE and GRETSKY 3D HOCKEY), and "Flagstaff" was a two-TMU version of that board that also included the Cage audio hardware, a proprietary audio board that provided 16 channels of 16-bit sound. Switching hardware gears in midproduction was a bit of a mixed blessing; we had to port the physics model to the new platform and revamp the art tool chain. In the end, however, the new hardware turned out to be just what the doctor ordered. Once the port was done, Alan Gray was free to work on the game and underlying technologies, and the hardware effort, focused in Chicago, was in hands that were devoted entirely to that pursuit. At this point, we devised a new schedule based on the availability of the new hardware, which was six months away, and the crunch began. We eventually met that schedule, thanks to some serious help in the eleventh hour.

2. LEADERSHIP LACKING. The RUSH art effort suffered from the art team's lack of a strong leader. Initially, this task fell to Michael Prittie because he was the most senior of the group. Michael was a fine artist/modeler/animator, but lacked the technical background to lead a cutting-edge, real-time 3D effort. Next in line was Spencer Lindsay, who was definitely the technical art lead throughout the project. At that point, however, Spencer wasn't ready to assume the duties of managing and scheduling the rest of the art team. For a while, Michael and Spencer tried to divide the lead duties between them, which really didn't work.

As a result of all this confusion, Rob Adams, who was in charge of texture production and 2D work for the game, was, for the most part, left to his own devices. Rob was a talented artist, and he produced a plethora of textures. However, there was minimal organization of these textures into a library, much of the modularity of the overall texture set had to be rethought, and the project required a global color balancing. Rob wasn't modeling worlds until late in the project, and as a result, wasn't properly aware of some of the implications that our mapping methods (for example, separating building tops and building bottoms so that the textures could easily be combined into a variety of buildings, or tailoring the house and building bottoms to the predefined hill angles that we were using to model the tracks) should have had on his texture development. The discrepancies between Rob's work and our mapping efforts represented relatively small problems, but precluded handy solutions to the daunting task of modeling three-and-a-half-miles worth of city streets while trying to avoid too much repetition. The lesson to be learned from this set of circumstances, in my opinion, is that everyone on an art team should do both modeling and texturing, as the two are closely linked in today's 3D games. In fact, Rob's texturing skills improved when he began modeling in earnest, and he turned out to be an excellent modeler as well, a much-needed help in the latter stages of the game.

Eventually, towards the end of the project, we decided that I should take over as art director. I was brought onto the team as game designer, but I had just finished a blit-based game, so I was initially discounted as a 3D artist. I quickly became frustrated, though, at designing tracks on paper and watching over Spencer's shoulder as he built the road surface for the first track. With the team's permission, I began working the night shift so that I could use the Onyx to learn MultiGen II and proceeded to model the road surface for the second two tracks. When the track surfaces were done and the game design was in a fairly stable state, I went on to start modeling scenery for the tracks as well. At this point, I began to realize that the texture library needed to be rethought, and it was the resolution of this issue that convinced the team to let me give it a go as art director. This reorganization was only a few months before the end of the project. We were behind on most fronts at that point, but we were prepared to take a fresh look at things and push through. Upper management saw things differently, however, and so the face of the art team changed again in the eleventh hour, necessitating an application of sheer labor towards meeting a deadline.

3. Corporate chaos. Along with a series of lay-offs in late 1996, upper management at Atari eliminated the position of Director of Animation, formerly held by Tom Capizzi. They also decided that since we were behind, Tom should take over as art director for the RUSH team. I'm sure Tom would be the first to admit that he received his direction on how the game should be finished from myself and the other RUSH artists, but I'm the first to admit that the project could never have been as polished a final product without Tom's help. Tom took care of the cabinet graphics, logos, and attract movies (with Greg Allen and Brent Englund on the video shoots), and furthermore put together a subteam to finish up the cars. Tom contracted Kirk Young and chain-ganged Jeff Shears and Gene Higashi from another team to finish the car effort, while the rest of the art team concentrated on finishing the tracks and select screens. Tom also had the dubious pleasure of inheriting a big organizational and relational mess, and I am eternally grateful to him for taking that mess off of my back just as I was hunkering down to hoist it up; but in the end, it all worked out.

