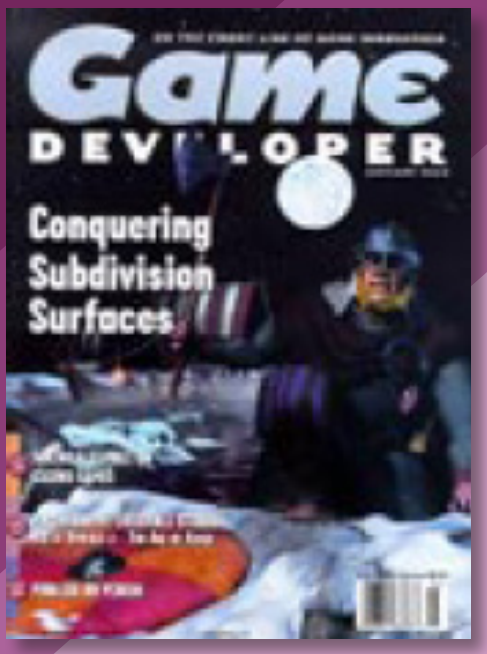
gd

GAME DEVELOPER MAGAZINE

JANUARY 2000



UBM Tech

Betting on the Industry's Future

Our industry has, by recent accounts, overtaken movies and spectator sports in terms of annual revenue in the U.S., but there's another form of entertainment that's still far ahead of us: gambling. The gambling industry is so huge, in fact, that in 1997 (the last year for which data was available) it dwarfed the annual revenues of videogames, movies, spectator sports, recorded music, and theme parks *combined*. That year it took in $50.9 billion, according to the recently released findings of the National Gambling Impact Study Commission (available at http://www.ngisc.gov). And it will likely continue to grow.

One of the fastest growing sectors of the gambling industry is "convenience" gambling, which currently rakes in about $9 billion per year. This sector comprises stand-alone wagering machines like video poker, video blackjack, slot machines, and video keno, which are collectively known as electronic gambling devices (EGDs). I suppose the growth in EGDs over the last decade shouldn't come as much of a surprise, because more and more states have turned to gambling as a way to bring in revenue and keep tax rates down, and gambling machines require little administration. Consequently, EGDs have cropped all over the place — gas stations, airports, convenience stores, supermarkets, bars, and other places where people have money in their pockets and time to kill. Because of their growing prevalence, we decided it was time to take a look at EGD development in this issue.

Currently the EGD market is somewhat peripheral to our "traditional" videogame market. Most EGDs are still mechanical, but increasingly they're going all digital, and some companies like Silicon Gaming are starting to market machines that feature 3D graphics and full-motion video. (Go ahead and chuckle to yourself that EGDs are a bit behind the technology curve compared to videogames, but just remember who takes in more money.)

What intrigues me is the potential for videogames to influence EGD design. At some point in the not-too-distant future, I believe casinos and other controlled gambling establishments — those from which children are restricted — will seek out EGDs that feature more immersive game play than today's simple, traditional poker and slot machine games. If that day arrives, the opportunities open to videogame developers will widen considerably. Game development skills might be in high demand from the likes of Bally, Williams, Silicon Gaming, and Casino Data Systems, and shows such as the GDC and E3 might begin catering to EGD companies.

Of course, the future of EGDs rests largely in the hands of the federal and state governments, which are currently in the process of evaluating the long-term effects of gambling on individuals and communities. As for the EGD market, the NGISC report noted that convenience gambling was not as beneficial as other forms of gambling, in that it provides "fewer economic benefits and creates potentially greater social costs by making gambling more available and accessible." The commission recommended that states stop authorizing convenience gambling establishments, and even advised shutting down existing operations. Whether these recommendations actually are adopted is an entirely different matter, but EGD makers may face further regulation.

Fortunately, within casinos EGDs face little regulatory threat, and that is where the most interesting opportunities exist. (Because children are banned from casinos, games can be developed that appeal to the adolescent in all of us without concern for luring minors into the dark world of gambling.) It's these locations that hold intriguing potential for more exotic, creative EGD designs. Who knows, someday we might see a descendant of one of today's hit titles in a game cabinet at the MGM Grand, spitting out coins to a little old lady wearing a muumuu. ■

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

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

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

EDITORIAL

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

ADVERTISING SALES

National Sales Manager
Jennifer Orvik e: *jorvik@mfi.com* t: 415.905.2156
Publisher Relations Manager
Ayrien Houchin e: *ahouchin@mfi.com* t: 415.905.2788
Account Executive, Eastern Region
Afton Thatcher e: *athatcher@mfi.com* t: 415.905.2323
Account Executive, Western Region
Darrielle Sadle e: *dsadle@mfi.com* t: 415.905.2182
Account Executive, Northern California
Dan Nopar e: *nopar@mfgame.com* t: 415.356.3406
Account Representative, Silicon Valley
Mike Colligan e: *mike@mfgame.com* t: 415.356.3406

ADVERTISING PRODUCTION

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

MILLER FREEMAN GAME GROUP MARKETING

Marketing Director Gabe Zichermann
MarCom Manager Susan McDonald
Junior MarCom Project Manager Beena Jacob

CIRCULATION

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

INTERNATIONAL LICENSING INFORMATION

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

Miller Freeman
A United News & Media publication

CEO/Miller Freeman Global Tony Tillin
Chairman/Miller Freeman Inc. Marshall W. Freeman
President & CEO Donald A. Pazour
Executive Vice President/CFO Ed Pinedo
Executive Vice Presidents Darrell Denny, John Pearson, Galen Poss
Group President/Specialized Technologies Regina Ridley
Sr. Vice President/Creative Technologies KoAnn Vikören
Sr. Vice President/CIO Lynn Reedy
Sr. Vice President/Human Resources Macy Fecto

BPA INTERNATIONAL

DSP Pokes Holes in Terrain Generation

Kai Martin's article "Automatic Generation of Large-Scale Terrain Models" (October 1999) overlooked some important issues. Basically, this was an article describing a signal processing operation; but when we use the fundamental ideas of signal processing to analyze Martin's techniques, flaws are revealed. Following the article's advice can lead to wasted artist hours and a poorly-performing game engine.

Figure 2b of the article shows a bitmap that has been "scaled up." Each pixel of the original image has become a square block of pixels in the big image. This is not the correct way to enlarge an image. Programmers often think that pixels are square, but rather pixels must be treated as infinitesimal sample points of a signal. To enlarge a bitmap correctly, you must "resample" it. Usually this is done by interleaving the pixels with zero values, then applying a low-pass filter. This filter is carefully chosen to limit the output to the frequency content of the original signal.

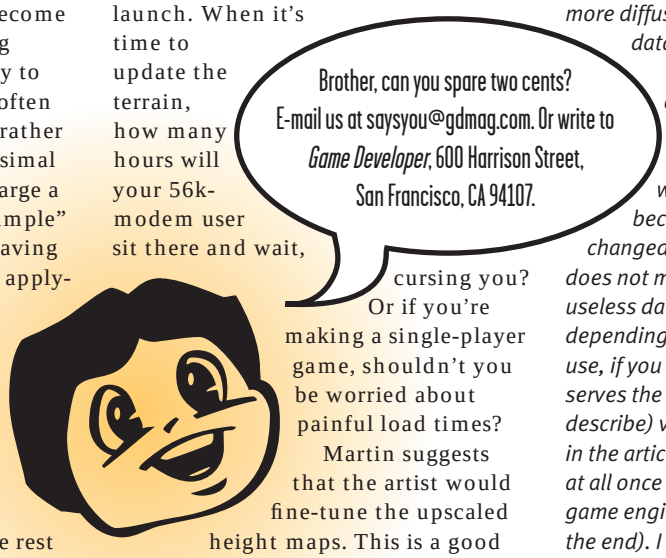
If the height map were properly scaled up, the result would be smooth and 100 percent faithful to the original. This would eliminate the need for the rest of the article. Instead, Martin performs the incorrect scaling, then notes that the result is unpleasant. He devises ad hoc ways of eliminating the noise created by the erroneous scaling. By filtering with big gaussians, or by fitting loose splines to the terrain, he smooths the image severely; this destroys its high-frequency information.

When you know a signal's frequency range, the Nyquist Sampling theorem tells you at what resolution to store it. By smoothing excessively, Martin reduces the terrain to a frequency band that can be represented by a much lower resolution. Most of the memory used to store the image is wasted.

The map designer's effort is wasted, too. Suppose he draws the terrain as a series of 256×256 blocks. You pass each block through a system that enlarges it to 2048×2048, adds noise, then smooths it to an effective resolution of 140×140. Seventy percent of the artist's work is wasted (256×256 is 65,536; 140×140 is 19,600). Furthermore, the final image uses 214 times more memory than is justified by its information content, as (2048×2048)/(140×140) is about 214.

Because of excessive smoothing, the height values created by the artist will not survive to the output. Peaks and valleys will melt. Many trial-and-error attempts will be required for the artist to achieve what he wants. With a proper upsampling of the input, the peaks and valleys would be reproduced exactly.

To the end user, that 214-times-memory-bloat is going to hurt. A persistent-world online game requires the ability to be auto-updated after launch. When it's time to update the terrain, how many hours will your 56k-modem user sit there and wait, cursing you? Or if you're making a single-player game, shouldn't you be worried about painful load times?



Martin suggests that the artist would fine-tune the upscaled height maps. This is a good idea, but it doesn't require the maps to eat so much storage. Instead, each 140×140 map should be resampled to 2048×2048. Then the artist makes his changes. The engine subtracts the original image from the touched up version, yielding a "delta image." This delta image is 2048×2048, but it is extremely compressible. Any modern compression algorithm would make it very, very small. Meanwhile, the input is stored at its original size, 140×140. During game play, when it's time to load a block of terrain, the engine adds the two images. Thus storage requirements are shrunk dramatically.

I would like to have described these issues without jargon. But regardless of specific technical issues, the most important point is this: Today, knowledge of signal processing is important to game programmers. Tomorrow it will be more so.

Jonathan Blow
Bolt Action Software
via e-mail

AUTHOR KAI MARTIN RESPONDS: *If you read carefully, you'll notice that I did point out that using the convolution filters I described in the article does decrease the amount of local detail (that is, they change the original image values proportional to the size of the filter):*

Page 50: "...choosing the size of your convolution mask will depend on the amount of filtering you need and level of detail your final image requires."

Page 51: "...there is a trade-off between losing local detail from the original height bitmap (since smoothing reduces noise by spreading it over a larger area, making it more diffuse) and generating more terrain data from a bitmap of given size."

You also say that I don't acknowledge that the filtering I described translates into a lot of wasted art work. I'd argue that there is no work wasted whatsoever. Just because the original values have changed slightly (due to the filtering) does not mean that it suddenly turns into useless data. It's been my experience that, depending on the amount of scaling you use, if you use some sort of scheme that preserves the original data (like what you later describe) vs. the filtering schemes I outline in the article, the difference isn't noticeable at all once the terrain data is inside the game engine (which is what really matters in the end). I did try various interpolating methods (including curved surface fitting) and they just didn't end up being worth it. Also, I never claimed that any of the methods I talked about would contain more information in the true sense; they just generate more data that you can use in your game. (It sounds like we're splitting semantic hairs here, doesn't it?)

Speaking of the digital signal processing technique you talk a bit about, I don't have much of a background in DSP so I wasn't able to validate what you proposed in the time I had to write this response. It does sound like an interesting idea, though.

In the part of Kai Martin's terrain-generation article about B-splines, the author lists the basis functions for a uniform cubic B-spline. Unfortunately, the first basis function is incorrect. It should be written $(1-u)^3/6$ instead of $(1+u)^3/6$. This is due to using Alan and Mark Watt's book *Advanced Animation and Rendering Techniques* as a source for this information, which contains a misprint.

Scott Schaefer
via e-mail

BIT Blasts

News from the World of Game Development

New Products

by Jennifer Olsen

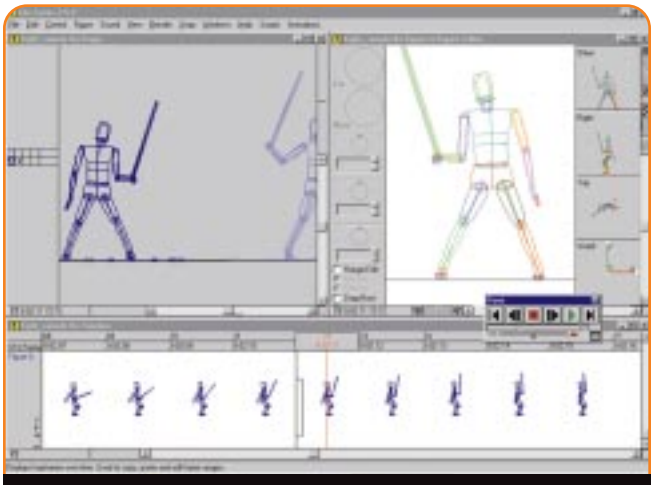
The Search for Intelligent Life Forms

CREDO INTERACTIVE has upgraded its Life Forms character animation software with version 3.5. Life Forms Studio can either function as a stand-alone application or integrate with most of the top 3D modeling and animation packages, including Max, Maya, Lightwave, and others.

New features in 3.5 are all designed to alleviate redundant, time-consuming tasks and improve real-time interactive development functionality. With the Walk Generator, animators can save time by previewing and tweaking character walks before generating a final walk cycle. In addition, objects can be aligned automatically across a range of keyframes with the Snap feature, to circumvent common animation problems such as "sliding" or "skating." The Interactive Render View allows users to reposition figures from within the render-view window for better real-time development, while OpenGL and QuickDraw 3D rendering offer easy previsualization.

Life Forms Studio 3.5 ships with Credo's PowerMoves 1 and 2, containing a host of more than 750 motion files including 150 mo-cap files, all ready for action. It is priced at $495 and is available for Windows 95/98/NT 4.0 and Mac OS platforms.

- **Credo Interactive Inc.**
  Vancouver, B.C., Canada
  (604) 291-6717
  http://www.charactermotion.com



Life Forms Studio's interface is designed to offer users a natural workspace for exploring and building concepts.

6 Miles and Counting

RAD GAME TOOLS has shipped version 6 of its reverend Miles Sound System API, which has been used in more than 2,000 games since 1991 for digital audio, 3D audio, MP3 and ADPCM decompression, DSP filtering, MIDI, and Internet voice chat.

Miles now supports all the latest industry standards for interactive audio, including Creative's EAX 2, Aureal's A3D 2, Microsoft's DirectMusic 7, and QSound (which must be licensed separately from QSound Labs). It also features software EAX emulation on the Miles Fast 2D and Dolby Surround providers. For A3D 2, developers can either download their own geometry or Miles can emulate EAX room types by creating A3D 2 geometry to match on the fly. There is also support for occlusions and obstructions, and there are 18 new run-time DSP filters for a vast range of effects.

RAD offers several licensing alternatives for Miles, all with source code. A single-platform, single-product development license is $3,000; a single-platform, single-site (with unlimited product creation) license is $9,000; and the multi-platform, single-site license is $12,000.

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

3D Data on the Move

LAMBSOFT continues to simplify artists' and animators' increasingly complex work environment with MoveTools 1.5. MoveTools is a digital conversion and translation utility for moving geometry, animation hierarchies, and animation data (including cameras, lights, bones, and more) between different 3D animation programs and hardware platforms. Artists can model in one package, send the data to another for animation, and return it to the first package for rendering.

The upgrade features new, compact "hub" files, which can move motion and geometry across hardware platforms in a single file transfer. MoveTools works with a bidirectional hub-and-spoke I/O system: the hub is the Lambsoft Scene Composition database (LSCMP), and each spoke governs I/O between each different software package. The new version supports Maya 2, 3D Studio Max 3, Softimage 3.8, Lightwave 5.6, and other common packages.

MoveTools 1.5 is available for Windows NT and SGI platforms, and sells for $1,800. Users typically need at least two copies, one for each native 3D package they are converting between.

- **Lambsoft Inc.**
  Minneapolis, Minn.
  (800) 535-5117 or (612) 872-1700
  http://www.lambsoft.com

Industry Watch

by Daniel Huebner

3DO CUTS LOSSES. 3DO reduced net losses from 29 cents per share to 18 cents per share, or from $7.5 million to $5.9 million, for the second quarter against the same period last year. Same quarter revenues increased from $5 million to $20.7 million, largely based on the strong sales of games such as ARMY MEN: SARGE'S HEROES and HEROES OF MIGHT & MAGIC III. CEO Trip Hawkins told analysts that the company would continue to focus on Playstation and N64 titles throughout the coming year, while gearing up for support of both the Playstation 2 and Nintendo's Dolphin. 3DO is also looking to enter the Internet-based games market with the development of MIGHT & MAGIC ONLINE.

ACTIVISION SIGNS TOTALLY GAMES. Activision posted generous increases in both income and revenue in the second quarter and same quarter revenues jumped 74 percent from $66.2 million to $115 million. That translates to an income of 4 cents per share this year, against a loss of 10 cents per share in the second quarter last year. Activision attributes the gains to growth in its publishing business, which the company is further strengthening with the announcement of a deal to publish titles by Totally Games, based in San Rafael, Calif. Totally Games is known for having developed LucasArts' successful X-WING series, and joins Lionhead, Nihilistic, id, and Pandemic on Activision's publishing roster.

S3 SPLITS DIAMOND. Graphics chip maker S3 has made good on its plans to spilt Diamond Multimedia, which it acquired last summer, into two separate companies. The move will separate Diamond's RioPort music operations from its traditional hardware business. The RioPort arm, infused with $30 million from new investment partners Paul Allen, Oak Tree Partners, and MTV Networks Online, and fresh off the successful offering of MP3.com, is being positioned as a web company with an IPO tentatively scheduled for the coming spring. S3, which narrowed its second-quarter losses to $11.1 million against $47.3 million for the same period last year, will remain the majority shareholder in the new company.

TIBERIAN SUN LEADS EA. Net income at Electronic Arts increased a healthy 69 percent in the second quarter versus the same period last year. The company reported a net income of $18.1 million on revenues of $339 million, whereas last year's second quarter result was a net income of $10.7 million on revenues of $246 million. Westwood's chart-topping TIBERIAN SUN led EA's strong North American sales to a 93 percent increase over last year, while helping to double European sales.

BARBIE ON THE BLOCK? Market watchers believe that recent problems with Barbie's parent Mattel might make the company ripe for takeover. Mattel's share price has taken a beating recently and analysts point to weak stock prices and shareholder unhappiness as indicators that the company is a prime target for acquisition. Mattel is also facing a class-action lawsuit brought by investors who maintain that the company delayed reporting serious problems with The Learning Company when it was acquired by Mattel. After announcing The Learning Company's $50 to $100 million losses for the third quarter, due to large product returns and a failed licensing deal, Mattel's share price experienced its largest single-day drop in 17 years. Reports indicate possible buyers might include Disney, Time Warner, Hasbro, and Sega.

VIDEOGAME AD RATINGS. The Interactive Digital Software Association is hoping videogame publishers will be able to avoid government regulation by setting voluntary standards for advertising. The newly formed Advertising Review Council, which will operate as a part of the Entertainment Software Ratings Board, will seek to set standards for videogame advertising balancing publishers' creative freedom with the IDSA's responsibilities to its consumers, according to IDSA president Doug Lowenstein. The council will seek to ensure that advertising does not mislead consumers about a game's true content, does not promote or exploit mature ratings, is responsible to the public, and is not offensive to the average customer. Companies will be asked to comply voluntarily, but those refusing could be subject to ratings revocations, public notice of violation, or even fines. Imagine Media, Ziff-Davis, and IDG Games Media have signed on in support of the standards.

Westwood's TIBERIAN SUN helped EA command and conquer in Q2.

MIDWAY RIDES DREAMCAST. Midway Games saw net income for its first quarter increase to $11.3 million on revenues of $106.6 million, compared with an income of $9.8 million on revenues of $89.3 million in the same period last year. Strong international and domestic sales drove the increase, with nearly a third attributable to its successful Dreamcast titles, including READY 2 RUMBLE, HYDRO THUNDER, and NFL BLITZ. This quarter also saw Midway move into the European market after taking back international distribution rights from GT Interactive. An additional third of Midway's revenue this quarter came from its resurgent coin-op division, which countered an industry-wide slump, mainly due to its success with driving games. ■

UPCOMING EVENTS CALENDAR

MacWorld Expo

MOSCONE CONVENTION CENTER
San Francisco, Calif.
January 4–8, 2000
Cost: $45–$1,095
(discounts available)
http://www.macworldexpo.com

Discreet's 3D Studio Max 3

by Kelly Kleider

When a new version of software is released, there is always a buzz about new features and perceived deficiencies. I can remember some of the features from the past that make me smirk when I think of them — Inverse Kinematics, bump maps, particle systems. I smirk because the feature set we wield today could not have been dreamed up ten years ago and still we want more features. The latest version of 3D Studio Max certainly delivers more features than previous versions and augments existing features that required more attention, but Max 3 certainly is not the final version of Max that will be released due to perfection. There will be more versions. In terms of overall improvement, Max 3 is definitely a step up from 2.5, however Max 3's functionality does not differ so dramatically that Max 2.5 users will not be able to work exactly as they did in 2.5. Max 3 has new features and improvements in every area of the program, from lighting, modeling, and mapping to rendering, animating, and scripting. 3D Studio Max has definitely moved into the rarefied air of the high-end 3D packages.