4. The game's difficult learning curve. The biggest design flaw with RUSH was that, despite our best efforts, its learning curve was still a bit steep for a portion of the arcade audience. Driving a realistic car model through the streets of San Francisco at extreme speeds is just plain hard to do. We wanted players to be able to get good at it, but we also wanted the casual player to be able to play it and not be scared off. We tried to address this problem in our design with two major tactics. The first was a smooth progression of the skill level required for each of the tracks. Players can drive Track 1 by just putting a foot on the gas; a player in the Beginner car can pretty much go around the track without steering. Which brings us to the second tactic: the cars were divided into four classes, going from the Extreme, which is the full simulation, to the Beginner, which has serious training wheels, with a smooth continuum between the two. By the time a player has mastered Track 3, which actually requires braking (or at least taking a foot off the gas) to get the best times and can finish the course without crashing in the Extreme car, he or she has spent a lot of time and a lot money feeding his or her addiction.

The problem lay in the fact that too many people chose the Extreme car when they ought not to have done so. A player can choose Track 3 and use the Beginner car and still not have too bad a time of it, but a beginner who chooses the Extreme car on any of the tracks is in for a rough ride. One of the last revisions to the game featured graphics and sound cues designed to make players aware of the dangers of the Extreme car. This tactic may have helped somewhat, but John Ray contends (and I agree), that we should have put access to the Extreme car on a secret button combination. Secret button combinations are commonplace Easter eggs in arcade games these days, and we used them for other player configurable aspects of the game successfully. In the end, we decided, unwisely, not to use one for access to the Extreme car. If we had actually limited access to the Extreme car, we probably could have prevented a certain percentage of players from being scared off by the difficulty of the game.

5. Rushed design. The only other major problem we had with the design of the game was a lack of final tuning. I was so busy scrambling to build tracks that a few fine-tuning issues slipped through the cracks, despite an excellent testing crew. I'm not going to elaborate on those, but suffice it to say that it's possible to cheat a little in rare instances with the initial release of the game, and the drones can be kind of evil sometimes.

Finished Rush cabinets rolling off the line in Waukegan, Ill.

Pushing Boundaries

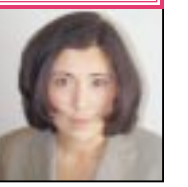
Now, I don't want you to get the impression that the art effort on SF RUSH was a complete fiasco. That was decidedly not the case, and though it may have been a sprawling mess some of the time, it was successful in the end, and we managed to push a couple of boundaries along the way. Many elements contributed to the success of the game design, but in the end, I think the interplay of two main elements distinguish RUSH from other racing games of its kind and create a unique experience. The first is the combined sense of realism gained from the realistic physics

model, the force-feedback steering, the surround sound, and all the little touches. The strength of the car and the layout of the tracks let players perform some incredible, decidedly unrealistic stunts. This style of game play, combined with a sense of tactile realism, creates a positively surreal experience that can be quite a rush, so to speak.

Since SF RUSH was released in December of 1996, sales of the game have exceeded 10,000 units (a large number for a deluxe, sit-down, coin-op game), while sales of consumer versions of RUSH have topped the 500K mark and continue to climb. We were fortunate enough to have Ed Logg — the man who created ASTEROIDS, CENTIPEDE, and GAUNTLET, and converted WAYNE GRETZKY'S 3D HOCKEY to the N64 — volunteer his team to do the N64 port for Christmas 1997. The in-house consumer RUSH team not only made the sprawling beast run properly under the N64s limited resources, but also managed to add new graphics and a portfolio of player adjustable effects (wind, fog, and so on) and options (tag, secret keys, and others). At the same time, the coin-op team was working on an update to the arcade version (SAN FRANCISCO RUSH: THE ROCK). This release would feature new tracks, as well as incorporating a new set of cars that were done by some of the MACE team artists (Jeremy Mattson, Patrice Crawford, and Matt Harvey). The face of the RUSH team changed once again not long after the game went to production. Steve Riesenberger, Aaron Hightower, Rick Gonzales, Garret Jost, and Brian Davis have all joined the team, and everyone but John Ray and Spencer Lindsay have moved on to different pastures. Both the consumer and the revamped coin-op RUSH teams are currently hard at work on two separate RUSH sequels.