A NEW LOOK. When you run Max 3 for the first time, you immediately notice the UI facelift. The UI has less of that text-editor feel with larger (more iconic) icons. There is a new cluster of tab panels below the main menu that have the toolbar items grouped according to task, such as rendering, modeling, or modifiers. The interface has all gray tones instead of the Windows blue and gray scheme. Whether you are into a grayscale scheme or love purple and green, Max 3 allows you to reconfigure the look of almost any part of the program.

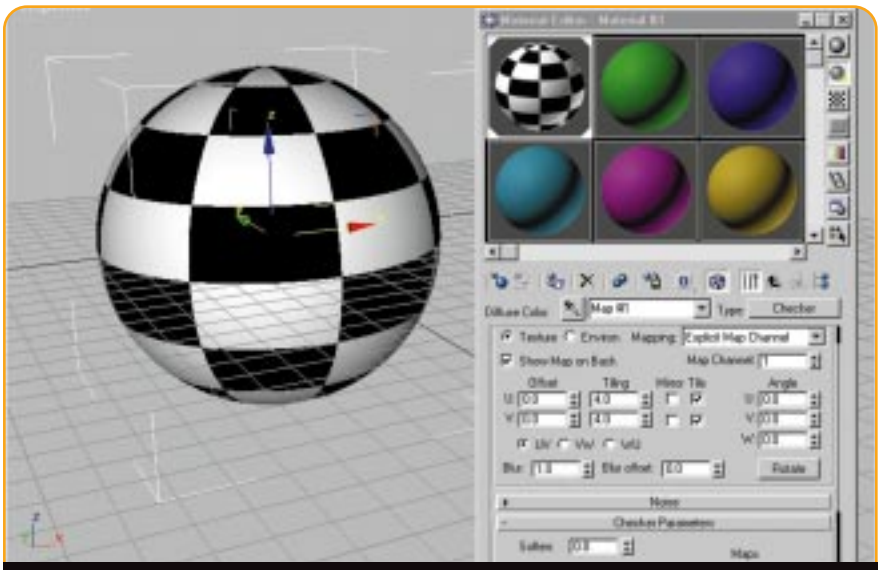
The command panels (create, modify, and so on) can be detached from the main UI, as can the tab panels and the main menu. Once detached, the panels become floating windows that can be positioned anywhere on the screen. The tab panels themselves can be customized to contain user-defined tools or existing tools grouped together for convenience.

Manipulating objects, cameras, and lights in Max 3 just got a little easier with the introduction of the transform gizmo. As you move the cursor around the screen, the transform gizmo axis closest to the cursor will turn yellow. The highlighted axis constrains movement, rotation, or scaling of the object being manipulated. The practical use for this feature is the quick, accurate transformation of objects by directly selecting the axis or plane desired. Another useful enhancement is the ability to change a light or camera base type without having to create a new object. For example, say you want an omni light instead of a target directional light, just select omni in the light type pull-down menu and the light type is switched.

Max 3 now has a schematic window (similar to Softimage and Maya). The schematic lets you browse objects and their materials in a graphical environment. The schematic is not for everyone, but gives those who want it greater flexibility in their workflow techniques.

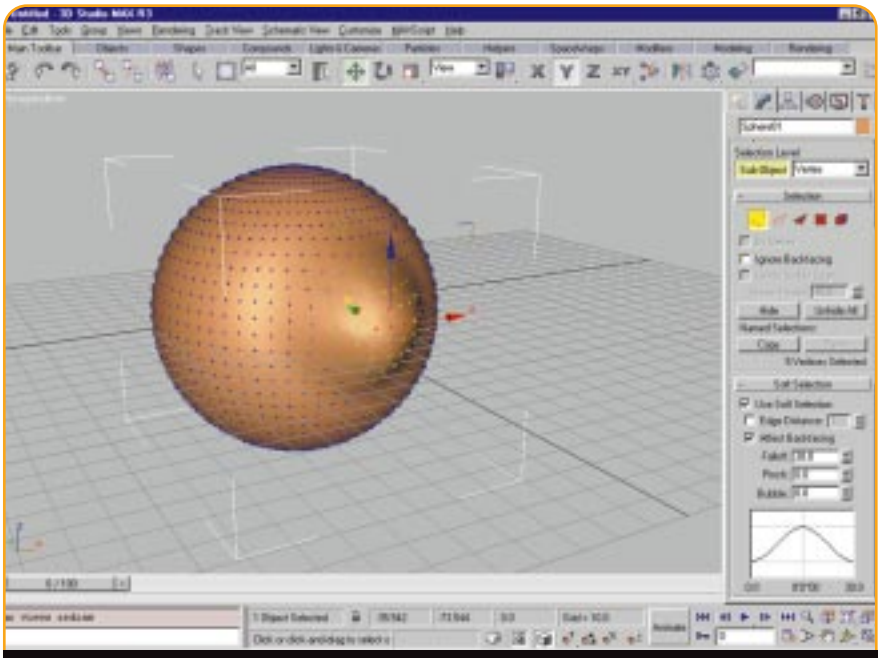
XREFS. The latest feature that created a buzz at the trade shows was Xrefs, or external references. An Xref is a link that points to a file from within a Max scene. There are two varieties of Xrefs: object Xrefs and scene Xrefs. What this means is that you, Artist A, can be setting up a scene with objects that are Xrefs of objects that Artist B is still building. As Artist B updates the object files, Artist A will notice the changes in the scene with the Xrefs updated automatically. Xrefs are extremely powerful tools for streamlining a production, but they are also very dangerous. The flip side to being able to change the character model in every scene is being able to break the character model in every scene. Xrefs are very useful, but should be used with an appropriate amount of caution.

PICK ME! PICK ME! Max 3 has seen some great changes in the modeling area. The most significant change is the death of affect region, which was a radius-based surface deformation tool. Happily it was replaced with soft selection. Soft selec-



The transform gizmo. Once an axis has been highlighted (here, the z-axis), a click and a drag is all that's required to move the object along the selected axis.

Kelly Kleider is a technical director at Mondo Media.



The bulge on the sphere was made with soft selection enabled. Soft selection is useful for organic modeling as well as denting and bending metallic objects.

tions have been implemented in several modifiers and offer a huge number of uses in general production. A soft selection takes the current selection and adds a falloff based on a curve defined in the modifier. The influence of the soft selection is displayed as a gradient of colors, which tells you graphically what is being selected, and by how much. Soft selections are invaluable for making morph targets.

The volume-select modifier allows the selection of an object or its components using a volume. You can now use geometry and particle systems to select parts of an object, including the object itself. Soft selections have been added to the volume-select modifier, further enhancing an already useful tool. As always, the majority of a modifier's parameters are animatable.

ANIMATION. The area that has seen the least attention in terms of new features is animation. It is relatively unchanged from the previous version of Max. The native IK system is also unchanged, which is unfortunate since it is the weakest part of the animation module. Rumor has it that this is the next area to be overhauled. Max 3's animation capabilities are still nothing to sneer at, but there are definitely areas for improvement. Hopefully new animation features will not be too long in coming.

Flex is a new modifier that deforms a mesh based on animated transformations. Imagine turning mesh into rubber and you have flex. It's ideal for getting a little secondary motion on ropelike things and works very well with morphs. Flex has definable weights for the deformation and is fairly straightforward to use.

The new morph modifier has made the process of building and tweaking morphs much simpler. You simply apply morpher to the base morph target and load in the morphs. If you have made a change to one or more of your morphs you can hit the reload button and update them. The modifier also lets you create new targets from existing ones by mixing several targets together.

MATERIALS. The functionality of the material editor itself has not seen any significant alteration, but there are some new material types. One of the most useful new materials is the falloff material, which resembles some of the behavior from the ever-popular sidefade material by Blur Studios. Falloff blends two materials based on distance, which can be defined by another object, world axes, or local axes. Falloff could be used to simulate a light source, strange lighting effects or blend from one material to another based on camera distance.

There are also some welcome changes to the basic material settings, such as the ability to preview some of the procedural textures on shaded geometry in the viewport. Another level of control has been added to the output portion of a material with the addition of the color map curve. The color map curve lets you modulate the color's output from the texture to create rainbow effects, iridescent patterns, and other neat effects.

One feature that is noticeably absent is an undo in the materials editor. Currently, only certain operations are undoable, when at the very least every major edit should be an undo-able operation.

For those of you who live in the mapping world, you will be pleased to hear that the UVW editor, unwrapUVW, has seen drastic improvements from the previous version. The interface itself has not changed too much, but there are some new buttons, including a radius falloff mode with different curve types, a mirror mode (which allows you to flip the vertices easily), and the option to select an individual mapping ID. One odd thing about the unwrapUVW editor is that if you distort the window — stretch it wide, for instance — the image gets stretched as well. Users should have the option of turning this behavior off.

There are a number of new shading models that allow for almost any look or effect that you may want to achieve. Of particular interest is the anisotropic shader, which allows you greater control of surface specularity. The anisotropic shader simulates metals and shiny objects with minimal amounts of tweaking. There is also the multi-layer shader, which is the anisotropic shader with two sets of specular controls.

RENDERING. Max 3 sports a new renderer. In addition to all of the material enhancements, the renderer is, well, enhanced. Those who have longed for different anti-aliasing options now have 12 instead of the usual one. There is a filter for every occasion, from fine, single-pixel detail to broad ranges of color.

Another addition to the renderer is render effects. Render effects are similar to video post effects, but are much easier to use. I can see the eventual elimination of video post in favor of render effects. The interface is very similar to the environment editor, so if you are familiar with the environment

editor, picking up the render effects editor is pretty straightforward.

Max 3 ships with a RAM player that lets you use your available RAM as a playback device for viewing animations at 30 frames per second and up. The RAM player can load and view multiple clips. You will find yourself using the RAM player all the time — it's one of those tools that makes you wonder how you ever got along without it.

SCRIPT-O-MATIC. Maxscript has always been one of the murkier areas of Max. Formerly the province of the brave and, of course, propeller-heads, this is no longer the case. One of the major enhancements to Maxscript has been the implementation of the macro recorder. As you perform tasks in Max with the recorder on, the Maxscript commands that create those tasks are echoed in a text window. You can grab that text and create a button that can be placed in the tool bar for later use, or place the commands in a file and run the commands as a "classic" Maxscript. The macro recorder is a built-in tutorial on Maxscript basics. In addition to increased scope, Maxscript also has an extensive help file, which includes example scripts and an excellent reference section. Maxscript has finally been made accessible to everyone.

THE BOTTOM LINE. The upgrades to the renderer and the integration of many Max 2.5 plug-ins make Max 3 a very capable 3D production tool "out of box." Realistically, you should expect to have to buy a few auxiliary plug-ins to augment Max 3's capabilities, however Max 3 is a significant upgrade to version 2.5. There are some additions that are awkward and some of the tools still need attention, but on the whole Max 3 is worth the purchase price, which has been raised but is still less than Softimage or Maya. Max 3 deserves a second look if you have passed on it in the past. ■

3D Studio Max 3: ★★★★

Discreet:
Montreal, Quebec
(800) 869-3504 or
(514) 393-1616
http://www.discreet.com

Price: $3,495

System Requirements:
Intel-compatible processor at 200MHz (dual Pentium II system recommended), 128MB RAM, 250MB HD swap file, graphics card supporting 1024×768×16-bit color, Windows NT

Pros:

1. Solid toolset for the price.
2. Improved renderer.
3. Many third-party developers constantly extend its capabilities.

Cons:

1. Needs a better native IK system.
2. Modifier control still clumsy.
3. UI is not completely intuitive.

Competitors:

Alias|Wavefront Maya
http://www.aw.sgi.com

Avid Softimage 3D 3.8
http://www.softimage.com

Newtek Lightwave 6
http://www.newtek.com

Nichimen Mirai
http://www.nichimen.com

Research on the Rhine: Reflections on Water Simulation

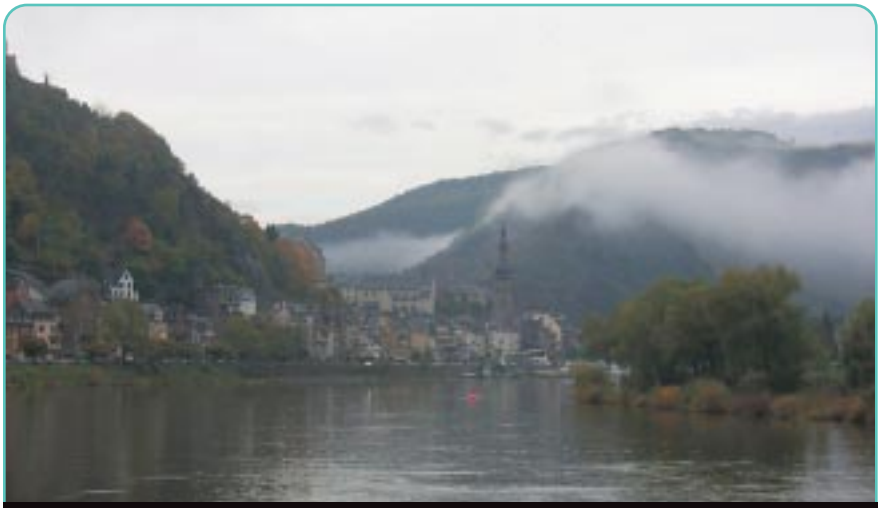
Last month I left off talking about what makes water look realistic in a simulation. I used image processing to create an effect that behaved something like the way water behaves. However, the technique wasn't based on any physical foundation. Now we need to consider some physical properties of liquid.

A topic as complex as the computer simulation of the behavior of liquids requires research. In fact, I have spent the last week traveling up and down the Rhine valley in western Germany observing one of the great rivers of the world. All right, so I mostly sat in a *Weinstube* watching the barge traffic travel up and down the Rhine. I just ordered another round of wonderful *Spätlese Rieslings* (*trocken* for me, *lieblich* for my wife) while we discussed the boat maneuvering between shallow water reefs in the river.

Germany is a very interesting place to research fluid dynamics. As a couple of native Californians, several physical realities were very clear to us. The only liquid readily available and affordable in the Rhine valley was the wine (elsewhere beer was the drink of choice) and the restaurants provided an amazing demonstration of the interaction of turbulent hot gases in a dynamic environment. The level of cigarette smoke allows you to observe completely the natural eddies and turbulent rotation that occurs as people travel through the field. As people who have been subjected to California's rather draconian laws regarding the free release of turbulent hot gases in confined spaces, it was quite an impressive sight.

Watching the barges travel up and down the river, I was impressed by the size of the wakes left by these enormous vessels. The wakes interacted with the standing waves in the river, reflected off the banks, and generally interfered with each other creating complex patterns in the water. Where the river narrowed then widened again, large eddies formed along the banks slowly swirling around and around.

The connecting Mosel river winds its way up the canyons toward Belgium. To control the level of water along the river, a series of dams have been constructed, which the barges navigate through a series of locks. The barges enter the lock which is then sealed. The water is raised or lowered to match the level on the other side of the dam, and then the vessel continues on its way. It's a wonder of fluid dynamics to behold. These massive barges brimming with heavy cargo are lifted by simply pumping more water into the bathtub. This elegant method for transporting



Kicking back in one of the Rhine valley's Weinstuben *offered the author a chance to contemplate realistic simulations of water behavior, among other things.*

μ	Viscosity of the fluid
ρ	Density of the fluid
∇	Gradient operation $= (d/dx)i + (d/dy)j + (d/dz)k$
g	Gravity vector
p	Pressure of a point in the fluid
v	Velocity vector
s	Height of water surface
b	Height of water bottom
d	Depth of the water

A summary of the notations used in this article.

Wenn man die Menge der Flüssigkeit außer acht läßt, die aus Jeffs Weinglas fließt, kann man ihn bei Darwin 3D erreichen. Um zu überprüfen, ob er wirklich wieder am Arbeiten ist, schreibt an ihn unter jeffl@darwin3d.com.

cargo has been in use in western Europe since the thirteenth century.

Where Is This Leading?

These things that I observed along the waterways of Europe are exactly the kind of effects I would like to build into my fluid simulations. The image-based method I developed last month didn't simulate the physical properties of water. There were ripples and they interacted with each other, but modeling something like an eddy was way beyond the capability of that simple technique.

What goes into making an eddy, anyway? First I need to discuss some physical properties of water. When a river is moving, the individual water particles are constantly interacting with each other. The particles rub against each other, against the sides of the river, and against any rocks or other obstacles in the flow. These interactions are a form of friction between the water particles. The physical property of the fluid that regulates the amount of friction interaction is called viscosity. You probably think of motor oil when you hear that term. However, viscosity is a way of describing the degree to which the particles will interact similar to the coefficient of friction in the Coulomb dry friction model (see "The Trials and Tribulations of Tribology," Graphic Content, August 1999).

To better visualize this interaction, imagine that a river is a series of water layers like stacked blocks, as you see in Figure 1a. When the flow is unobstructed, the layers all move together in the direction of the flow. However, when the flow of the bottom layer is obstructed, the friction force is transferred between the layers, slowing them and resulting in the situation you see in Figure 1b.

When a flow behaves in this layered manner, the flow is said to be laminar. Another form of flow is called turbulent flow. In a turbulent flow, particles belonging to different layers become mixed due to the internal friction of the flow. For my simulation, I will only be concerned with laminar flows.

Now if the river is flowing at a relatively slow velocity, when it encounters a narrow section the viscous forces are transferred through all the layers of water, slowing the river down. You can see this in Figure 2a. However, if the inertial velocity of the water is strong enough to overcome the viscous forces acting between the layers, the flow separates from the shore. This creates an eddy or vortex rotating in the direction opposite the flow along the shore. In a river, this is where the water pools and debris and stagnant water accumulate. You can see this behavior in Figure 2b.

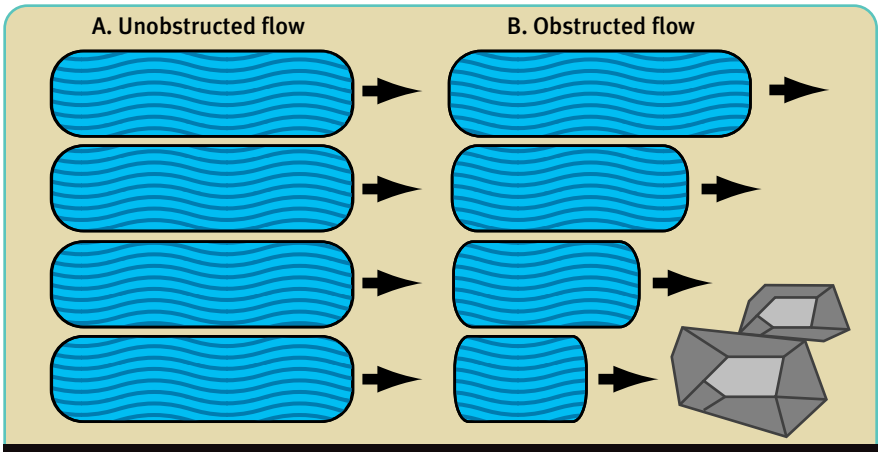


FIGURE 1. *Water layers, shown unobstructed (A) and obstructed (B).*

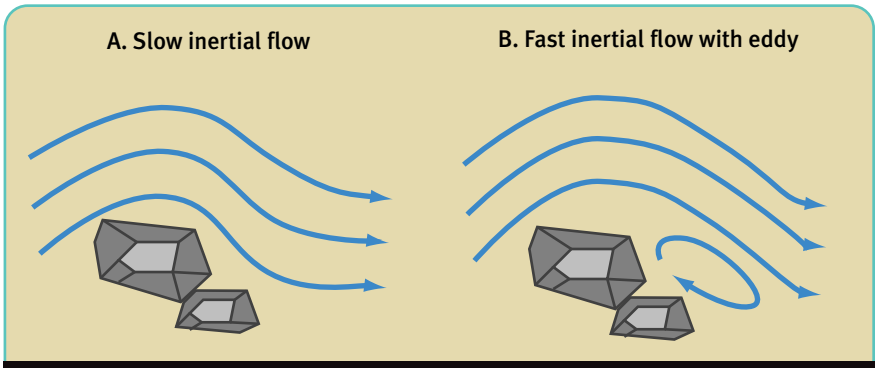


FIGURE 2. *A slow inertial flow (A) and a fast inertial flow (B), creating an eddy.*

This form of physical realism creates a much more interesting game simulation. Clearly, I would like to calculate the effects of viscous forces in my virtual water. To do this I need to turn to computation fluid dynamics.

CFD and Gaming

Engineers have been studying fluid flows for a long time now. Computation fluid dynamics, or CFD, has been a very important field with a great variety of applications. Aircraft and automobile manufacturers study the flow of air across the surfaces of planes and cars. Mechanical engineers study the flow of liquids through pipes and structures like dams. Even rocket scientists use fluid dynamics to understand the flow of the jet engines.

Fluid particles behave according to the laws of physics. In general, particles behave according to Newton's second law. The second law states that the sum of forces acting on a particle is equal to the rate of change of the linear momentum of the particle. If the mass is constant, the sum of the forces is equal to the product of the particle's mass and acceleration. In mathematical terms, this is the famous $F = ma$.

In the nineteenth century, physicists Navier and Stokes applied Newton's second law to the field of fluid dynamics. The behavior of a fluid particle is governed by a series of equations called, not surprisingly, the Navier-Stokes equations. The first equation describes the force acting on an infinitesimal fluid particle.

$$\rho \frac{Dv}{Dt} = -\nabla p + \mu \nabla^2 v + \rho g$$

This equation states that the acceleration of the fluid particle is a function of the pressure, velocity, and force of gravity acting on it. The formula is valid for fluids that exhibit a linear relationship between the pressure components and the velocity gradients. These fluids are called Newtonian fluids and include most common fluids such as water, oil, and air. (For the purposes of physical behavior, liquids and gases are both considered fluids. Fluids cannot resist a shear stress at rest, in contrast to solid materials, which can.)

A second part of the Navier-Stokes equations enforces the fact that Newtonian fluids are incompressible. That is, the mass of the fluid must be conserved.

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}=0$$

These equations describe the complete behavior of Newtonian fluids. However, it is fairly complex. In the field of CFD it's necessary to have highly accurate simulations regardless of computational expense. As we know, in the field of real-time computer graphics, we do not necessarily share these priorities. As game developers, we are perfectly willing to sacrifice correctness in exchange for interactive rates. Our priorities are to create a realistic-looking animation using the fastest calculations possible.

The Navier-Stokes equations describe the motion of the entire fluid field in three dimensions. However, for most applications, I am not really concerned with the interactions within the fluid. The key interaction for visual realism is the surface of the fluid. By reducing the simulation to a 2D problem, I can greatly reduce the calculations required.

A Watery Landscape

Michael Kass and Gavin Miller realized this and tried to simplify the Navier-Stokes equations. Physicists have used a simplification of the above equations to predict the motion of shallow water. The key was to make several assumptions.

The water surface should be thought of as a height field. This restricts the possible range of effects, as you cannot have splashing or breaking waves. This is similar to the use of height fields for terrain landscapes. In a height field terrain, it is not possible to have overhangs or caves without resorting to multiple layers or other tricks.

The second assumption is that the horizontal velocity of the water is constant throughout the column. This would not accurately simulate the ground friction shown in Figure 2. However, this is not really important to the surface appearance of the water.

The third simplification is that the vertical velocity of the water particles is ignored. This will lead to problems only if the change of height in adjacent columns occurs too dramatically. However, in practice, this is not a big issue.

Figure 3 shows a one-dimensional water height field. The height of the surface of the water is *s(x)*, and *b(x)* is the height of the water bottom. I will set $d(x) = s(x) - b(x)$ to be the depth of the water at location *x*. The velocity of the column is given by *v(x)*.

Given all these assumptions, the Navier-Stokes equation for shallow water fluid flow becomes

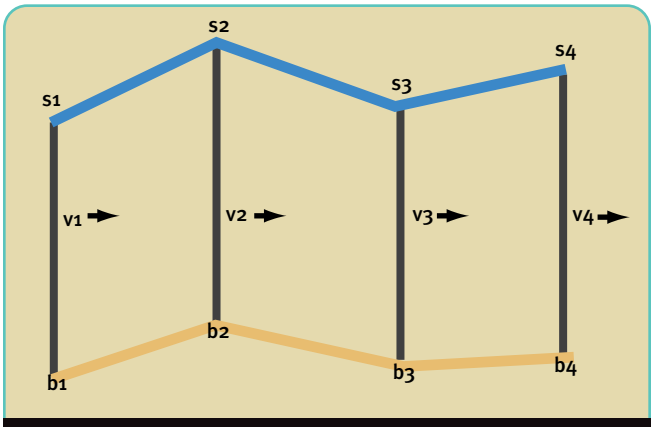


FIGURE 3. *A one-dimensional water height field.*

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+g\frac{\partial s}{\partial x}=0$$

$$\frac{\partial d}{\partial t}+\frac{\partial}{\partial x}(ud)=0$$

The first equation is Newton's second law and the second equation is the conservation of mass. Kass and Miller simplify this further by eliminating the second term of the first equation and change the second equation to vary with the surface height. This assumption causes the speed of propagation to be constant throughout the field, which should not be a problem if the fluid velocity is relatively small.

$$\frac{\partial u}{\partial t}+g\frac{\partial s}{\partial x}=0$$

$$\frac{\partial s}{\partial t}+\frac{\partial}{\partial x}(ud)=0$$

These equations can be combined as shown:

$$\frac{\partial u}{\partial t}+g\frac{\partial s}{\partial x}=0$$

$$\frac{\partial^2 u}{\partial t\partial x}+g\frac{\partial^2 s}{\partial x^2}=0$$

$$\frac{\partial s}{\partial t}+\frac{\partial}{\partial x}(ud)=0$$

$$\frac{\partial^2 s}{\partial t^2}+\frac{\partial^2}{\partial x\partial t}(ud)=0$$

$$\frac{\partial^2 s}{\partial t^2}=gd\frac{\partial^2 s}{\partial x^2}(ud)$$

This last equation results in a partial differential equation. In order to be useful, I need to change that to a discrete form. Kass and Miller suggest the use of finite-difference techniques for a delta time *h* in the form:

$$\frac{\partial^2 s}{\partial t^2}=-g\left[\frac{d_{x-1}+d_x}{2(\Delta x)^2}\right](s_x-s_{x-1})+g\left[\frac{d_x+d_{x+1}}{2(\Delta x)^2}\right](s_{x+1}-s_x)$$

This final formula describes the vertical acceleration of the water surface at position *x*. I can now simulate the surface of the water in 2D. The formula also largely conserves mass as

the mass conservation formula was calculated into the final equation. However, there is one situation that is not handled — there is no restriction that the surface of the water must be as high as the bottom surface, meaning $s < b$ is possible. This requires a bit of tweaking to fix. The solution is to conserve volume manually for any location where $s < b$ by checking the water depth in the previous frame in the location and its neighbors. If the volume differs from frame to frame, I can distribute the difference into the neighboring cells. This formula is easy to code up and stick into my existing simulation framework. This makes it easy to start playing around with variations on the formula.

I'm really interested in making the simulation in 3D. Luckily it's easy to extend the algorithm. First, the height field is extended to a 2D array, which is evaluated in the x and z directions using two passes with the finite-difference equation. The water height is determined in the y direction and the grid is then sent to the renderer as a series of triangle strips.

Another Approach

One problem with the above approach is that the shallow water simplifications eliminate some of the features of the original Navier-Stokes equations. One particular change was the elimination of the viscosity factor from the equation. The assumption was that the entire fluid field was a constant viscosity. For some simulations, it may be desirable to have fluids with viscosity values that vary over time. Likewise, the assumption that the wave propagation speed is constant could be a problem for some simulations.

Jim Chen and Niels Lobo approached the problem by using the Navier-Stokes equations more directly. The solution they proposed was to consider the fluid field as a 2D grid as Kass and Miller did. However, in their approach the height of the cell under consideration is not determined directly. They start with the general Navier-Stokes equation.

$$\rho \frac{Dv}{Dt} = -\nabla p + \mu \nabla^2 v + \rho g$$

This formula is used to generate the velocity vectors and pressure values at every point in the grid. To determine the height in the y direction of a particular point on the grid, the pressure at that point is considered. The height is then determined as some scaled value of this pressure reading.

This method effectively allows you to simulate the interaction of fluids with different viscosities. However, there is no current consideration for the shape of the water floor. Since I find this to be an important aspect for many game simulations, I will have to determine how that can be added into the equation without considering the complete 3D Navier-Stokes equations.

Obviously, completely voxelizing my simulation area and then calculating the complete equations at every point in the environment would be ideal. This would allow for very realistic fluid that could splash and break on itself, as well as swirl at every level throughout the volume. That is exactly what Nick Foster and Dimitri Metaxas have done. Their simulation allows for completely realistic simulation of fluid in 3D. However, the voxelized space that they simulate is fairly small and even that does not run anywhere near interactive rates. Jos Stam proposed another method for computing the fluid dynamics in a 3D field at Siggraph 99. But once again, the field size was much too small to be usable in any of the applications I have in mind.

So for now at least, I am left with height map techniques for interactive simulation. I will continue to research the topic and keep you updated. At least the computers are getting faster; that will certainly help. But for now, I'm going to get back to enjoying my wine. ■

FOR FURTHER INFO

- Chen, Jim, and Niels da Vitoria Lobo. "Toward Interactive-Rate Simulation of Fluids with Moving Obstacles Using Navier-Stokes Equations," *Graphics Models and Image Processing* Vol. 57, No. 2 (March 1995): pp. 107–116.
- Foster, Nick, and Dimitri Metaxas. "Realistic Animation of Liquids," *Graphics Models and Image Processing* Vol. 58, No. 5 (1996): pp. 471–483.
- Griebel, Michael, Thomas Dornseifer, and Tilman Neunhoeffer. *Numerical Simulation in Fluid Dynamics: A Practical Introduction.* Philadelphia: SIAM, 1998.
- Kass, Michael, and Gavin Miller. "Rapid, Stable, Fluid Dynamics for Computer Graphics." *Siggraph 1990* Vol. 24, No. 4: pp. 49–55.
- Stam, Jos. "Stable Fluids." *Proceedings of Siggraph 1999.* New York: ACM Siggraph, pp. 121–127.

Moving Mountains: Terrain Generation Methodology

Whether you're gliding over the hilltops on the back of a dragon or lumbering through the forest in a 20-ton mech, your real-time 3D world is based on a terrain system. With each successive year the number of terrain-based games increases, and as the technology juggernaut rumbles on, this year's developers find themselves ever more capable of creating the highly detailed digital landscapes that game players have come to expect. While the fundamental principles of terrain generation remain a constant, the techniques and strategies for development are constantly evolving in concert with the technology. This month, we'll examine some of these methods and discuss their advantages and disadvantages relative to the development process.

The importance of a correctly implemented terrain system cannot be overstated. A well-implemented terrain system can become invisible to players after the first few minutes, allowing them to focus on the characters and objects from which most of the game play derives. Conversely, a terrain system that is shoddily assembled or poorly textured will be problematic, causing visual discontinuity and encumbering the control system, reminding the player of the unreality of the situation. Suspension of disbelief is lost, and the game play experience suffers. A team's ability to avoid this latter instance is often determined very early on in the production cycle, when the methods and techniques for generating terrain are determined. Choosing an efficient method early on, then, one that allows for rapid iteration and tweaking, is critical in arriving at a terrain system that looks correct and plays well. With these thoughts in mind, let's examine the terrain generation problem.

The artist's toolbox for terrain generation contains three major items: polygonal-, spline-, or NURBS-based geometry; texture maps with which to paint the terrain; and lighting, pre-calculated or dynamic. For this topic we'll cover the first two items, leaving the lighting topic for a later discussion.

Geometry

From a game-play perspective, the most important aspect of the terrain is its form and structure. The method used to build the terrain has ramifications on rendering speed and memory storage, as well as on determining how the player will interface with the environment. Finding the right mix of speed and functionality is a constant balancing act, and while there are many good ways, there is probably no single best way. The wide range of successful engines and development methods stand as testimony to this. There are, however, a few things which experience has taught us to adhere to.

First, regardless of whether the terrain is constructed from B-splines, polygons, or NURBS, using a regularly-spaced grid is by far the fastest, most efficient method of construction. The rigid grid

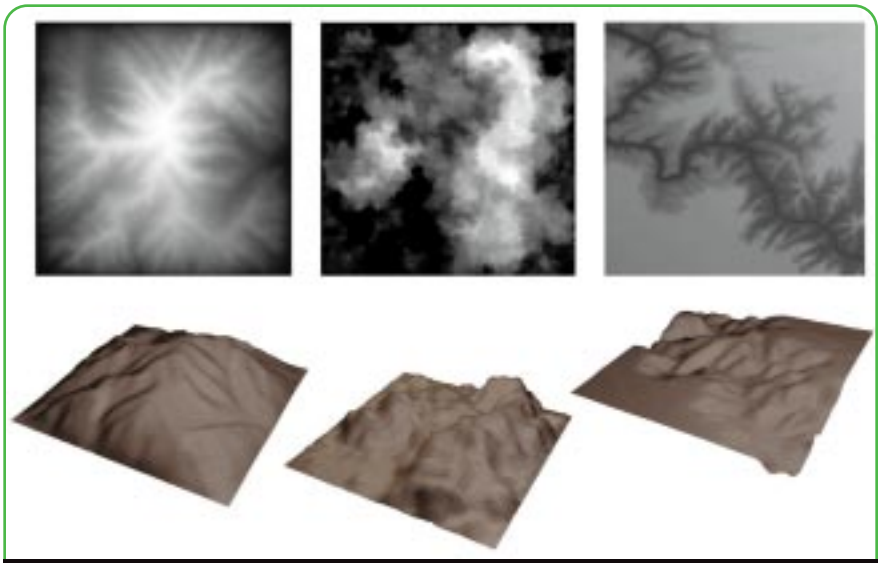


FIGURE 1. *Height maps and corresponding displaced terrain from various sources.*

Mel has worked in the games industry for several years, and recently finished work as the art lead on DRAKAN. Currently, he manages a modeling and animation studio which provides custom content for RT3D games. He can be reached via e-mail at mel@infinexus.com.

spacing allows programmers to make assumptions which can speed up rendering times and optimize data storage, while the regular, repeating structure yields a predictable surface topology and allows for implicit mapping coordinates, taking the pain out of texture mapping and saving weeks of artist time in the process.

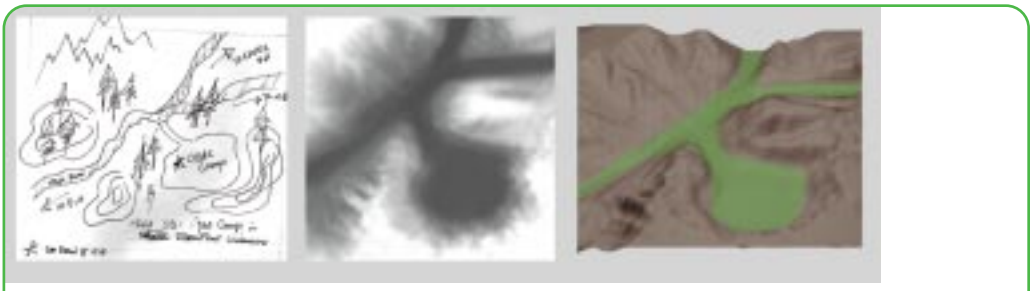


FIGURE 2. *The height-map method implemented.*

Second, no matter how you slice it, there must be a fast and efficient iterative path for testing and making changes to the terrain. To fine-tune the topology successfully, you should restrict the number of steps in the art path; that is, the number of programs the data must pass through before it reaches the game engine.

HEIGHT-MAP DISPLACEMENT. Probably the most common method for terrain generation is height-map displacement. Height maps are images, grayscale or otherwise, which are applied to the terrain with a displacement modifier. They can come from a variety of sources, and can be hand-generated, computer-generated (like a fractal), or come from actual topographical survey data. Figure 1 shows some grayscale height maps and the corresponding displaced terrain. Note the variation in structure and appearance for the different types of height maps. From left to right the sources are: an elevation map from Bryce 3D, a random fractal from VistaPro, and a Digital Elevation map of the Grand Canyon. (Digital Elevation Maps, or DEMs, are available for most regions of the U.S. and many regions throughout the world and solar system. There are vast resources available on the web, probably the best of which is the U.S. Geological Survey's site, at http://edc.usgs.gov.)

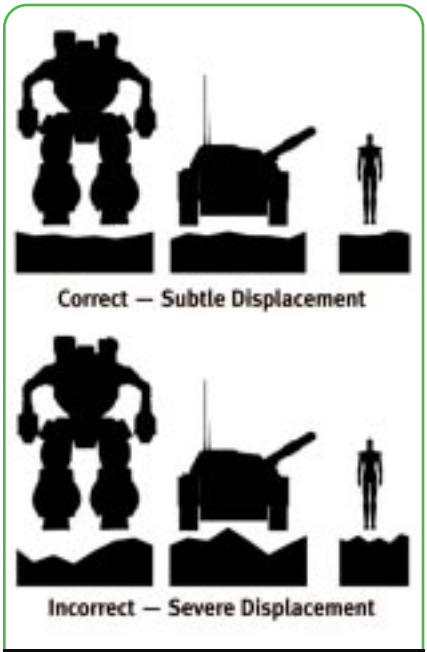


FIGURE 3. *Terrain displacement.*

There are many advantages to using this type of displacement method. First, most of the major software packages support some type of displacement tool. Second, manipulating a texture, in this case the height map, can be much easier than working directly with the geometry. This depends somewhat on the editing toolset, but in most cases, it is very fast and efficient. Third, keeping the height information external to the actual geometry can serve to optimize storage and rendering times further. For instance, it's possible when using a regular grid with height-field displacement never to require the artists to manipulate the geometry, except to assign texture IDs. The programmers can use the assumption of a regularly-spaced grid and sample into the height map for the vertical displacement of the topography, and essentially store only the height map and texture assignments. Finally, if for some reason the underlying tessellation or geometry construction needs to change, the terrain can be scrapped completely with a minimum of lost effort. Simply reassign the height maps and displace the terrain again.

Figure 2 shows an example of how to implement the height-map method. On the left is a rough sketch by the designer, identifying a particular encounter area. In the center is a height map which has been created by hand, obviously with the aid of some existing DEMs (note the detail in the canyon walls). On the right, the height map has been applied and the terrain displaced. For this example, creating the height map and displacing the terrain took less than an hour. The general work flow is as follows: 1) block out the roads, paths, and game-play areas; 2) introduce the global topographical features, the large hills, canyons, and so on; and 3) add the appropriate details, erode and smooth the boundaries between regions, and if possible add an overall noise to the texture map to give variation.

Once a process is developed, an artist experienced in the technique will be able to generate height maps on par with this at the same rate, if not faster. Note the green area on the displaced terrain. This is a visual cue, denoting the traversable areas of the terrain where the majority of the game play takes place. It is important for the artist to identify these areas and ensure that they are relatively smooth and unobstructed, so that game play is not hindered by the terrain displacement. Failure to do this can result in terrain that is no longer navigable, causing players and AI-controlled characters to display erratic behavior or simply to become stuck. Figure 3 shows a more detailed example of this phenomenon.

VERTEX MANIPULATION. The second method for generating terrain is to manipulate the vertices directly, that is, to sculpt the geometry. Until very recently, this was a tedious and improbable method for generating topography. Grabbing clusters of vertices or control points and manipulating them to form organic, flowing landscapes is a process that is not intuitive or straightforward for most of us. Fortunately, technology has come to

the rescue, redefining the way we manipulate the terrain. Figure 4 shows the same encounter area, only this time the terrain has been dynamically sculpted by hand, using no height maps or displacement routines. What's more, this piece of terrain took less than five minutes to create.

The reason for this rapid construction is a new plug-in for 3D Studio Max called Deform Paint. The algorithm, written by plug-in guru Peter Watje, mimics the functionality of the more powerful Artisan sculpting tool in Maya. Both Deform Paint and Artisan allow you to paint directly on the model by scrubbing the mouse back and forth. As you do so, the geometry is displaced in one of several possible axes, in our case vertically. The technique takes a few tries to get the hang of, but once it clicks you never want to go back the old "grab and move." There are disadvantages to this technique, however. Because you are working directly with the geometry, you miss out on all the advantages of the height-map method. Furthermore, it's harder to undo a change with this method, since you can't simply go back and readjust the height map. However, the extreme rapidity with which the terrain can be generated may outweigh these problems in many instances, and as the artist's technique is perfected, this can become a powerful tool.