Not long after the production of the original game, I moved on from the coin-op RUSH team. Tom Capizzi had already left the company to join Rhythm and Hues in Los Angeles, so Spencer Lindsay became art director again for RUSH: THE ROCK. Spencer had learned some valuable lessons over the course of the project — as we all had — and was well prepared to take up the reigns. Meanwhile, I was off on vain attempt to make something other than another racing game. I obviously failed miserably, as I've been working on nothing but another racing game for the last nine months. But that's another story… ■

by Susan Lewis



Crossing the Chasm: Tips for Startup Studios

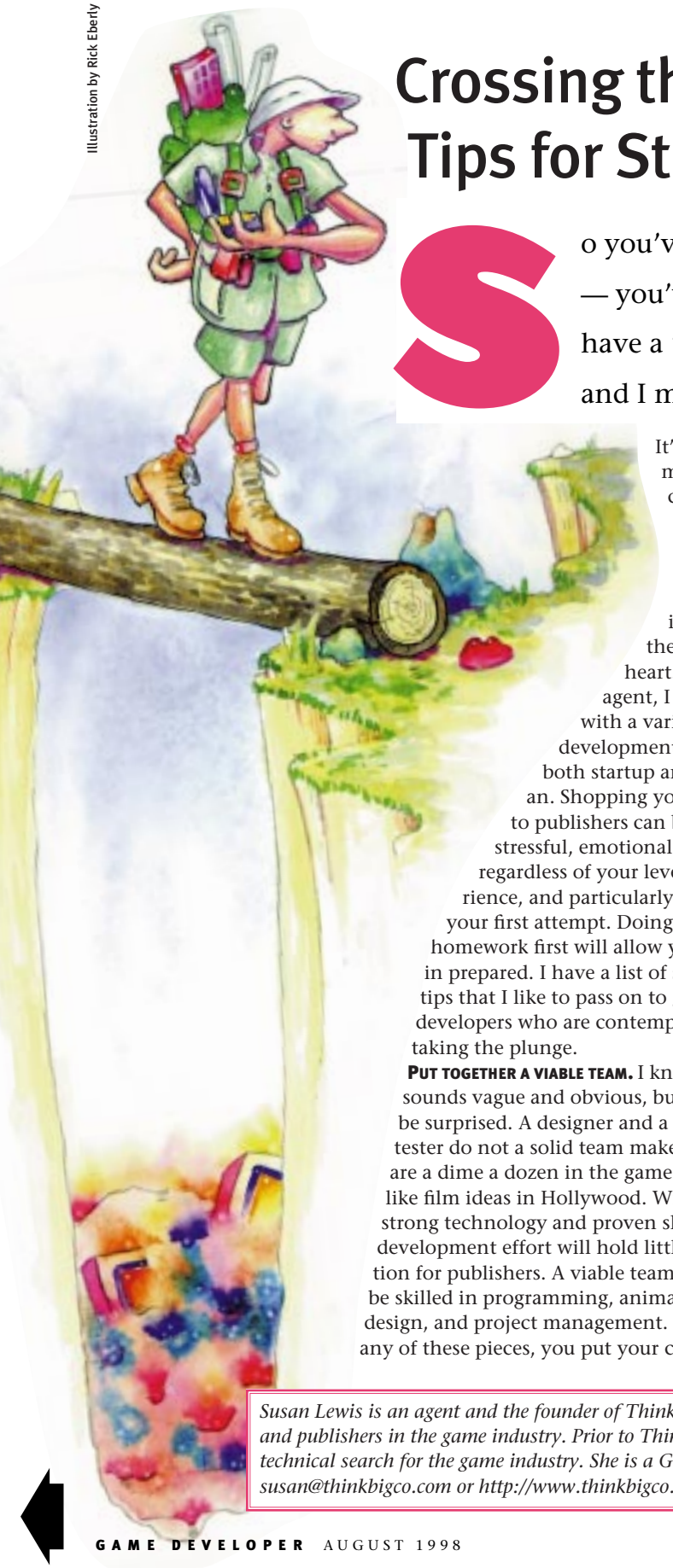


Illustration by Rick Eberly

So you've decided to split and do your own thing — you've gathered some industry cohorts and you have a "great idea for a game." Congratulations… and I mean that.

It's a gutsy move in a chaotic, often cut-throat industry, and it's not for the faint of heart. As an agent, I work with a variety of development groups, both startup and veteran. Shopping your game to publishers can be a stressful, emotional process, regardless of your level of experience, and particularly if it's your first attempt. Doing your homework first will allow you to go in prepared. I have a list of specific tips that I like to pass on to game developers who are contemplating taking the plunge.

PUT TOGETHER A VIABLE TEAM. I know that sounds vague and obvious, but you'd be surprised. A designer and a former tester do not a solid team make. Ideas are a dime a dozen in the game business, like film ideas in Hollywood. Without strong technology and proven skills, your development effort will hold little attraction for publishers. A viable team should be skilled in programming, animation, design, and project management. Without any of these pieces, you put your chances of getting a deal in serious jeopardy.

WRITE A DETAILED DESIGN SPEC. I cannot tell you how thrilled I am when a developer comes to me with a fleshed out design document; to a publisher, this shows you're serious and have thought the concept through. Publishers get rather concerned when the developer seems to be designing the game "off the cuff" during an initial meeting. I recently met with one rather enterprising developer who actually wrote a technical design document, along with the design spec, *before* meeting with the publishers.

DEVELOP A PROTOTYPE. A game concept, on its own, means very little in this industry. Publishing executives and agents are hit with loads of concepts every week, and without technology to back ideas up, your chances of landing that contract are slim. Unless you just jumped ship from the latest hit title, few publishers are willing to risk the necessary millions on an unproven team without a core technology. The best prototypes are fully interactive, allowing the user to explore a bit of what the game world is expected to look like. The final game rarely resembles the prototype, but it gives publishers a feel for what the core team is capable of putting together in a relatively short period of time.

KNOW YOUR PUBLISHER. Set up meetings with only the most appropriate publishers, so as not to waste your time or theirs. In other words, it makes little sense to begin your tour by pitching a PC game to a console-only publisher. Likewise, some publishers are visibly boutique in their title line-ups; it would be highly unlikely for a publisher only interested in PC military simulations to cut a deal on a 3D platformer for the PlayStation.

Continued on p. 71

Susan Lewis is an agent and the founder of ThinkBIG, which provides international representation to developers and publishers in the game industry. Prior to ThinkBIG, Lewis spent several years specializing in executive and technical search for the game industry. She is a Graduate of Brandeis University. Susan can be reached at susan@thinkbigco.com or http://www.thinkbigco.com.

Continued from p. 72

DO YOUR RESEARCH AND DEVELOP A SOLID PITCH. Make it easy for the product development people to explain your concept concisely to the other executives and marketing people. Everyone hates to think of their game as a "me too" title but if marketing is going to predict how the game will sell, they need to compare it to everything else out there. Take GRAND THEFT AUTO, for instance — indisputedly a highly original game. Yet, one might pitch it as "MICRO MACHINES meets A.P.B. meets SYNDICATE WARS (meets Quentin Tarentino…)." LEGAL CRIME could be pitched as "GRAND THEFT AUTO meets SIMCITY meets CAPITALISM." ARMY MEN equals "COMMAND & CONQUER meets *Toy Story*." You get the point. You might think that your game is unlike anything anyone has ever seen before, but when you leave the meeting, the producer or business development person you just met with has to explain your game to the rest of the company. Make it easy for them. And be prepared to discuss how comparable games have done in the past and to predict how your game will compete in the market that your game will face 16+ months from now.