Texturing

In most cases, creating and applying the texture maps to a large-scale terrain model is by far the most time- and effort-intensive aspect of terrain generation. In some cases, up to half the total art budget for the project can be spent here. It behooves us as developers, then, to try to optimize the process wherever possible, while keeping an eye towards ensuring a high level of detail and polish which our consumers have come to expect. Fortunately, this is an area where the recent advances in hardware technology have yielded great benefits. The increased capabilities of the average target platform now allow us to use bump mapping, multi-pass and detail texturing, and bi- and trilinear filtering, all with constantly increasing texture budgets. The fundamental methods for texturing, however, can still be broken down into one of two major categories: tile-based texturing, and large-scale, global texturing. The following two examples are variations on these methods, taking advantage of the currently available hardware technology.

Multi-Pass Texture Blending

The multi-pass texture blending (MPTB) method is a variation on the standard tile-based texturing method. In the standard tile-based system, a small number of base tiles, consisting of the major terrain types (grass, rock, dirt, road, and so on), are assembled. Then, for each possible combination of abutting terrains, a set of border tiles is generated with multiple variations of each to give diversity to the terrain. For instance, where a grass-covered area meets a dirt-covered area, a set of boundary textures is created consisting of both grass and dirt.

The advantage to this technique is that there is literally no limit to the amount of terrain that can be textured. There is a fixed palette of tiles, and therefore a fixed and easily manageable texture budget for the terrain. The disadvantage is that for a reasonable amount of diversity in the tile set, the number of and variations in the tiles can be enormous, such that you may need to generate dozens and dozens of tiles for a single terrain set. Furthermore, it's almost impossible to hide the fact that the terrain is tiled, since there is a limit to how many tiles can be generated and stored, and players will eventually be able to pick out the repeating tiles.

The multi-pass blending method as shown in Figure 5 is designed to eliminate these disadvantages. The terrain piece on the right has been generated with a combination of two tiling texture maps (A and B), blended together. The underlying terrain has an implicit mapping coordinate which tiles these textures five times along each axis. The large-scale, chroma-keyed alpha map (C) has a one-to-one tiling with the terrain, such that it is stretched over the entire terrain piece. (This is for purposes of discussion only, the map can easily be subdivided arbitrarily to yield what-

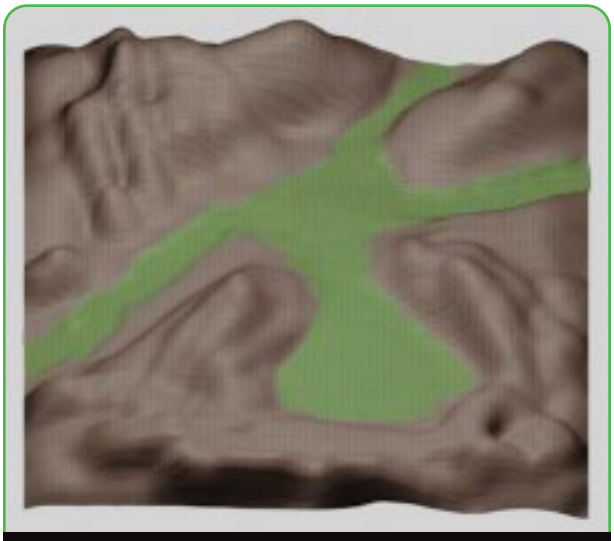


FIGURE 4. *This terrain has been dynamically sculpted by hand and took under five minutes to create.*

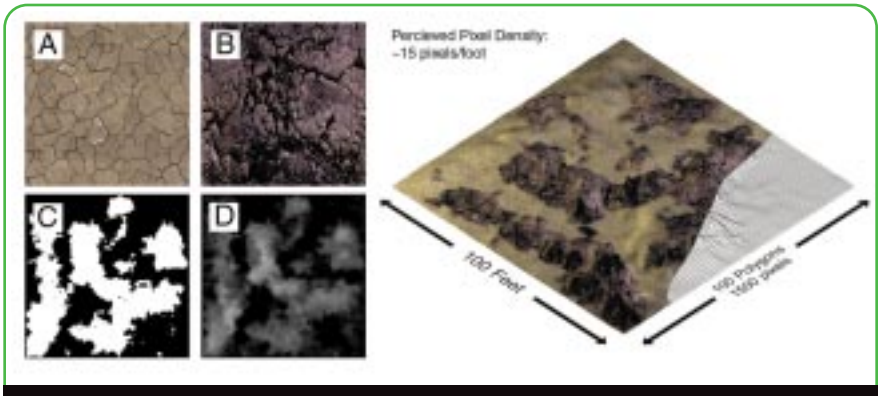


FIGURE 5. *Multi-pass texture blending.*

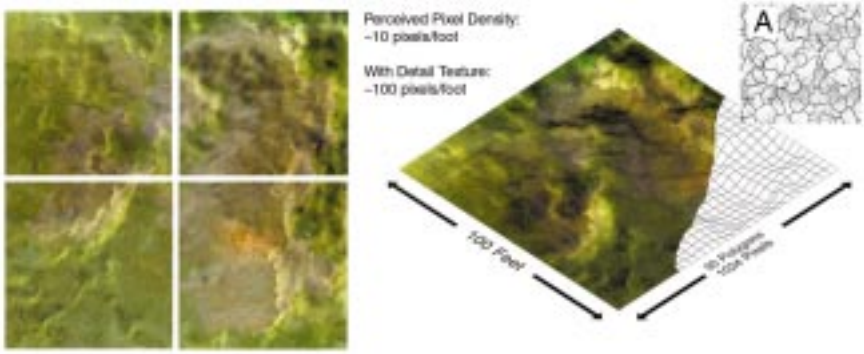


FIGURE 6. *Large-scale maps with detail textures.*

ever appropriate mapping coordinate the terrain will support.) The displacement map (D) used to displace the terrain is synched up with the alpha map.

Taking advantage of the fact that most cards on the market today support at least two-stage rendering, the two texture maps are blended on each group of polygons based on the governing alpha map (C). The actual blending process can occur in at least two different ways. In this example, the terrain has been subdivided aggressively, such that for this terrain piece there are 100 vertices along each axis. At render time each vertex samples into the global alpha map to determine whether it is assigned texture A or B. Where two adjacent vertices have different IDs, a smooth procedural blend is applied. Although it requires a relatively high level of tessellation, this blending technique has the potential to be extremely fast. A second method for implementing this would be to use a lower-tessellation terrain and to render it in multiple passes, double-rendering the blended polygons based on the governing alpha map.

The production advantages for MPTB are threefold. First, only the primary tile pieces need to be derived, so no boundary tiles need to be generated. Since the mapping coordinates for the terrain are implicit, once the primary textures and governing alpha map are completed, the terrain is essentially finished. Second, for a relatively low texture overhead (A and B are each 512×512 pixels), an arbitrarily high pixel density can be achieved on the terrain without many of the tiling artifacts usually seen. Third, endless variations can be generated rapidly simply by changing the governing alpha texture (C), and its associated displacement map (D). This method seems primarily suited for projects requiring massive amounts of terrain where the level of detail is of secondary importance.

Large-Scale Maps with Detail Textures

The second method is more straightforward, focusing more on the visual effect than on the efficiency of the production process. Nevertheless, it too is an expedient and time-saving technique for texturing large sections of terrain. As the name implies, the large-scale map method is just that. Instead of painting the terrain with smaller, repeating tiles, the terrain is textured with larger, unique textures that may or may not tile in any given direction. Removing the tiling restriction gives the texture artist an enormous amount of flexibility in adding detail to the terrain.

As you can see in Figure 6, a mammoth 1024×1024 texture has been applied globally over the entire terrain piece. It has been arbitrarily cut into four pieces, but this does not have to be the case. In fact, the texture map can be as large as the engine or hardware can support. Regardless, depending on the implementation, it should be trivial for the programming team to write an algorithm that subdivides the large texture as necessary. In this case, the texture has been composited using several steps, the final step being to render the texture on a high-resolution mesh which has been displaced with the same displacement map as that of the final in-game terrain. In this way, much of the detail from the high-resolution mesh remains "baked" into the texture in the form of shadows and lighting information. Still, the detail in the terrain is ultimately limited, since the large texture is being stretched over such a massive area. To address this, the perceived pixel density is boosted by the addition of a detail texture (A). This 512×512 texture is composited onto the global texture, and is tiled five times in each direction, greatly increasing the pixel density and thereby the perceived texture detail. Clearly most of the time here is spent in generating the high-resolution texture map. As such, this method is primarily suited to projects which need a high level of polish and detail in the terrain, but which do not require massive amounts of terrain to be generated.

Process and Execution

Almost without exception, developers get to the ends of their projects wishing they had more time to work on the things they feel are important. In many cases, this perceived lack of time in the schedule results from the methods and techniques of content creation remaining in flux too far into the development cycle. Subsequently, it would seem of paramount importance that the artists, designers, and programmers agree upon — and rigidly adhere to — the set of development methods to be used well in advance of the production process. Having the foresight to identify the best methods is a talent that comes with experience, and depends largely on the technical expertise of the developers. It is expedient, then, for us as 3D artists and animators, to become experts with our respective toolsets, so that when called upon, we can provide the necessary expertise in our field. ■

Acknowledgements

Special thanks to Dave Coathupe, Stuart Denman, Wyeth Ridgway, and Peter Watje.

The Burdens of New Technology

It's that time of the year again. You know, when every graphics chip company seems to have a hot-ticket technology for the next wave of 3D games. As we enter the new millennium, the significant gambits in 3D graphics on the PC appear to be hardware-accelerated transformation and lighting (T&L) from the likes of Nvidia and S3, and the magic of T-buffer from 3dfx. There's even talk of something called the X-Box from Microsoft, a very PC way of emulating the closed-box advantages of a console. I can't tell you what hot technology is worth backing all the way, but it may not be that important anyway.

The Players

Foremost in the minds of most graphics companies is the looming launch of the Playstation 2. In many ways, Sony has put the PC industry on the defensive because it promises to deliver both a high-end technology and a mass-market consumer product. With none of the legacy of the PC industry, Sony has redefined the architecture of the Playstation line, and at the same time come up with the kind of polygon counts and bus bandwidths that the PC folks only dream about. Some say that is why Microsoft has been trying to gauge support among game developers for something called the X-Box.

The X-Box is shrouded in rumor, innuendo, and speculation. What seems to be clear is that it's at heart a Windows PC, but the components of the box are fixed, meaning they are predetermined and not upgradable, making the X-Box a competitive to the next-generation consoles. It's a fixed function PC that could be available for under $300 retail. Probably the most anticipated aspect of the X-Box design is the graphics, because that's where Sony's Emotion Engine and Graphics Synthesizer stand out. Graphics seem to be the linchpin of any consumer computing strategy right now, but then again the promise of 3D is enough to assure any hardware developers plenty of media coverage and e-zine speculation.

So far, two companies' wares have been touted as X-Box graphics candidates: Nvidia's GeForce 256 and 3dfx's next-generation product, code-named Napalm or Voodoo 4. As far as the GeForce 256 is concerned, the facts are very clear because the product is already out. This is Nvidia's flagship graphics architecture for at least the next 18 months. It's the first chipset to have hardware-accelerated transformation and lighting integrated, and it is Nvidia's newly anointed Graphics Processing Unit (GPU). The GPU designation is telling; Nvidia wants to make sure that people outside of the graphics industry appreciate the complexity and power of next-generation graphics hardware. Modern day graphics chips are big, complicated pieces of silicon, even though the CPU gets all the attention.

GeForce is a consumer-level product, albeit at the high end of the price scale. However, if you think of it as also being competitive enough in performance and features for inclusion in PC workstations used in CAD or digital content creation, then it's a bargain (there is a GeForce derivative called Quadro that Nvidia recommends for the workstation market). Nvidia has shown that by sticking to DirectX and OpenGL it can get both the recognition of avid gamers, who tend to be the early adopters of 3D graphics accelerators, and OEMs, who tend to care little about gamers' feelings and just want to get enough features and benchmark numbers to place their products. The most interesting point for game developers is that Nvidia has been very aggressive in touting the benefits of DirectX 7's hardware T&L support, although in practice the benefits of the technology haven't been immediately apparent in the benchmarks. In addition, Nvidia is also highlighting its GPU's cube environment mapping technology, but it's the inclusion of hardware T&L that's a watershed in 3D graphics hardware. Game developers are now faced with hardware that does its stuff further up the 3D graphics pipeline, and the hardware makers want to get them to hand over the vertices.

S3 is another company that supports hardware-accelerated T&L, in its Savage 2000 chipset. S3 is less sanguine about the technology, but supports it more out of a realization that its real benefits are still some way off in the future when the development community starts to support it earnestly. It must be pointed out that S3 hired some game designers and created four levels for QUAKE 3: ARENA, ostensibly to show what the benefits of hardware T&L could be in a real-world application. S3 was also one of the first companies to put its weight behind texture compression, and S3's implementation, S3TC, has found its way into DirectX, and will apparently be included in Nintendo's next-genera-

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

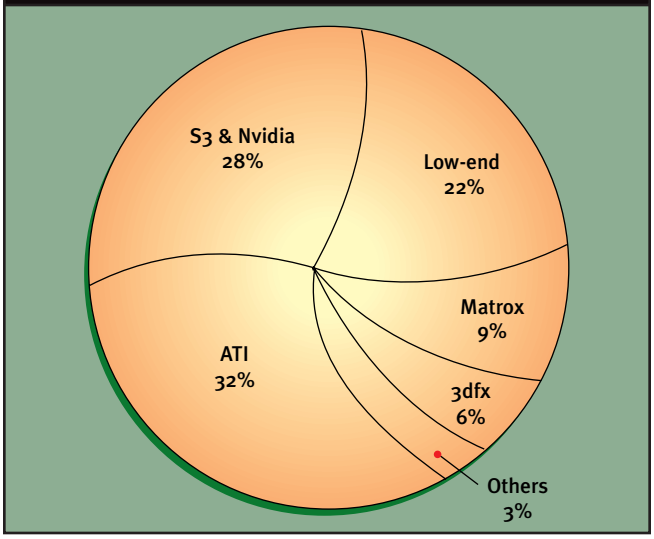
tion Dolphin system. So there's a precedent which implies that if you build it, they will in fact come. Companies like S3 and Nvidia will be under some pressure in the next year to show where their technologies stand against those of Sony. It may be an unrealistic comparison, but at issue is game developer support for advances in PC graphics technology, over investments in Sony's vision. It doesn't necessarily mean that those developers who jump on the Sony bandwagon will abandon the PC, but it could mean that the 3D you see on a console might be ahead of what you'll see on a PC. With Internet connectivity available to both platforms, that's going to leave the PC in a secondary role as a home computing platform, something that has never happened.

As we went to press, it was still unclear what the next-generation 3dfx graphics chip would look like, but the company has been touting one technology sure to be included in the design: the T-buffer. The 3dfx T-buffer provides a number of new, never-before-seen effects on consumer-level PCs. These include entire full-scene spatial anti-aliasing, motion blur, and depth of field. The company acknowledges that the T-buffer owes its existence to the heritage of the accumulation buffer, developed at SGI. An accumulation buffer in an OpenGL workstation provides an additional buffer to integrate multiple renderings of a scene. The T-buffer essentially does the same thing as an accumulation buffer, combining multiple different images to create its effects, however it does it at significantly less cost, and it is designed for the more frame-sensitive world of PC gaming. The T-buffer's results are impressive, and you can see how 3dfx has managed to go right to the heart of issues that would attract the creative aspects of game development. It certainly isn't an orthodox path that the company is taking, but that's where 3dfx has been at its strongest in the past.

Image Is Everything

So there you have it, some excellent technologies and some compelling reasons to use them. Yet the main problem with great hardware technologies remains that while they may deliver compelling visuals, they invariably require investments in time and effort. Furthermore, there are always different flavors of 3D technology (Nvidia's and S3's implementation of hardware T&L being a prime example), therefore developers have to make room for the cost of supporting more than one graphics chip architecture. It could be argued that in the case of hardware-accelerated transformation and lighting, graphics chip vendors are merely riding the coattails of DirectX 7. DirectX 7 supports hardware-accelerated transformation and lighting, and that should make it ubiquitous. Of course, no two Direct3D drivers are the same. In 3dfx's case, the company commands a very loyal and large following of game enthusiasts among consumers and developers alike, possibly because of the benefits of its proprietary Glide API and its support of Macintosh and Linux.

CHART 1. *Market share statistics for desktop graphics chips in 1998 (source: International Data Corp., 1999).*



The differences that exist between the philosophies of Nvidia, S3, 3dfx, and every other graphics chip vendor mirror what has taken place at the highest end of the graphics business. Progress in consumer 3D graphics is following the same path as the evolution of workstation 3D graphics. In the PC workstation market, the CPU has long been a recognized bottleneck for the graphics subsystem. 3Dlabs, probably the market leader in this segment, has provided hardware geometry support through its Gamma chipset, but that hasn't translated into an overwhelming technical or performance leadership for the company, even against those companies that have eschewed geometry hardware. One reason may be that geometry processing does help certain applications in the high end where image quality is of importance, but in some applications, such as CAD, it has little impact on line drawing speeds. For game developers, the same situation could arise. For example, hardware-accelerated T&L favors static images more than it does dynamic 3D ones, therefore racing and flight sims would be the genres that would see immediate benefits of using Nvidia's and S3's hardware. In racing and flight sims the main light source is usually the sun, and the terrain remains static for the most part.

In the case of the T-buffer, it's more difficult to predict the potential. Certainly the results of using some of T-buffer's technologies are immediate. Depth of field and motion blur create effects that you only see in photography, and will require some thought on implementation. For example, if a game scene requires that the camera zoom in on a foreground object, and the T-buffer allows the depth of field effect to blur the background much as a real camera lens would, how does a developer account for the same effect in almost every other graphics accelerator that won't have T-buffer? It may not matter. 3dfx's following among game developers and the "cool factor" of its technologies may carry it through.

Ultimately, game developers will want to know about the installed base and the effects of these technologies on development, support, and game sales. Like every other graphics technology that comes around, the decision is probably going to be financial. This is true even for X-Box, where Microsoft has to convince the development community that it can create a market for an alternative PC-based gaming platform. This should all sound familiar to the hardened veterans of 3D gaming.

The Technology Growth Curve

I have always found the amount of data available on the installed base of PCs to be less than satisfactory for most game developers' uses. Unfortunately, most market research on the PC industry is designed for the benefit of manufacturers and vendors of hardware, who prefer to see sales projections rather than any great focus on legacy systems. In this regard, the console industry offers clear advantages because you can guarantee that for every console sold there is a ratio of game titles per system also sold, and you certainly have little to fear about diverging hardware specifications. Of course, this is a problem that Microsoft could address with X-Box, but until more details emerge, we can only speculate.

Looking at things purely from a graphics-chip standpoint, 3dfx sells its hardware primarily through retail channels which means that it is reaching an after-market of users. As a result, if you were to consider only the installed base and growth of Glide systems, you are probably looking at a shrinking market. The big PC OEMs are buying their graphics chips from ATI, Nvidia, and S3, and there is an overwhelming number of non-brand PC makers, systems integrators, and resellers to take into consideration as well. Still, 3dfx is selling to a very dedicated retail audience. 3dfx graphics consumers are probably the most easily defined demographic in the industry — they're bound to have an interest in games. 3dfx is the only real upgrade 3D accelerator for Macintosh users and was the first product for Linux gaming enthusiasts.

It's fair to say that any projected numbers for the sales of graphics boards with hardware T&L or T-buffer would not tell the whole story. I don't know of any numbers that would give game developers some guidance on which of these technologies would be the most widely adopted. You can be certain that T&L will find its way into more PCs than T-buffer technology because Nvidia and S3 sell a hell of a lot more chips than 3dfx. However, I've yet to see any realistic assessments on where those chips are ending up, whether it's in the home, at work, or for what purpose.

Nevertheless, I think you can get a pretty good feel for the market from some of the information here. The data provided by IDC in Chart 1 is for 1998, but I doubt very much that the order of things will change. I expect ATI, S3, Nvidia, Matrox, and 3dfx to be the main brand leaders. However, I took the liberty of lumping together SiS, Cirrus Logic, Intel, and Trident numbers under the heading of "low end." Obviously, there is still a significant percentage of the market that is less-than-3D if you will, although it's safe to assume that the majority of these products are finding their way into the corporate market. It's too early to tell what impact the low-cost PC business will have on the penetration of the better 3D products from Nvidia, S3, and 3dfx. The bright spot is that at least Nvidia and S3 are committed to migrating their 3D technologies into these lower-cost systems, meaning that hardware T&L will eventually find its way into a broader range of systems, but that is at least two or three years away. Probably the next development to look for is whether ATI or Matrox adopt hardware T&L. If and when they do, there should be little to hold back a game development move up the 3D graphics pipeline. ■

Back

Subdivision Surface Theory

Images from *Geri's Game* courtesy of Pixar Inc.

by Brian Sharp

About 18 months ago at Siggraph '98, Pixar unveiled a short animated film. Christened *Geri's Game*, it was, to quote its Academy Award press release, the "endearing tale of an aging codger who likes to play chess in the park against himself." Not only was it artistically stunning, but it was also a technological powerhouse. The short served as a vehicle to demonstrate Pixar's latest addition to its production environment, a surface scheme known as subdivision surfaces.