BUILDING A BRAND, SEQUELS. Very few publishers have succeeded in developing major brands. Electronic Arts, Hasbro, and Mattel have arguably been the most successful in this endeavor, and various publishers have succeeded in churning out tremendously successful sequel lines (the DOOM/QUAKE series, MORTAL KOMBAT, and FINAL FANTASY, to name a few). Most publishers are keen to leverage the money they spend developing version one into derivatives, sequels, add-ons, and ports. What is the long term potential of your game, and how does the publisher garner long-term profits?

TELL THE TRUTH. Be honest when discussing how much time you'll need to develop your game, how much it will realistically cost to complete, and how long your team will take to ramp up to begin the project. If you make unrealistic guarantees, most publishers will quickly recognize this. And even if you manage to convince them to sign the deal, nothing prevents them from backing out later. Keeping a deal can be more work than getting one signed, particularly if you've given the publisher unrealistic expectations of your abilities.

FINANCIAL STABILITY. Publishers get nervous when they know you're hanging on by a thread, financially. This means that if you slip on a milestone and they delay payment, they run the risk that you'll go out of business. If a publisher has already put hundreds of thousands of dollars into a product, you have them over a barrel. And don't think of this as a way to gain leverage; remember that the publisher owns the game you're developing and can, in many cases, take the project away from you and assign it to a more dependable developer. Having your staff and the equipment that you'll need in place makes you more attractive to a publisher. If you come to them without any company infrastructure in place, they'll know that a sizeable portion of your budget will be going to setting up your company (hiring, computers, furniture, and so on), and that the publisher for your second project will benefit from the money that could have gone into the first.

UNDERSTAND THE RELATIONSHIP BETWEEN DEVELOPER AND PUBLISHER. Understanding deal structures and publisher-developer dynamics keeps you from looking naïve (and hence, vulnerable) to the publisher. Know what a realistic development budget is and understand that this budget is an advance on royalties, not a cost borne by the publisher; and with that, know what a realistic royalty structure is for a first-time developer. Bear in mind that the developer does not receive any back-end royalties until the development budget has been recouped by the publisher. The sad fact is, depending on the advances that you received to develop the game and the royalty rate that you negotiated, it would not be unheard of for your game to have to sell in excess of 250,000 or even 300,000 units before you received any further money.

GET IN FRONT OF THE CHECK-SIGNER. This point should be obvious, but locating the bottom line at a publishing company is never easy (and sometimes impossible) for a startup developer with few or no connections. You want to meet with the decision-maker or someone close to them. It's a painfully subjective business, and if an inexperienced submissions coordinator fails to see the value in your game, the powers that be will never see it. Again, these are connections that take time to develop, but when the opportunity to pitch your game to the executives rises, go for it and don't be intimidated. If the game is as good as you think it is, they should *want* to make the time to meet with you.

Getting a publishing deal is a daunting task. As a startup company, be prepared for a lot of rejection. In an industry where typically only the top 20 games make money, where fewer than 10 percent of all games released sell over 100,000 units, and where the average development budget has risen to $1.5 to $2 million, publishers are understandably cautious. Why is your game going to hit the top 10? What distinguishes you from the hundreds of other groups approaching each and every publisher every month? Who is your target audience? How can this be leveraged into future titles? Why will the consumer buy your game instead of TOMB RAIDER 3, QUAKE 3, FINAL FANTASY VIII, RESIDENT EVIL 3, MADDEN '00, and so on, and so on, and so on? Do not take a meeting with a publisher if you are unable to answer these questions. ■