Subdivision surfaces are a way to describe a surface using a polygonal model. Like the polygonal model, the surface can be of any shape or size — it's not limited to a rectangular patch. Unlike that polygonal model, the surface itself is perfectly smooth. Subdivision surface schemes allow you to take the original polygonal model and produce an approximation of the surface by adding vertices and subdividing existing polygons. The approximation can be as coarse or as detailed as your needs allow. Because Pixar's rendering system requires everything to be broken into polygons that are a half-pixel across, subdivision surfaces allowed them to tessellate automatically to that level everywhere. As such, the artists didn't need to worry about how close Geri was to the camera. While your game probably can't quite deal with half-pixel polygons, whatever size you do choose, your models can scale up and down in polygon count with the speed of the machine and their distance from the camera.

The technology itself is, for the most part, not new, but its application up until recently has been fairly limited. Indeed, *Geri's Game* is still one of the only compelling demonstrations of subdivision surfaces. Nonetheless, it brought attention to subdivision surfaces

When he's not sleeping through meetings or plotting to take over the world, Brian's busy furtively subdividing, hoping one day to develop his own well-defined tangent plane. Critique his continuity at bsharp@acm.org.

as a relatively new, up-and-coming technique for implementing scalable geometry.

Along with Pixar's work, quite a few researchers are actively tackling issues in the area of subdivision surfaces, and several Siggraph papers each year advance them academically and put them to use in solving problems. By now, they are a fairly mature technology, and a compelling contender among scalability solutions.

The game development community realizes that scalable geometry techniques are an important part of developing next-generation game engines. The spread between high-end and low-end hardware seems to get bigger each year (thanks to current and forthcoming geometry accelerators such as Nvidia's GeForce 256 and S3's Savage2000), forcing game developers to find ways to cater to the masses that use low-end machines while building in features that make the most of hardcore gamers' advanced hardware. As a result, on the low end our engines should still be capable of using fewer than 10,000 polygons per scene, but on the high end, the sky's the limit: even hundreds of thousands of polygons per scene can cruise along at 60 frames per second. Scalable geometry techniques such as subdivision surfaces are therefore necessary to accommodate this variation in hardware capabilities.

In this article, a number of different kinds of subdivision surfaces will be discussed. As a preliminary warning, this article is entirely theory. Next month, we'll look at an example implementation of one of the schemes, the modified butterfly, which I'll discuss here. Keep in mind as you read this article that not every concept described here will be practical for use in your engine. Indeed, some subdivision surface models may not be feasible for use in games at all. But knowing the strengths and weaknesses of the various models will help you make the right decision for your next game.

The What and the Why

First, what is a subdivision surface? The obvious answer is that it's a surface generated through subdivision. To elaborate, every subdivision surface starts with an original polygonal surface, called a control net. Then the surface is subdivided into additional polygons and all the vertices are moved according to some set of rules. The rules for moving the vertices are different from scheme to scheme, and it is these rules that determine the properties of the surface. The rules of most schemes (including all the ones discussed here) involve keeping the old vertices around, optionally moving them, and introducing new vertices. There are schemes that remove the old vertices at each step, but they're in the definite minority.

The one thing the control net and the eventual surface (called the limit surface) have in common is that they are topologically the same. Topology is a way of describing the structure of a surface that isn't changed by an elastic deformation, that is, a stretching or twisting. A good example and common joke is that to a topologist, a coffee cup and a donut are identical. The donut hole corresponds to the hole in the handle of the coffee mug. On the other hand, a sphere and coffee mug are not topologically equivalent, since no amount of stretching and twisting can punch a hole in that sphere.

Topology is one reason that subdivision surfaces are worth a look. With Bézier or B-spline patches, modeling complex surfaces amounts to trying to cover them with pieces of rectangular cloth. It's not easy, and often not possible if you don't make some of the patch edges degenerate (yielding triangular patches). Furthermore, trying to animate that object can make continuity very difficult, and if you're not very careful, your model will show creases and artifacts near patch seams.

That's where subdivision surfaces come in. You can make a subdivision surface out of any arbitrary (preferably closed) mesh, which means that subdivision surfaces can consist of arbitrary topology. On top of that, since the mesh produces a single surface, you can animate the control net without worrying about seams or other continuity issues.

As far as actual uses in games, I believe that subdivision surfaces are an ideal solution for character modeling. Environments and other parts of a game generally don't have the fine detail or strange topology that would require subdivision surfaces, but characters can have joint areas that are particularly hard to model with patches, and characters are in constant animation, which makes maintaining continuity conditions very important.

THE BASICS. Before we start discussing individual schemes, let's look at the basic characteristics of subdivision surfaces in general. This gives us a framework for classifying and comparing the schemes as we come across them. Most of these characteristics carry notable implications with them, whether they are implied computational costs or implied ease-of-use considerations, or anything else. These will usually be the criteria on which you might choose one scheme above another.

CONTINUITY: THE HOLY GRAIL. The first characteristic of a scheme is its continuity. Schemes are referred to as having C^n continuity, where n determines how many derivatives are continuous. So if a surface is C^0 continuous, it means that no derivatives are continuous, that the surface itself doesn't have open holes. If a surface is C^1 continuous, it means that the surface is closed and that its tangents are continuous (so there aren't any sharp seams).

This probably won't be a major selling point of one scheme above another, since just about every scheme has C^1 continuity everywhere. Some have C^2 continuity in some places, but the majority have areas where the best they can claim is C^1. So most schemes are alike in this regard.

However, continuity is most certainly worth mentioning because it's one of the major reasons to think about using subdivision surfaces in the first place. After all, Pixar could have modeled Geri using as many polygons as they wanted, since they're not running their movies in real time. But no matter how many polygons they used, you could get close enough that Geri's skin would look faceted from the polygons. The point of using a subdivision model is that you have that ideal limit surface at which you can always throw more and more polygons as you get closer and closer to, no matter how high the display resolution or how close the model is to the screen. Only a very small portion of the real world is flat with sharp edges. For everything else, there's subdivision surfaces.

To Interpolate or not to Interpolate...

While the degree of continuity is generally the same for all subdivision schemes, there are a number of characteristics that vary notably between schemes. One important aspect of a scheme is whether it is an approximating scheme or an interpolating scheme. If it's an approximating scheme, it means that the vertices of the control net don't lie on the surface itself. So, at each step of subdivision, the existing vertices in the control net are moved closer to the limit surface. The benefit of an approximating scheme is that the resulting surface tends to be very fair, having few undulations and ripples. Even if the control net is of very high frequency with sharp points, the scheme will tend to smooth it out, as the sharpest points move the furthest onto the limit surface. On the other hand, this can be to the approximating scheme's detriment, too. It can be difficult to work with, as it's harder to envision the end result while building a control net, and it may be hard to craft more undulating, rippling surfaces as the scheme fights to smooth them out.

If it's an interpolating scheme, it means that the vertices of the control net actually lie on the limit surface. This means that at each recursive step, the existing vertices of the control net are not moved. The benefit of this is that it can be much more obvious from a control net what the limit surface will look like, since the control net vertices are all on the surface. However, it can sometimes be deceptively difficult to get an interpolating surface to look just the way you want, as the surface can develop unsightly bulges in areas where it strains to interpolate the vertices and still maintain its continuity. Nonetheless, this is usually not a tremendous problem.

Figure 1 shows examples of an approximating scheme (on the left) and an interpolating scheme (on the right). The white outline is the control net, and the red wireframe is the resulting surface after a few subdivision steps. You can see the difference quite clearly: the approximating surface seems to pull away from the net, while the interpolating surface flows through the vertices of the net.

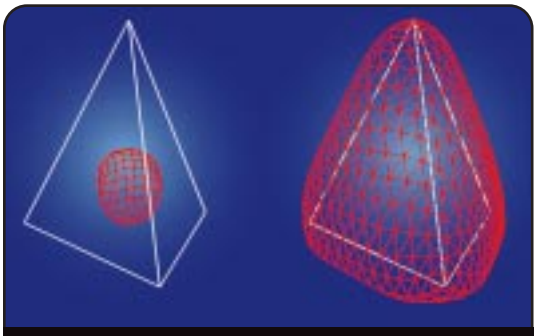


FIGURE 1. *Two schemes subdivide a tetrahedron. The left scheme is approximating, and the right is interpolating.*

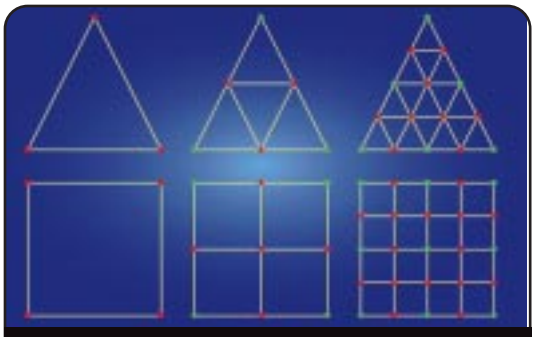


FIGURE 2. *The differences between the tessellation used by a triangular scheme (top) and a quadrilateral scheme (bottom).*

Surfaces in Uniform

Another set of characteristics of a scheme brings in four more terms. A scheme can be either uniform or nonuniform, and it can be either stationary or nonstationary. These terms describe how the rules of the scheme are applied to the surface. If the scheme is uniform, it means that all areas of a control net are subdivided using the same set of rules, whereas a nonuniform scheme might subdivide one edge one way and another edge another way. If a scheme is stationary, it means that the same set of rules is used to subdivide the net at each step. A nonstationary scheme, on the other hand, might first subdivide the net one way, and then the next time around use a different set of rules.

All the schemes we'll talk about here are fundamentally both uniform and stationary. There are some extensions to these schemes that make them nonstationary or nonuniform, but there aren't many subdivision schemes that are fundamentally nonstationary or nonuniform. One of the main reasons for this is that most of the mathematical tools we have for analyzing schemes are unable to deal with dynamically changing rules sets.

Subdivision Shape

Another characteristic of a scheme, albeit less significant than the prior ones, is whether it is triangular or quadrilateral. As the names would imply, a triangular scheme operates on triangular control nets, and a quadrilateral scheme operates on quadrilateral nets. Clearly, it would be inconvenient if you had to restrict yourself to these primitives when building models. Therefore, most quadrilateral schemes (including the one discussed here) have rules for subdividing *n*-sided polygons. For triangular schemes, you generally need to split the polygons into triangles before handing them over to be subdivided. This is easy enough to do, but one downside is that for some schemes, the way you break your polygons into triangles can change the limit surface. The changes are usually minor, though, so you simply need to be consistent: if you randomly choose which diagonal of a quadrilateral to split on every frame, you'll end up with popping artifacts.

Figure 2 shows examples of a triangular subdivision scheme as compared to a quadrilateral scheme. Notice that the triangular scheme only adds new vertices along the edges, whereas the quadrilateral scheme needs to add a vertex in the center of each face. This is one reason why triangular schemes tend to be somewhat easier to understand: their rules have that one fewer step in them.

Extraordinary Vertices

The preferred vertex valence is another property of subdivision schemes. The valence of a vertex is the number of edges coming out of it.

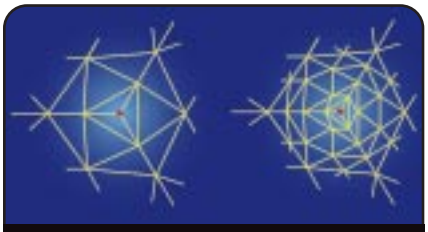


FIGURE 3. *A triangular net (left) and after one subdivision step (right). The red vertex is extraordinary.*

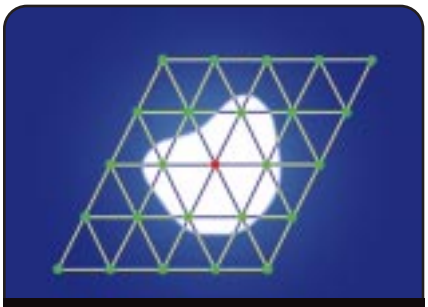


FIGURE 4. *A hypothetical mask. Here, the white region is a mask used to dictate which vertices are used in a computation involving the red vertex.*

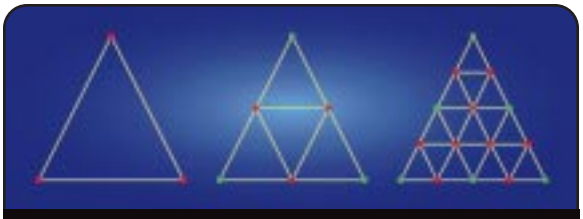


FIGURE 5. *Two steps of subdividing a triangle with the polyhedral scheme.*

Most every vertex a scheme produces during subdivision has the same valence. Vertices of that valence are the regular vertices of a scheme. Vertices of any other valence are known as extraordinary vertices. Their effect depends on the subdivision scheme, but historically there have been problems analyzing the limit surface near extraordinary vertices. As we look at various schemes, we'll see the effect that extraordinary vertices have on each one.

Most schemes don't ever produce extraordinary vertices during subdivision, so the number of extraordinary vertices is set by the original control net and never changes. Figure 3 is an example of two steps of a triangular scheme with an extraordinary vertex in the center. Notice how it remains the only extraordinary vertex after a step of subdivision. Also note that the valence of the regular vertices is 6. This is common for triangular schemes, as they all tend to split the triangles in the same way — by adding new vertices along the edges and breaking each triangle into four smaller triangles.

Surface Evaluation

Surface evaluation is the process of taking a control net, adding vertices, and breaking faces into more, smaller faces to find a better polygonal approximation of the limit surface. There are a number of ways to evaluate a subdivision surface. All subdivision schemes can be evaluated recursively. Furthermore, most (including all the ones discussed here) can be explicitly evaluated at the vertex points of the control net. For interpolating schemes, this means that you can explicitly calculate the surface normals at the vertices using what are called tangent masks. For approximating schemes it means you can also explicitly calculate the vertex's limit position, using what are called evaluation masks. In this context, a mask isn't the same kind of mask that you might use during binary arithmetic. Our masks are more analogous to the masks worn at a masquerade. They are like stencil cutouts, shapes that can be "placed" on the control net, and their shape determines which of the surrounding vertices are taken into account (and how much effect each has) in determining the end result, be it the vertex location or its tangent vectors. Figure 4 shows a visual example of applying a mask to a surface at a vertex.

An important aspect of evaluation is the scheme's support. The support refers to the size of the region considered during evaluation. A scheme is said to have compact support if it doesn't have to look very far from the evaluation point. Compact support is generally desirable because it means that changes to a surface are local — they don't affect the surface farther away.

A Note on Notation

Since the original authors of many subdivision schemes weren't operating in concert with one another, the notation used between schemes tends to vary fairly wildly. Here, I've tried to stick with a fairly consistent notation. When talking about a specific vertex, it is v. If it matters what level of recursion it's at, that level i is indicated as a superscript, so the vertex is v^i. The vertex's valence is N. The neighboring vertices of the vertex are e_j where j is in the range $[0,N-1]$. Again, if the level of recursion matters, that level i is a superscript, so e^i_j is the jth edge vertex at level i. I try to use this notation everywhere, but there are a few places where it's much clearer to use a different notation.

The one problem with a standard notation is that if you access some of the references at the end of this article, they will very likely use their own, different notation. As long as the concepts make sense, though, it shouldn't be difficult to figure out someone else's naming convention.

The Polyhedral Scheme

The polyhedral scheme is about the simplest subdivision scheme of all, which makes it a good didactic tool but not the kind of scheme you'd ever actually want to use. It's a triangular scheme where you subdivide by adding new vertices along the midpoints of each edge, and then break each existing triangle into four triangles using the new edge vertices. A simple example is shown in Figure 5. The problem with this, of course, is that it doesn't produce smooth surfaces. It doesn't even change the shape of the control net at all. But it serves to demonstrate some concepts fairly well.

The scheme is clearly interpolating since it doesn't move the vertices once they're created. It's also triangular, since it operates on a triangular mesh. Furthermore, the scheme is uniform since the edge's location doesn't affect the rules used to subdivide it, and stationary since the same midpoint subdivision is used over and over. The surface is only C^0 continuous, since along the edges of polygons it doesn't have a well-defined tangent plane. The regular ver-

tices of this scheme are of valence 6, as that's the valence of new vertices created by the scheme. However, this scheme is simple enough that it doesn't suffer because of its extraordinary vertices.

The evaluation of the scheme isn't hard at all. You can evaluate it recursively using the subdivision rules. As far as evaluation and tangent masks go, it's clear that we don't need an evaluation mask, since the points are already on the limit surface. Tangent masks don't really make any sense, since our surface isn't smooth and therefore doesn't have well-defined tangents everywhere.

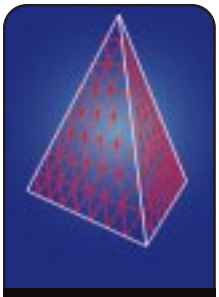


FIGURE 6. *A tetrahedron control net (in white) and a polygonal surface approximation (in red) produced using the polyhedral scheme.*

Figure 6 shows a tetrahedron control net in white with a red wireframe of the surface after a few subdivision steps of the polyhedral scheme.

Float Like a Butterfly...

The next scheme is known as the butterfly subdivision scheme, or, in its current form, the modified butterfly scheme. It shares some similarities with the polyhedral scheme, but has some differences, notably that it's C^1 continuous and therefore actually produces a smooth surface.

The butterfly scheme has a fairly interesting history to it. In 1990, Dyn, Levin, and Gregory published a paper titled "A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control" (see For Further Info box, p. 42). It described the first butterfly scheme. The title is derived from the stencil, or map of neighbors used during evaluation, which is shaped like a butterfly (Figure 7). The scheme is interpolating and triangular, so all it ever does is add vertices along the edges of existing triangles. The rules for adding those vertices are simple, and the support is compact. For each edge, sum up the vertices in the stencil-shaped area around that edge, weighting each one by a predetermined weight. The result is the new vertex. The weights used, corresponding to the vertex labelings in Figure 7, are these:

$$a:\frac{1}{2},\ b:\frac{1}{8}+2w,\ c:-\frac{1}{16}-w$$

In this case, w is a tension parameter, which controls how "tightly" the limit surface is pulled towards the control net — note that if w equals –1/16, the scheme simply linearly interpolates the endpoints and the surface isn't smooth.

One question that the scheme doesn't answer, though, is what to do if the area around an edge doesn't look like that butterfly stencil. Specifically, if either of the edges' endpoints is of a valence less than 5, there isn't sufficient information to use the scheme, leaving you with no choice but to choose $w = -1/16$ near that area, resulting in a surface that

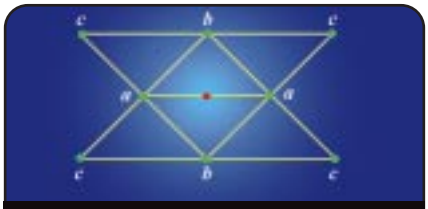


FIGURE 7. *The 8-point stencil for the original Butterfly scheme.*

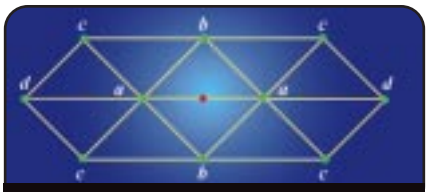


FIGURE 8. *The 10-point stencil from the modified butterfly scheme.*

isn't smooth near those extraordinary points. This means that while the surface is smooth almost everywhere, there will be isolated jagged points that really stand out visually and make the surface harder for an artist to craft.

In 1993, Dyn and his colleagues extended the butterfly scheme to use a ten-point stencil, so that the default case was the one shown in Figure 8, similar to the eight-point case with the rear vertices added in. The new weights are:

$$a:\frac{1}{2}-w,\ b:\frac{1}{8}+2w,\ c:-\frac{1}{16}-w,\ d:w$$

Note that by adding w to the d points and subtracting it from the a points, the stencil's total weighting still adds up to 1. Intuitively, this is important because it means that the new point will be in the neighborhood of the ones used to generate it. If the weights summed to, say, 2, then the point would be twice as far from the origin as the points used to generate it, which would be undesirable.

This new scheme even reduces to the old scheme as a subset — choosing $w = 0$ results in the same rule set as the eight-point butterfly stencil. However, this extension didn't address the smoothness problem at extraordinary vertices.

In 1996, Zorin, Schröder, and Sweldens published an extension of the butterfly scheme known as the modified butterfly scheme. The primary intent of their extension was to develop rules to use for extraordinary vertices, making the surface C^1 continuous everywhere.

If both of the endpoints of the edge are regular valence-6 vertices, the scheme uses the standard butterfly's ten-point stencil with the same weights.

If only one of the endpoints is extraordinary, the new vertex is computed by the weighted sum of the extraordinary vertex and its neighbors (see the stencil in Figure 9). Note that you actually do not consider some of the neighbors of the regular vertex in doing this, which might seem a little odd. Given the extraordinary vertex's valence of N, the weights used are:

$$N = 3:\left(v:\frac{3}{4}, e_0:\frac{5}{12}, e_1:-\frac{1}{12}, e_2:-\frac{1}{12}\right)$$

$$N = 4:\left(v:\frac{3}{4}, e_0:\frac{3}{8}, e_1:0, e_2:-\frac{1}{8}, e_3:0\right)$$

$$N \geq 5:\left(v:\frac{3}{4}, e_j:\left(0.25+\cos\left(\frac{2\pi j}{N}\right)+0.5*\cos\left(\frac{4\pi j}{N}\right)\right)/N\right)$$

The full justification for these weights is available in Zorin's thesis (see For Further Info box, p. 42).

If both endpoints of the edge are extraordinary, the vertex is computed by averaging the results produced by each of the

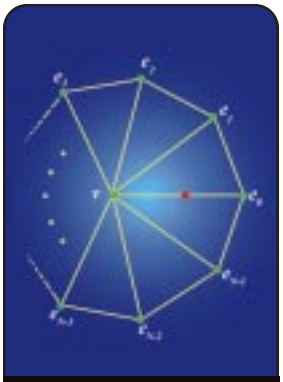


FIGURE 9. *The stencil for an extraordinary vertex in the modified butterfly scheme.*

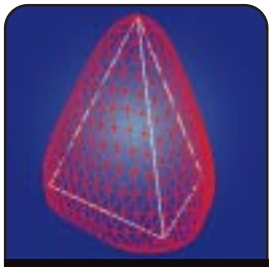


FIGURE 10. *A tetrahedron control net (in white) and a polygonal surface approximation (in red) produced using the modified butterfly scheme.*

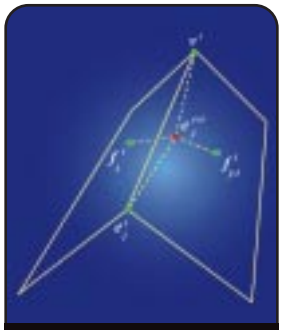


FIGURE 11. *Calculation of a new edge vertex in a Catmull-Clark surface. The new edge vertex is the average of the four points.*

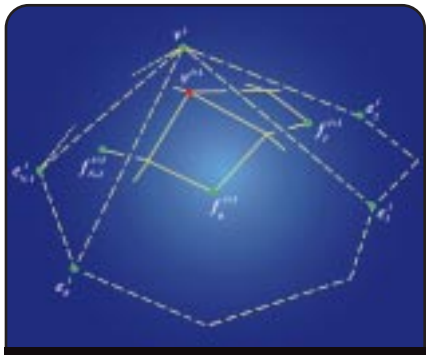


FIGURE 12. *The points used to calculate the new position of a vertex in a Catmull-Clark surface. The points used are in green; the new vertex location is in red.*

endpoints. So, evaluate the vertex once for each endpoint using the appropriate weights from above, and average the resulting two candidates.

Those, then, are the rules for recursively evaluating the surface. Since the scheme is interpolating, you don't need an evaluation mask, but it would be nice to have a tangent mask to explicitly find the tangents at vertices. Such a mask exists, although it's fairly lengthy to write out, and not particularly enlightening. It can be found in Zorin's thesis, and I'll discuss it next month when implementing this scheme.

Figure 10 shows a tetrahedron control net in white with a red wireframe of the surface after a few subdivision steps of the modified butterfly scheme.

Catmull-Clark Surfaces

The final scheme we'll examine has some significant differences from the modified butterfly. Notably, it's quadrilateral and it's approximating, and so presents some new challenges. Its regular vertices are of valence 4, since a regular quadrilateral surface is a rectangular grid with vertices of valence 4.

Because this scheme is quadrilateral, it has to deal with things like placing vertices in the centers of polygons, and the rules are generally a bit more complex. Vertex addition is done in three steps. For each face in the old control net, add a vertex in its center, where the center is found by averaging its vertices. Then, for each edge in the old control net, a new vertex is added equal to the average of the edge's endpoints and the new adjacent face points (see Figure 11 for an illustration).

Finally, move the original vertices of the old control net using neighboring points in the calculation. The stencil is shown in Figure 12; the rules are as follows:

$$v^{i+1} = \frac{N-2}{N} v^i + \frac{1}{N^2}\left(\sum_{j=0}^{N-1}\left(e_j^i + f_j^{i+1}\right)\right)$$

New edges are then formed by connecting each new face point to its adjacent new edge points and connecting each new vertex point to its adjacent new edge points. This defines the faces as well, and it brings up an interesting point: consider what happens when you subdivide a surface with a polygon that is not a quadrilateral. The resulting new face vertex will be connected to k new edge vertices, and k will not be equal to four. Therefore, the new face vertex is an extraordinary vertex. This is the only one of the three schemes shown here where the scheme can actually create an extraordinary vertex during subdivision.

This is not as bad as it may seem, though. After a single subdivision step, all the faces in the control net are quadrilaterals. Therefore, the scheme can only introduce new extraordinary vertices during the first subdivision step. After a single subdivision step, the number of extraordinary vertices is set and will not change.

The scheme also has evaluation and tangent masks for evaluation at the vertices. The full discussion and proof of the evaluation mask can be found in Halstead et al. and is fairly lengthy. The mask itself is fairly simple, though. For a vertex of valence N, the mask is equal to:

$$v^{\infty} = \frac{N^2 v^1 + \sum_{j=0}^{N-1}\left(4e_j^1 + f_j^1\right)}{N(N+5)}$$

It's interesting to note that this mask requires that we've subdivided the net once, since it uses the face and edge vertices of the same level as the corner vertices, and face and edge vertices are not available in the original control net.

The tangent masks carry an equally lengthy discussion, but their resulting formula is also fairly complicated. Because most of it can be precomputed for each valence and stored in a lookup table, it's not computationally expensive, it's just a large formula:

$$A_N = 1 + \cos\left(\frac{2\pi}{N}\right) + \cos\left(\frac{\pi}{N}\right)\sqrt{2\left(9 + \cos\left(2\pi / N\right)\right)}$$

$$t_i = \sum_{j=0}^{N-1}\left(A_N \cos\left(\frac{2\pi j}{N}\right) e^1_{(j+i) \bmod N} + \left(\cos\left(\frac{2\pi j}{N}\right) + \cos\left(\frac{2\pi(j+1)}{N}\right)\right) f^1_{(j+i) \bmod N}\right)$$

The surface normal is then the normalized cross product of t_0 and t_1.

Figure 13 shows a tetrahedron control net in white with a red wireframe of the surface after a few subdivision steps of the Catmull-Clark scheme.

Catmull-Clark Extended

Catmull-Clark surfaces hold the distinction of being the favored surfaces for use in high-end rendering; they were the model employed by Pixar in *Geri's Game*. Their mathematical elegance and the amount of work devoted to them make them a fairly attractive choice. For instance, work has been done on generating Catmull-Clark surfaces that interpolate a set of points, which, as an approximating scheme, they do not usually do. Furthermore, Pixar extended them for *Geri's Game* to allow for sharp and semi-sharp creases in the surface.

Pixar's scheme generating these creases is fairly straightforward. It allows an artist to specify for an edge or vertex that subdivision near that edge or vertex should be done sharply (using polyhedral subdivision) for some number of steps, from 0 to infinity. Intuitively, the more sharp steps that are used, the more creased the surface will appear near that edge. If the number is finite, then the surface will still be smooth, since eventually the surface will resume using the normal Catmull-Clark subdivision rules. If the crease is infinitely sharp, it isn't smooth at all. Pixar put these to use on Geri's skin features, adding creases to various locations across his body like between his skin and fingernails.

FIGURE 13. *A tetrahedron control net (in white) and a polygonal surface approximation (in red) produced using the Catmull-Clark scheme.*

It's worth noting that while this greatly extends the application of the surfaces, it changes the properties of the scheme. The scheme becomes both nonuniform, since different edges and vertices can be of differing degrees of sharpness, and nonstationary, because a semi-sharp crease is evaluated linearly for some number of steps and then smoothly for the rest. Near the creases, the surface no longer reduces to the B-spline surface, and it also invalidates the evaluation and tangent masks.

Geri's Game clearly demonstrates the benefit of sharp and semi-sharp creases. However, for use in games, the evaluation and tangent masks are fairly important, and so it's difficult to say whether the increased computational cost is worth the added functionality.

Are You Dizzy Yet?

After this whirlwind tour of subdivision surfaces, you might be feeling a little light-headed or dizzy. Hopefully though, you've picked up the concepts behind subdivision surfaces and maybe even thought of some good applications for them in projects you're working on or getting ready to start. Since there's nowhere near enough space to discuss implementation details for even just these three schemes, next month we'll bear down and focus on one of them, the modified butterfly scheme. I'll mention the reasons I think it's a good choice for use in games, discuss some of the benefits and detriments, and then present an example implementation.

Until then, there's certainly no dearth of information on subdivision surfaces. Much of it is available online. The ACM Digital Library is an excellent resource for this topic as much of the work in subdivision surfaces has been published in the recent Siggraph conferences. Furthermore, many of the papers, Siggraph or not, are available directly from authors' web sites. ■

Acknowledgements

Thanks to Pixar for graciously allowing us to use images from their short animation, *Geri's Game*. Thanks also to Denis Zorin for his suggestions and references, Jos Stam at Alias|Wavefront for his help and suggestions, and to Alias|Wavefront for allowing him to release his precomputed eigenstructures. Thanks to Chris Goodman of 3dfx for discussions, latté, and those hard-to-find papers, and to Adrian Perez of Carnegie-Mellon University for suggesting the subdivision scheme I eventually settled on.

FOR FURTHER INFO

- Catmull, E., and J. Clark. "Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes." *Computer Aided Design*, 1978.
- DeRose, T., M. Kass, and T. Truong. "Subdivision Surfaces in Character Animation." *Siggraph '98*. pp. 85–94.
- Dyn, N., J. A. Gregory, and D. A. Levin. "Butterfly Subdivision Scheme for Surface Interpolation with Tension Control." *ACM Transactions on Graphics*. Vol. 9, No. 2 (April 1990): pp. 160–169.
- Dyn, N., S. Hed, and D. Levin. "Subdivision Schemes for Surface Interpolation." *Workshop in Computational Geometry* (1993), A. C. et al., Ed.," *World Scientific*, pp. 97–118.
- Halstead, M., M. Kass, and T. DeRose. "Efficient, Fair Interpolation Using Catmull-Clark Surfaces." *Siggraph '93*. p. 35.
- Stollnitz, E., T. DeRose, and D. Salesin. *Wavelets for Computer Graphics*. San Francisco: Morgan-Kaufman, 1996.
- Zorin, D. "Stationary Subdivision and Multiresolution Surface Representations." Ph.D. diss., California Institute of Technology, 1997. (Available at ftp://ftp.cs.caltech.edu/tr/cs-tr-97-32.ps.Z)
- Zorin, D., P. Schröder, and W. Sweldens. "Interpolating Subdivision for Meshes with Arbitrary Topology." *Siggraph '96*. pp. 189–192.

ACM Digital Library
http://www.acm.org/dl

Joe Stam's web site
http://reality.sgi.com/jstam_sea/index.html

Denis Zorin's web site
http://www.mrl.nyu.edu/dzorin

Charles Loop's web site
http://research.microsoft.com/~cloop

Siggraph '99 Subdivision Course Details, Notes, Slides
http://www.mrl.nyu.edu/dzorin/sig99

Geometric Modeling
http://muldoon.cipic.ucdavis.edu/CAGDNotes

Playing for Keeps: Designing a Casino Game

by Steve Boelhouwer

Take a coin chute for the people to put their money in, and a cash box for the money to go into, and put something in between that will interest the people, and you've invented a slot machine.

— Charlie Fey (1862–1944), inventor of the slot machine as we know it today.

Only unawareness, and some persistent Hollywood stereotypes prevents most people from recognizing the tremendous changes that are occurring in the gaming (gambling) machine industry. Long gone are the days when slot machines were placed in casinos only as amusements for the wives of high-rolling craps or blackjack players. Today's casino floors are brimming with high-tech machines that are capable of providing rich, immersive experiences, and the revenue they produce far exceeds that which flows across the green felt gambling tables. This article will explore the history and evolution of the slot machine industry, and provide some insight into the development processes used to create these games. Those from the PC and console game businesses may be surprised to see how similar the development of these devices is to their own profession. We will also delve into the special math considerations involved in creating a successful gaming machine, and take a look into the complex, ubiquitous regulatory structure that oversees most aspects of the industry.

Before we begin, let's debunk some common urban legends regarding slot machines. There is a popular misconception that a game can be made "looser" (made to pay back more) or "tighter" (made to pay back less) simply by turning a screw or knob inside the game cabinet. Others believe that the games are "fixed" to hit big jackpots on predetermined days, such as major holidays or grand openings. This simply isn't the case. The outcome of every handle pull on a modern gaming device is a completely random event (or at least as random as today's technology allows). As we shall see, it's possible to control the overall odds and payback rate on a machine, but not possible (and highly illegal) to control individual game outcomes. Regulators, casino owners, and game manufacturers go to great lengths to maintain the fairness of all games, both in fact and in perception.

Why? Because the importance of electronic gaming to the modern casino industry is enormous. Gaming devices (which include all types of electronic gambling devices, including reel-spinning slot machines, video slots, and electronic versions of live games such as poker, keno, and blackjack) account for nearly 75 percent of all casino revenues, and fill over 80

Steve Boelhouwer is co-art director at Casino Data Systems, a leading designer and manufacturer of gaming devices. He has almost 15 years' experience in the casino gaming industry. *Steve would gratefully like to acknowledge the assistance of fellow art director Kim Tempest and CDS founder and CEO Steve Weiss for their invaluable input into this article.* *Contact Steve at SBoelhouwer@csds.com*

percent of the total casino floor space. It's estimated there are approximately 460,000 gaming devices in legal operation throughout North America and the annual replacement market alone runs around 70,000 units annually. Let's take a look at how the industry grew to what it is today.

Bells and Cherries: A Brief History

Slot machines first appeared on San Francisco's Barbary Coast in the 1890s. California laws of that era prohibited gambling machines that paid jackpots in money, so the games were redesigned as "trade stimulators." For example, if a lucky player lined up matching symbols on the reels, the owner of the establishment would pay the winner ten cigars. The fruit symbols (cherries, plums, and so on) used on the reels of modern slot machines originated from this scheme, as these icons once represented payouts of fruit-flavored chewing gum. It's a safe bet to assume that a dollar or two was paid out instead of cigars or gum when the police weren't around.

The state of Nevada legalized casino gambling in 1931, thereby creating a legal American market for slot machines. Games of that era were completely mechanical, using complex collections of springs, wheels, and gears to drive the spinning reels. Mechanical games were the norm until the early 1960s, when Bally Manufacturing introduced Money Honey, the industry's first electromechanical slot machine. The game was a huge success. Computerized reel-spinning slot machines were introduced in 1981, and video-display games were introduced during this period as well. Today, virtually all legal gaming devices in the United States are microprocessor-based, whether they spin reels or blast pixels onto a video screen.

In the early 1990s, the convergence of two events had another profound impact on the industry. First, legalized casino gaming exploded beyond its historical boundaries of Nevada and Atlantic City. Many local and state governments were looking for ways to increase tax revenues, and gaming seemed like an easy way to fill the public coffers. Riverboat casinos were launched on the Mississippi River at a pace that would have made Mark Twain proud. At the same time, new federal rulings allowed for a tremendous expansion of gaming on Native American lands. (The world's largest casino, Foxwoods, is owned by the Pequot tribe in Leyward, Conn.) The result of this surge in demand was the slot makers now had substantial amounts of cash to fuel further R&D efforts.

This game is an example of a "traditional" reel-spinning slot machine. This particular model has an overhead wheel and LED panel to display game messages and bonus information.

The PC gaming industry was experiencing a boom of a different sort during that era. "Multimedia" was the buzzword of the day, and the impact that the introduction of the CD-ROM had on the computer gaming world needs no repeating here. Forward-thinking slot manufacturers realized that a similar revolution could be carried over to their industry as well, providing the means to increase the entertainment value offered by their games drastically.

Technological Changes

Before this innovation could begin, some fundamental changes to the standard game architecture had to be made. Traditionally, gaming devices are ROM-based, with all game code, graphics, and sound residing in programmable, read-only memory modules (EPROMs). This architecture, which is still in wide use today, is rugged and has certain security advantages. The security aspect of burning all of the game control code into nonvolatile EPROMs is particularly important, given the highly regulated nature of the industry (I'll expand on this later), and the huge amounts of money that can be at stake. Nonetheless, this architecture has all of the inherent limitations that are associated with classic coin-op arcade games. Most EPROM-based system boards operate at very slow clock speeds, support only a limited amount of memory, and have basic graphics capabilities at best (typically 4-bit color).

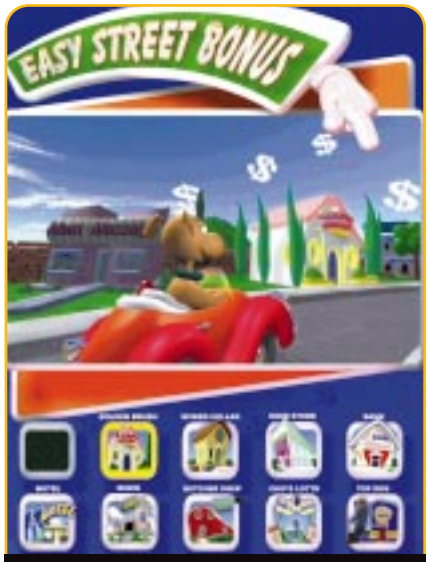
Programming tools for these platforms are typically limited to simple DOS-prompt linkers and C compilers. As such, it is difficult to enhance the player experience significantly within the constraints of this environment. The challenge was to develop a new platform that could still provide the security and reliability of an EPROM-based game, and at the same time allow for vastly improved graphics, sound, and interaction.

The industry took several different approaches to this challenge. In 1997, the state of Nevada approved a new platform based on PC-style hardware (all gaming devices must pass regulatory muster before they can legally be offered for play). This device utilized a Pentium processor, a hard disk, and a full-color graphics subsystem to deliver content. All game code and media assets were encrypted and stored on the hard disk, which was jumpered to prevent unauthorized writing to the drive. While the games offered on this platform were traditional (slots, video

poker, and video keno), the increased graphics capabilities, professional-quality animation, and high-caliber sound offered a considerably modernized playing experience. However, many gaming jurisdictions perceived problems with the security of this architecture, and further approvals were slow in coming.

A more rounded approach was later developed by my employer, Casino Data Systems (CDS), which keeps all game control functions (the code which controls random number generation, win decoding, money handling, security, and accounting functions) in EPROM and utilizes a PC-based system for storage and execution of the multimedia functions only. With the platforms in place, titles could now be developed which took advantage of the increased capabilities.

The main reel screen from CDS's Easy Street slot game.



The second-level bonus screen from CDS's Easy Street video slot game.

Game Development and the Target Marketplace

In broad terms, the development of gambling games is not significantly different from the development cycle used in the PC, console, and coin-op games industries. A market segment is identified and targeted, a specification is defined, prototypes are produced, tested, and refined, and if all is successful the title is moved into full production. The unique aspects of game development in this industry involve special math considerations and meeting the regulatory requirements that address almost all aspects of a game. These regulations can have some unusual side effects on the day-to-day production work. Art, programming, and business have always been somewhat tentative bedfellows; adding government to the mix can really shake things up.

The overall demographics of the gaming market are not difficult to identify. The minimum legal gaming age in all jurisdictions is 21, and the baby-boomer and older demographics are the key targets for casinos, given their higher percentage of discretionary income. This tends to make the game designs more conservative than other mediums. Older adults are also typically less computer-literate than younger people, so games must be kept "comfortable" even for the customers who do not know how to program their VCRs. Furthermore, this design parameter also drives significantly less content than is typical for other forms of entertainment. Gaming devices would never have hundreds of levels or dozens of characters, for example.

CDS's Bandit Bingo game.

Within these constraints, target marketplace segments can be identified in several ways. They can be categorized by the nature of the games (reel-spinning slots, video slots, video poker, and so on), by denomination (a game that accepts nickels is designed and targeted differently from a $1 game, for example), or by the targeted player type. In general, the industry has perceived two categories of customers, the "tourist" and the "local." Games designed for the tourist market have traditionally been flashier, easier to understand and play, and higher-earning for the casino. The local player typically looks for games that have the highest payback percentage (often video poker), ignoring the bells and whistles of the tourist-style games. In the past, locals' games received little attention in areas such as game choreography, interactivity, and graphics and sound quality. However, the aforementioned technological breakthroughs, along with improvements in the game design process, have resulted in new titles that have broad appeal to players of all types, and the tourist/local distinctions are beginning to blur. (They have also resulted in significantly shorter replacement cycles. As with the PC gaming industry, the faster the technology advances, the faster it becomes outdated.)

Designing a Successful Gaming Experience

A successful game design satisfies the following objectives:

INITIAL ATTRACTION. Why will this game be attractive to players when placed on a casino floor with thousands of other

games? Some games lure players simply by offering a large jackpot, while others offer unique themes, recognizable brand names, or just an attractive overall package. These games are typically placed in banks of four to twelve machines and are offered for play 24/7, a rigorous environment to say the least.

PLAYER APPEAL. What are the game characteristics that will keep people playing once they have chosen this game? Again, certain models accomplish this by offering the promise of a huge jackpot, but players are increasingly opting for models that offer a higher level of overall entertainment value.

COMPLETION. What is the goal the player is striving for? This again can be simply a large top award, but the trend is to develop games with enhanced secondary features over and above the standard game play. Examples of these features include bonus games: different game levels which are reached by satisfying a specified objective in the primary game. Bonus games typically feature special animation, distinctive music, and most importantly, the opportunity for additional payouts.

CELEBRATION. What type of feedback does the game provide when the player wins a jackpot or achieved a game milestone? Casino gaming is typically a very social experience, and providing feedback to the player that he or she has accomplished something special is critical to a game's success.

Not surprisingly, these objectives aren't that different from those established for non-gambling games as well.

Also similar are the tools utilized for prototyping and development. After game and math concepts are established and storyboarded, prototypes are built using Macromedia Director or similar proprietary tools. Once a game is approved for production, code is typically developed in C or C++ (usually with Visual C++), although code destined to execute from EPROM may be written with legacy or custom tools. Artwork is developed using all of the usual suspects: 3D Studio Max, Lightwave, After Effects, and of course Photoshop. Custom graphics tools may be used to dither images to lower color depths or convert to proprietary file formats.

In an ideal world, the workflow of game development follows closely the procedures detailed in the article "Bringing Engineering Discipline to Game Development" (December 1998). But as we're all aware, factors such as market conditions, personnel changes, and simple deadlines can compress development cycles to something less than the theoretical ideal. Still, it's critical to respect the conventional alpha/beta/silver/gold release process in order to refine game attributes, squash bugs, and prepare the title for regulatory submission. Product development cycles can vary significantly; market opportunities have driven games from storyboard to shipping dock in as little as six months. More typical development cycles run 12 to 18 months.

Several "banks" of CDS Easy Street slot machines with accompanying signage.

The distribution channels for gaming devices can be particularly brutal. Unless a title is extremely strong, most casinos demand a trial period before they will commit to a purchase. During this period, which is typically a minimum of 30 days, casinos get to keep and operate the games at no cost to them. Once concluded, they then make the choice to keep or return the game. Often a buyer will request that the machine be converted to another title if they feel the installed one is not earning enough; these conversions are also typically done at no cost to the buyer. Then, should the casino decide to move ahead with the purchase, often a substantial discount is requested (the logic being that they are buying a used game at this point). Most game manufacturers distribute their games directly, although third-party distributors are often used for international markets.

Money for Nothing

Because these are gambling games, the game math is a critical factor in the success or failure of a title. This math determines, on a statistical basis, how big and how frequently the jackpots are hit. A game that is perceived by players as offering neither high payouts nor any other type of entertainment value will quickly be shunned by the marketplace. The basic math considerations are a game's hit frequency (the statistical percentage of plays in which some type of payout is awarded), as well as the overall payback percentage (the percentage of money wagered by players that is returned to them via payouts). Most jurisdictions have established laws that require gaming devices to offer a minimum payback percentage of 75 percent, and it may be surprising to some that the great majority of modern games return at least 90 percent of the total monies wagered.

With some game types, the overall payback percentage is dependent to some extent on player skill. Games such as video poker and video black-

jack, for example, require the player to make certain decisions during each hand of play (such as which cards to hold). The skill and strategies used can swing the payback percentage several points in either direction. This is an important factor in the design of video poker games in particular (generally regarded as the most popular "locals" game), as some models can offer up to (and sometimes even more than) 100 percent payback with the right strategy.

At the heart of the math engine (and indeed, the game itself), lies the random number generator, or RNG. A far cry from the simple random functions provided by most development environments, a gaming device's RNG consists of sophisticated algorithms designed to ensure that all outcomes are as random as is technologically possible. The best RNGs use algorithms that change the random seed constantly and unpredictably, as the ability to detect predictive patterns in a game is a classic cheating technique.

When a game is initiated, output from the RNG is funneled through the odds calculations. At this point, the game outcome is calculated and displayed (either by spinning reels or a video screen), and evaluated for potential winning combinations. Payouts are based on the amount wagered and the paytable, the schedule of awards for any given game outcome. Naturally, the longer the odds of hitting a certain combination, the higher the payout will be. It's important that the paytable be displayed and communicated clearly to the player, otherwise disputes are likely to result. Traditionally, paytables were printed on the backlit glass that is part of almost all game cabinets, however the increased popularity of video-based games has allowed developers to display more informative and interactive paytables on the video monitors. These paytables, along with general game help, are usually accessed through touchscreen buttons.

CDS's Reel Racers video slot machine. The hardware deployed in casino games must be robust enough to withstand 24/7 operation in less than ideal conditions.

More Technology, and the Call for Professionals to Build It

In addition to the math engine, modern gaming devices have several additional subsystems critical to their success. I/O functions are very important, as the games must monitor all internal functions. Examples of these functions include coin and bill handling, button and/or touchscreen status, and integrity checks (door open, for example). Regulations mandate that games constantly monitor these functions and go into "tilt" mode should any abnormalities be detected. Games must be able to survive an unexpected shutdown gracefully with absolutely no loss of data, as very few casinos provide adequate uninterruptible power supply (UPS) systems on their floors. Additionally, a complete game history log must be kept in nonvolatile memory (typically on an EPROM chip) so that casino personnel and regulators can review the game's history should a customer dispute erupt.

Chance from the Easy Street game shown here with a background scene.

Networking capabilities are another important consideration in today's machines. The great majority of casinos today utilize slot monitoring systems, which are in effect large, real-time databases of machine information and player data. All games on the casino floor are networked to this database via proprietary, encrypted protocols. These systems have enabled the development of the popular slot clubs, wherein customers sign up to have their play monitored online. Once enrolled, players earn bonus points based upon their play. These points can then be redeemed for cash or merchandise. Casinos benefit from these systems not only by attracting player loyalty, but also by building a tremendous database of marketing information. They also use these systems to extract a great deal of accounting and security information from the gaming devices, and are required by law in many states.

Games may also offer progressive jackpots, which increase over time and are based on the amount wagered by players trying to win them. These jackpots are typically advertised on large overhead LED or plasma TV signage, and other proprietary, encrypted protocols (typically serial-based) have been developed to enable communication between the games and the displays. Games can be networked together to form a progressive link, where play from all games contributes towards one large jackpot pool. In recent years, these networks have grown to encompass machines spread throughout a state (and in some cases, several states), thereby enabling the posting of lottery-sized jackpots (with accompanying lottery-sized odds, of course).

As a result of the advancing technology and design methods, the gaming device business offers significant job opportunities for game programmers and artists. The skill sets necessary for these professions are virtually identical to those sought after in other types of game development, with the advantage to those who have a background in (or a penchant for) the mathematical areas. Nevada, the home of most major manufacturers, has never been known

First-level bonus screen of CDS's Monkey Business game.

Outtake from the Monkey Business promotional movie.

as a major hotbed of technological talent, so when the need to find staff arose, the industry turned toward (not surprisingly) Silicon Valley. Several major manufacturers (including mine) have established design studios in the Valley, where it's easier to attract world-class talent. Others have found offering relocation to Nevada attractive due to the significantly lower cost of living. Still others prefer simply to outsource their multimedia production to well-known California studios.

Popular Titles

So what types of games are most popular today? Traditional reel-spinning games continue to constitute the greatest percentage of the market (these modern games differ from reel-spinners of yore in that the reels are now controlled by stepper motors driven by the main processor board). However, their dominance is being threatened by the dramatic growth of video-based games, and all indications point to this trend continuing.

Within the video classification, poker games remain popular, mainly because of their high payback rates. Growth potential for these games remains limited, however. The real boom is occurring in the video slot arena, where designers can implement myriad features unavailable on any mechanical-reel machine. The hot ticket today is "secondary bonusing," the addition of different game levels beyond the base reel game. Initially, these were implemented as simple second-level screens that appeared over the slot reels and allowed the player to make several game-show type choices (picking one of several doors, for example). The result of these choices is almost always a bonus payout. Current game choreography expands on this premise.

In CDS's Bandit Bingo, for example, players begin by choosing a bingo card that is placed on-screen below the slot reels. As players spin the reels, bingo ball symbols appear. Should a bingo ball symbol match a number on the card, it bounces off the reels and daubs the bingo card. Once players have bingo, they are taken to a second screen where they choose one of five special bingo balls for a bonus payout. This game offers the opportunity to play two traditional casino games simultaneously (slots and bingo), and also provides for the bonus experience. Easy Street, another CDS title, offers a road-based board game as the initial bonus. Players earn bonus payouts for each successful turn on the board, and should they make it to the end they are taken to an additional bonus level. Here, an animation of the game's main character, Chance the Dog, plays in a window. As Chance drives his car down Easy Street, players use the touchscreen to select any of the buildings he passes for an additional bonus jackpot.

CDS's Easy Street video slot machine (complete cabinet). The game, part of CDS's Bandit model series, utilizes a hybrid ROM/PC architecture to achieve high security while delivering engaging, immersive content.

Recently, the industry has begun to look beyond its own borders for attractive content. This has resulted in a recent surge of games based on licensed brand names from outside the gaming industry. Titles have recently been released that are based on board games (Monopoly), television (*Wheel of Fortune*), and deceased entertainers (Elvis). Although this approach offers instant brand recognition, and has been very successful in several cases, the huge royalties involved make development of such games a risky proposition.

The Long Arm of the Law

Underneath all of the design and technology lies the basic fact that these machines are designed for gambling, with the classic elements of chance, risk, and reward. Gambling

has always been a controversial topic in the United States, and the industry's rapid expansion in the early 1990s provided plenty of fuel for the ongoing debate. In 1997, Congress authorized the National Gaming Impact Study Commission, the purpose of which was to report on the economic, political, and social effects of the gaming industry.

Because of the controversy, as well as the somewhat notorious origins of casino gaming (remember the movie *Casino*?), the industry is subject to intense governmental regulation on all levels. Businesses wishing to enter the industry must go through a vigorous background investigation by every state in which they intend to do business. These investigations, which delve into all aspects of an applicant's background, can cost well over $100,000 per state with no guarantee that a license will eventually be issued. For this reason alone, the cost of entry into the industry is very high. All employees of a gaming company can be subject to personal background checks as well.

The main reel screen from the CDS Monkey Business game.

The second-level bonus round from CDS's Monkey Business game.

Beyond these corporate and personal licensing requirements, the games themselves are also subject to heavy regulation. Both hardware and software are subject to review, and the laboratories that examine these games enforce strict rules on the randomness, payback percentage, security, and auditability of all games. Certain states (Nevada, New Jersey, Mississippi and Michigan) have established their own testing laboratories, while other states prefer to rely on the services of independent testing facilities such as Gaming Laborites International. Hardware components are subject to environmental testing (electrostatic discharge resistance, line noise, and so on), and all software is reviewed at the source level. Approved programs are identified with a digital signature, and once approved, even a one-bit change to the code will render it noncompliant. The time required for these approvals varies wildly from jurisdiction to jurisdiction, but is typically never less than 30 days. Obviously, this added compliance time can have a major impact on shipping schedules.

The state of Nevada has recently added a new twist to the regulatory process by proposing rules that address game content itself. Specifically, these new regulations prohibit advertising on any gaming device, and further ban the use of any themes or artwork that may appeal to children. Obviously, the subjective nature of these proposed new regulations troubles many in the industry, and has already caused the withdrawal from consideration a game based on Comedy Central's *South Park* animated TV series.

No discussion of electronic gaming would be complete without touching on the rise of Internet gambling. Supporters of online wagering tout the convenience of simply logging on and betting. Despite this, true online betting carries enormous risks and has been shunned by the vast majority of the legitimate gaming industry. The primary reason is that the lack of any sort of comprehensive regulatory structure leaves Internet gambling ripe for fraud, abuse, and deception. There are no guarantees that any of these games are fair, or that players will actually be paid should they win and decide to collect. Not surprisingly, many of these sites are hosted from islands in the Caribbean, away from U.S. legal protections. Additionally, there is no easy way to guarantee that people gambling on the Internet are of legal age or capacity. As a result, Congress is currently considering a bill that would outlaw Internet gambling, and similar laws have already been enacted by several states. Until these problems can be resolved (if ever), Internet gambling will remain in that online netherworld currently inhabited by pornography and warez web sites.

This is not to say the Internet has been totally ignored by the legitimate gaming industry, however. Forward-thinking companies are developing marketing and other related programs which leverage the power of the medium without actually offering online wagering. For example, the previously mentioned slot monitoring databases have already begun to grow online hooks, and the prospects for similar systems are bright.

The Challenge of the Future

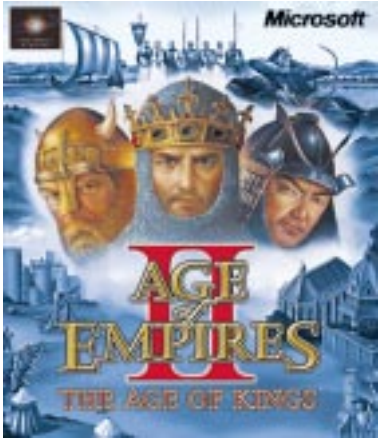
As the gaming device industry enters the 21st century, it faces many of the same challenges and opportunities present in the PC, console, and coin-op gaming worlds. Industry consolidation will continue, with the large, well-funded companies absorbing the smaller players. Advances in the technology employed will continue to shorten the life span of these games, although probably never to the extent that the PC and console markets must deal with. And fierce competition will continue to force the development of quality, interactive content, while never forgetting to offer players a good gamble. That's a safe bet. ■

POSTMORTEM

Ensemble Studios' Age of Empires II: The Age of Kings

by Matt Pritchard

For anyone who has ever worked on a PC game and poured their heart and soul into their work, they may have imagined in an optimistic moment, "If this game sells a million copies..." Maybe it was spoken out loud, or carefully whispered so that no one else would hear. It's the expression of the dreams and promise of success that drives so many of us. But recently I found myself somewhere I never anticipated as I listened to this ironic ending to that very statement: "... I am going to be so disappointed if that's all it sells." And you know what? I had to agree.



Matt Pritchard is busy trying to be a modern renaissance man. When not working, he can be found with his family or playing with his collection of antique video games and computers. He can be reached at mpritchard@ensemblestudios.com.

Catching Up

Two years ago in this very column (Postmortem, March 1998) I told you the story of Ensemble Studios, a scrappy upstart that overcame challenges to create the game AGE OF EMPIRES (AOE). Since its release two years ago in the great real-time strategy (RTS) wars of 1997, approximately three million copies of AOE have been sold worldwide, along with almost a million copes of the RISE OF ROME (ROR) expansion pack. The totals don't give the whole story, though. AOE proved to be a consistent seller, hanging around the top of the PC Data charts, and even re-entered the top ten a year-and-a-half after its release. The demographics of the buyers were another surprise. Sure we had the sales to the 14- to 28-year-old male hard-core players, but we also had significant sales to older players, women of all ages, and casual game players of all sorts. That is to say we had a crossover hit on our hands. If you have ever watched the VH1 show *Behind the Music,* then you know the story of the upstart band that finds itself suddenly on top of the world — things change, and not always for the better. I wouldn't go so far as to say that we sank into a wild orgy of sex, fast cars, and money — despite the wishes of a couple of our guys — but this change along with the benefits of success brought us a whole new set of challenges, making our next game no easier than the first.

Designing a Sequel

It was a surprise to no one that Ensemble Studios' next game would be a sequel to AOE, although most people probably didn't know that we had a contract with our publisher for a sequel long before the original game was finished. Given our historically-based themes and time periods in AOE, the chosen time period for AGE OF EMPIRES II: THE AGE OF KINGS (AOK), the Middle Ages, practically picked itself. That was the only easy part, however. Like a band going back into the studio after a hit record, there were differing opinions of what direction to take next. Do we play it safe and stick tightly to the AOE formula, or do we get bold and daring and take the whole game genre in new directions? This is the million-dollar question every successful game is faced with when the topic of a follow-up is raised. But the successful band I'm using as an analogy is fortunate. They don't have to contend with the unbelievably rapid pace of evolution in PC hardware and games.

Improvements to the game in every area from graphics to user interface are expected in this business as a matter of fact. Expectations can be a bitch sometimes. Take the vast demographics of AOE players that I mentioned earlier — they are the largest group of people most likely to buy the sequel — and everyone is concerned about making sure that this huge and diverse group will like the next game so much they will run out and buy it. We'll just do more of what we did right in AOE, we said. That sounds great, but it's almost impossible to quantify in a meaningful, detailed way. The game business is brutal to those who fail to move forward with the times, but it's also equally brutal to those who experiment too much and stray from the expectations of the players.

TOP ROW, FROM LEFT TO RIGHT: *Jeff Goodsill (COO), Brad Crow (art lead), Brian Hehmann (artist), Angelo Laudon (lead programmer), Sandy Petersen (designer), Dave Pottinger (programmer), Ian Fisher (designer), Harter Ryan (producer), Duncan McKissick (artist), Trey Taylor (programmer), Mario Grimani (programmer), Paul Bettner (programmer), Chris Van Doren (artist), Jeff Dotson (artist), John Evanson (programmer), Doug Brucks (programmer), Roy Rabey (IS support), Paul Slusser (artist), Chea O'Neill (artist), Bob Wallace (strategic), Mike McCart (webmaster).*

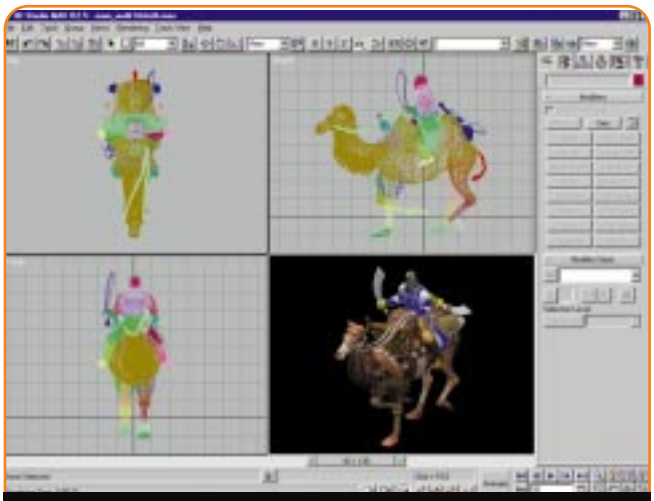
BOTTOM ROW: *Rob Fermier (programmer), Nellie Sherman (logistics), Stephen Rippy (music), Herb Marselas (programmer), Mark Terrano (lead designer), Chris Rippy (sound), Herb Ellwood (artist), Thonny Namounglo (artist), Duane Santos (artist), David Lewis (programmer), Sean Wolf (artist), Bruce Shelley (lead designer), Matt Pritchard (programmer), Brian Moon (CFO), Tony Goodman (CEO), Don Gagen (artist), Greg Street (designer). Not pictured: Tim Deen (programmer) Brian Sullivan (strategic), Chad Walker (artist), Eric Walker (artist), Scott Winsett (lead artist).*

When we started work on AOK, we thought that we could make use of our existing code and tools, and that this would make the sequel easier to create than the original. Filled with these optimistic thoughts, we concluded that we could develop AOK in a single year. This was also going to be our opportunity to add all those dream features and make our magnum opus of computer games. So we set about to do just that. To make enhancements for AOK, we had pulled together a giant wish list of features and ideas from inside and outside sources. To the game design we added all sorts of neat new features such as off-map trade, renewable resources, combat facings, sophisticated diplomacy and systems of religion, and so on. Of course, the art, sound, and game content were also going to be bigger and better and bolder and brighter and...well...you get the idea.

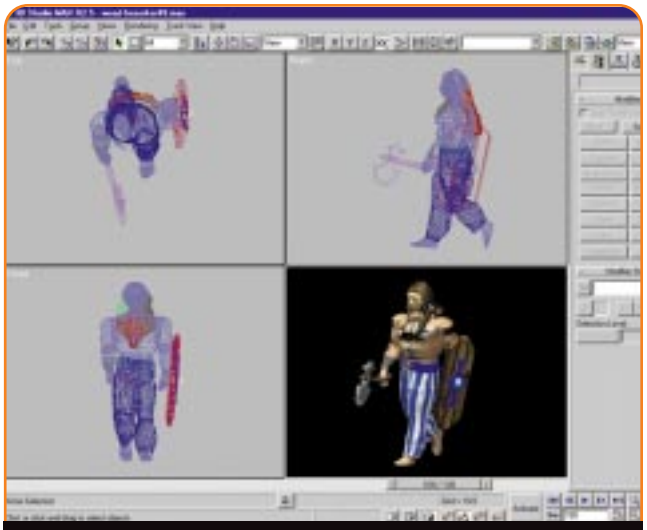
Several months down the road, reality reared its ugly head in big way: we had bitten off more than we could

AGE OF EMPIRES II: THE AGE OF KINGS

Ensemble Studios
Dallas, Tex.
(214) 378-6868
http://www.ensemblestudios.com
Release date: October 1999
Intended platform: Windows 95/98/NT
Project length: 24 months
Team size: 40
Critical development hardware: Pentium II 450 128MB, Dual Xeon 450 512MB
Critical development software: Visual C++, 3D Studio Max



A 3D Studio Max model of the Mameluke character. Twenty thousand polygons got reduced down to several hundred pixels for the final game.



A 3D Studio Max model for a male villager. For AoK, female villagers were also added.

chew and the game's design was losing focus. Instead of sticking to the core of what makes an RTS game great, we had gone off in many contradictory directions. Along with that came the realization that there was no way that we were going to finish AoK in a single year and have it anywhere close to the quality of AoE. This was a sobering time for Ensemble Studios staff and our publisher, Microsoft. While the Ensemble Studios crew adjusted quickly, it caused a few problems for some of the people at Microsoft: "Uh, guys, we've already gone ahead and committed to our bosses that we would have another Age of Empires game this year," is probably a good way to paraphrase it. From this situation, a contingency plan was born. We were going to take another year to finish AoK, giving us time to get the game back on track and to create the ambitious content for it. We also had a plan to help our publisher out: we would create an in-house expansion pack for AoE. It would be a significant addition to the game, yet require only a small amount of our resources, and most importantly, it would be ready in time for Christmas 1998, taking the slot originally planned for AoK. Thus was born the RoR expansion pack. RoR helped, but it didn't take all the pressure off us. Unlike the latitude we had with AoE, which had also come out a year late, our new deadlines for AoK were very firm and hung over us the entire time. The pressure was very much on.

What Went Right

We did what it took to make AoK a triple-A game. While the decisions to take an extra year and reset the units to an AoE baseline were tough in the short term, they were the right decisions to make. The commitment of Ensemble Studios to exceed the quality of its prior games never wavered. To realize our goals, we added the additional programmers, artists, and designers that we needed. When we needed to stop, take a hard assessment of what we were doing, and kill our own children if need be, we did just that. We pushed ourselves hard and we came together as a team.

1. Addressed the major criticisms of the first game. Despite AoE's success and generally glowing reviews, there were two things about the game that were repeatedly criticized: the artificial intelligence of the computer players and the pathfinding and movement of units. And to be honest, they were right. Because these issues got so much press, we knew going in that if we didn't address them in a visible and obvious way, AoK was going to be raked over the coals by reviewers and users. It didn't matter that other popular RTS games had pathfinding that was just as bad, or that our AIs didn't cheat and theirs did — we weren't going be judged against them, but rather against ourselves.

A bird's-eye view of the AoK game world. Compared to AoE, AoK has bigger worlds with more objects and richer graphics.

To handle the computer-player AI, we

hired Mario Grimani, an industry veteran with significant AI experience under his belt. The computer-player AI from AoE was thrown out, and a new, expert-system, script-based AI was developed. While Grimani was doing the coding, Sandy Petersen led the design team in developing scripts for the new AI. Input from the whole company was encouraged, and various people contributed scripts that were pitted against each other in an evolutionary fashion to develop a computer player that could race a human player up through the ages and react to his tactics.

For the pathfinding problems, nothing less than an all-out blitz was ordered up. The game engine's movement system was redesigned and no fewer than three separate pathfinding and two obstruction systems were developed, requiring five different people working on them at various times. A high-level pathfinder computes general routes across the world map, ignoring such trivial things as people walking, which were handled by lower-level pathfinders that could thread a path through a closely packed group of units. In the end, we were so successful in ridding the movement problems that hampered AoE that reviewers and players couldn't help but take notice and acknowledge the improvement.

2. We innovated within the genre. While in the end AoK stayed much closer to its AoE roots than we had initially envisioned, we pushed the RTS gaming experience forward with a host of improvements. Some of these were interface-only improvements, such as the "Find Next Idle Villager" command, completely customizable hot-keys, and the extensive rollover help. Other improvements changed the game play itself, such as the Town Bell (ring it and all your villagers run inside the Town Center to defend it), in-game technology tree, and of course, Automatic Formations. One of the most praised features, Automatic Formations, caused a group of selected units to automatically arrange themselves logically by putting the strongest units up front and the ones needing protection in the rear. They stay in formation while traveling, replacing the "random horde" that players had become accustomed to in RTS games. Programmer Dave Pottinger originally set out to create a formation system incorporating characteristics of a turn-based war game's formation system, but as the game progressed and our understanding and vision for the game matured, a complicated formation system gave way to a simpler system that better served the game.

A scene from one of the game scenarios. The updated graphics engine and building scale allowed us to create scenes much more impressive than in AoE.

When I wrote the graphics engine for AoE, I used a 166MHz Pentium at work and tested on my 486 at home. A 2MB video card was my target, but the game would run with only 1MB of video memory. Today I have a 32MB TNT2 card in my 500MHz Pentium III system. These changes in the typical game player's system are mirrored by the increase in player expectations for a great visual experience. In AoK, I'm proud to say, we met and exceeded game players expectations. The first thing that you notice upon playing AoK is the scale. The units in the game are about the same size, but the buildings and trees are no longer iconic. They are large structures with a scale that looks as if the units could comfortably reside inside them. Castles and Wonders are now gigantic, imposing structures that fill the screen. And the art itself is just so much better. Our entire art staff gained a great deal of experience and skill with AoE and RoR, and AoK became a showcase for their improved talent. It wasn't just the units and buildings, though. In AoE, the terrain had something of an Astroturf feel to it and the need to make transition tiles by hand limited the game to four terrain textures. For AoK, a whole new terrain system was developed, allowing us to mix terrains together, shade elevation in 3D, greatly increase the number of textures, and even alpha-blend textures such as water. The highest compliments came at the 1999 E3 show when we were unable to convince people from some of our biggest competitors that AoK was still a 256-color game.

3. Better use of bug tracking software and crunch-time management. During the development of AoE, we had a single machine in the office that would connect up to RAID, a remote bug database in Redmond, Wash., via an ISDN modem. This was used to handle bugs found by testers at Microsoft. Every so often someone would fire the connection up and, if the machine at the other end was in a good mood, make hard copies of new bug reports to pass around to people. We also had a different software package for communicating bugs and issues among ourselves, but there were not enough users

on the license for everyone. Suffice it to say, this system left something to be desired, but it was all we had. During the development of AoK, ThinRAID was made available to us, allowing everyone to access the bug database directly from their web browser. Having only one system on everyone's desktop, available whenever needed, that was always up-to-date made a huge improvement in our ability to track bugs, stay on top of things, avoid redundancies, and just plain save time.

The last six months of development on AoE were pretty much one continuous blur of people working nonstop. This took a heavy toll on people, sometimes even straining their health or marriages. As a company, we vowed to not let things get that bad again. To further underscore the need, the composition of Ensemble Studios had shifted dramatically away from being mostly young, single men (with presumably no life) to being dominated by married men with a growing number of children and babies on the way. To protect ourselves, we scheduled crunch time well in advance at multiple points in the development process. The hours were 10 A.M. to midnight, Monday through Friday, with Wednesday nights ending at 7 P.M. so we could go home to our families. We had weekends off and meals were provided during the week. For the most part this worked very well, although having a "family night" where family members could join us for dinner once a week proved to be more of a distraction than we would have liked. Producer Harter Ryan deserves much credit for making crunch time so much easier on AoK.

A shot from a very early version of the game. Most everything shown would be revised before the game shipped.

4. BETTER USE OF TOOLS AND AUTOMATED TESTING. After AoE was finished, we developed several in-house and in-game tools to make the job of development easier. The most-used tool was ArtDesk, a multi-purpose program that converted graphics from standard formats to our proprietary formats, which allowed us to view and analyze the content of our graphics data, and generated many of the custom data files for the game. This easy-to-use GUI-based program replaced several antiquated DOS command-line utilities and automated many tasks, saving a huge amount of time over the development span. In an effort led by Herb Marselas, programming tools such as Lint, BoundsChecker, and TrueTime were used to a degree never approached during AoE's development and proved invaluable in improving the quality of our code. Finally, in-game utilities such the Unit Combat Comparison simulator allowed the designers to balance the game in a more scientific way. Every effort made in the tools area was rewarded with either time saved or significant improvements in the product. The only glaring omission in all this was the lack of an art asset management tool.

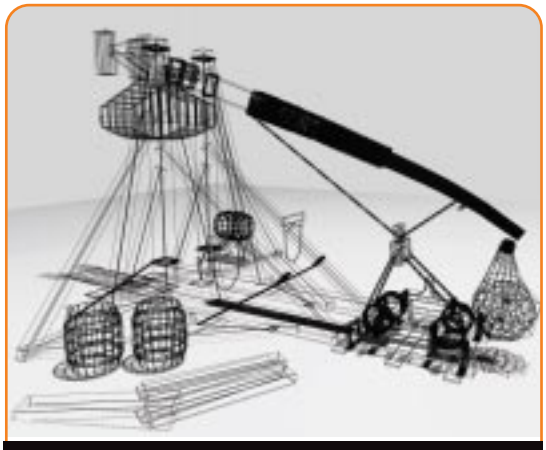
A Turkish mosque shows off the greater detail and improved skills of our art staff.

5. WE MET OUR SYSTEM REQUIREMENTS. A game that's expected to sell in the millions needs to be able to run on most of the computers it will encounter. Requiring cutting-edge systems or specific video cards won't work. With AoK being an 8-bit 2D game, meeting video card requirements wasn't going to be very difficult. But memory and processor-speed targets were another story. All the new systems in AoK would put their demands on the computer. Optimization issues were worked on hard for the last several months of development. The eleventh-hour addition of some clever tricks and a variable graphics-detail switch allowed us to hit our CPU target of a Pentium 166MHz, MMX-supported CPU. The minimum memory requirement of 32MB was also met, but with some reservations. Large multiplayer games on huge maps would need an extra eight or 16MB to be really playable. All in all though, the minimum system requirements for AoK are some of the lowest for games released in the Christmas 1999 season, widening the game's potential audience.

What Went Wrong

I'd like to say that we had fewer problems developing AoK than we did for AoE, but it didn't turn out that way. Some problems listed in the AoE Postmortem were addressed in AoK and others weren't. And like that band going back into the studio to record a follow up to a hit album, we encoun-

tered a whole slew of brand new problems, many of which we found we were just as unprepared for as we were the first time around. I've tried to include some of the issues that became more important due to the fact that we were making a sequel to a successful game.



Wireframe model for the trebuchet unit.

Fully textured and skinned, the trebuchet proved to be one of the most popular units in the game.

1. We still don't have a patch process. This was a problem area from the AoE Postmortem, and as of this writing it still has not been addressed. I outlined the reasons we needed a process to issue patches for our game in a timely manner in the AoE Postmortem. Additionally, a new reason reared its ugly head: cheating in multiplayer games. At first people found bugs in AoE and exploited them to win unfairly. Then it got even worse. Programs called "trainers" were developed that would actually modify the game's code while it was running to allow players to cheat.

Being the developer — not the publisher — of AoE, we don't have the final decision if or when a patch is to be released. As a result, all during 1999 our reputation as developers was assaulted by fans who saw us as uncaring about the problems that were driving people away from online play of our games. The topic of cheating in multiplayer games is so extensive I hope to do an article on it in the near future. We addressed this problem with our publisher and were promised a patch process. Unfortunately, AoK shipped with a couple bugs that seriously needed addressing in the short term. They're not show stoppers, but if not addressed soon, the game's (and our) reputation may suffer another black eye. If a patch for AoK is out by the time you read this, then you can conclude that we finally established our process.

2. Unfinished versions of the game got out. This is a problem that is born of success. Prerelease versions of nearly all games wind up circulating in pirate channels known as "warez." This happened with AoE. Imagine our surprise at reading an entire review of the game (an alpha version) eight months before it was released. Fortunately, almost no one bothered with it until the game was properly released because nobody knew much about it. AoK was a completely different story. It was a highly anticipated sequel to a very successful game, and the various warez sites were tripping all over themselves to get a copy of the latest build. And get a copy they did. They were usually only one or two weeks behind our latest build. It seemed as though copies were leaking out from every imaginable source — play-testers at Microsoft, previews sent to magazines, even internal sources. Unfortunately, positively identifying and fingering the culprits was almost impossible. There were hacking attempts on our FTP server and network, though the real rub came from the pirates in Hong Kong and Singapore. They took the warez versions of AoK, burned them onto CDs,

added some cover art, and sold the game throughout the Pacific Rim. In Korea, the CD vendors operating in front of Microsoft's headquarters had a warez version of AoK for sale. Warez versions were even turning up on eBay. Though we doubt bottom-line sales were hurt much, our pride certainly suffered. Any of our future games will probably require connecting to a secure server of our own design to operate, even for single-player games.

3. Play-testing had a lot of problems. This item is a catchall for several problems that we encountered. On the good news front, our new offices have a dedicated play-test area equipped with identically configured machines. The bad news is that we didn't make the best of it.

Many of our play-tests were not organized and focused enough, seriously reducing the amount of new and meaningful feedback obtained. It wasn't always clear when we were testing for specific bugs and issues, and when we were testing for "fun." We had a schedule of participants which drew upon the whole company, but schedule conflicts and lax enforcement resulted in the same people playing most of the games. We played too much multiplayer and not enough attention was given to the single-player game. And some people took it much too seriously, trash-talking other players, celebrating wins at the loser's expense and storming off when they were losing. Play-test problems weren't confined to Ensemble, though. At Microsoft, it was discovered that a play-tester had turned cheats on, playing to win not to test, in almost every game for over a month, which invalidated all the feedback from that group for the prior two months.

4. Art asset management was nonexistent. The number of individual frames of graphics in AoK is in the tens of thousands, and we didn't do a good job managing it. The programmers had a source-control system to help coordinate their primary output of code and the designers had the game's database system, but no such equivalent existed for the game's art assets. Artists could be working on something with no idea that anyone else was also working on it. There was no way to get a momentary snapshot of who was working on what, other than going around from office to office. Plus, there was no way to tell which files were actually live and being used and which ones were just taking up space. Also missing was a way to go back and find prior versions of art, or to guarantee that new versions wouldn't be overwritten. As we have grown as a company, this problem has grown even faster. To address this problem in the future, a source-control similar to the art asset management system is being developed for use in all future projects.

5. Problems with third-party APIs and software. Another one of the items from the AoE postmortem returns again. Microsoft's DirectPlay API still has a number of issues that make it less than perfect. One of its biggest problems is documentation and testing of the lesser-used portions of the API, or rather the lack thereof. Late in the development of AoK, our communications programmer, Paul Bettner,

was able to communicate with the DirectPlay developers and an interesting scenario played out several times: Paul would attempt to solve some problem and the developers would indicate that it wouldn't work because of bugs in DirectPlay that they knew about but that were not documented.

DirectPlay wasn't the only problem. We decided to use DirectShow to handle our cinematics. The short version of this story is that it just didn't work. And then there was the Zone software for Microsoft's online Gaming Zone. The Zone software was developed too late in the process and had a number of problems, due to a lack of time to test and correct. Unfortunately, this means that direct TCP/IP games are more reliable than those played over the Zone, which is disappointing. This was not all the Zone's fault because we did not get our requirements to them soon enough.

The Show Goes On

One of the touchiest and most personal issues concerned letting success go to our heads. The success of AoE is something that a lot of people in this business have not experienced. It exceeded our wildest dreams and allowed our company to take charge of our destiny. I remember when we got our first AoE royalty check — I had never held a multi-million dollar check before. That was great. We all got caught up in how good we were doing. Over time an attitude of invincibility set in. With a success like AoE, it's easy to forget what it was like to wonder if we were going to be in business the next year. At some of the industry events such as the Game Developers Conference and E3, some of our people behaved in ways that embarrassed us. With success comes a responsibility to behave appropriately — the game industry is a small and incestuous one, and nothing lasts forever. Behaving in an exemplary manner and being friends with the industry at large is far more important than chest-beating about our current success. Suffice it to say that people in the Ensemble Studios organization have stepped forward to address this and we have challenged ourselves to be better people.

All the early indications for AoK are that it's going to be a blockbuster on the order of its predecessor, and maybe even greater. The reviews from the press have been unbelievably positive. According to PC Data, AoK was the number-one selling game in October.

The great success of AoE made it possible for us to go to the next level of making great games. Though it enabled us to grow and acquire greater resources, it also raised expectations for our next game and spawned a host of new challenges. Meeting these new expectations has proved to be just as tough and rewarding a journey as creating the first game. In the end we succeeded in creating a game to be proud of, and I feel privileged to have been part of it. ■

by Brian Hook

Ship Your Damn Game!

Some people liken game development to other art forms such as painting, cinema, and music. While there is a certain truth to this comparison, sometimes "visionaries" confuse the pursuit of artistic vision with slacking off while playing QUAKE on the company LAN. Making a successful game without losing your sanity and/or having the entire team quit at project end requires a minimum level of discipline, engineering talent, perspective, and vision. Hey, it's great to have a vision to pursue, but remember that unlike writing a song, developing a game is a slightly more involved process than, say, breaking up with your girlfriend, drinking a few shots of tequila, and riffing on your acoustic guitar until sunrise in your dorm room. That MUD you maintained your sophomore year doesn't count.

My message is simple: ship your damn game.

Look, you're a game developer, not the bassist for Korn. This means that you're actually expected to produce some meaningful work that is profitable for your investors (those people who give you money based on the expectation that they will eventually get it back, possibly with interest). Sure, have your "vision," but understand that you're first and foremost a business, not some Real Cool Guy that just happens to develop computer games on the side when not busting ass trying to win the Geek with the Most Piercings Award at the Game Developers Conference. Ship your damn game before you start asking for cover shots on *Rolling Stone*.

Shipping a game really isn't that difficult if you think about it — just pretend you're an actual professional interested in creating a profitable product (a.k.a. "the title") for your investor (a.k.a. "The Man"). This means, above all else, taking the development of your title seriously, which entails fun stuff like actually doing your job (note: your job is not to surf the web, lamp around on #IM2COOL on IRC, or do web interviews with gaming "news" sites). Ship your damn game before taking an on-the-job vacation.

I'm not advocating that you go corporate and buy a three-piece suit and a Lexus, but I am advocating that you, as a developer, take your art seriously both as a business and as a form of creative expression. This means handling your company and team with maturity and staying focused on your project. When you get that wad o' cash from Mr. Publisher, spend it wisely, watch your finances, and understand that you haven't "made it" just yet. And when Mr. Publisher comes knocking asking where your milestone is, don't start railing about The Man holding you down. As much as we love to hate publishers, the vast majority of missed milestones can be directly attributed to the developer's own mistakes: a combination of naively optimistic scheduling, an overconfident assessment of a team's ability to finish everything "if we bust ass this weekend," and a few too many hours refining one's railgun tactics on Q3TEST2.

I'm tired of watching a bunch of self-absorbed dorks spout about their intrinsic coolness and how they're gonna make a [insert category champ] Killer. Back up your rhetoric, magazine covers, screenshots, interviews, trash talk, and .AVIs with a real game

Brian Hook is a programmer at Verant Interactive. Before working for Verant he was an employee at id Software where he worked on the popular QUAKE 2 and QUAKE 3 titles. Before that Brian was an engineer at 3Dfx where he designed Glide, their hardware access API. In addition, Brian has also worked as a consultant with Nvidia and Silicon Graphics. Brian was formerly a columnist for Game Developer *magazine and writes his own column, Ask Hook, at http://www.voodooextreme.com.*

— something that I can zip down to CompUSA and purchase. Until then, shut your pie hole and ship your damn game.

Finally, just because you've got sharp objects stuck in uncomfortable places, colored hair, and really loud clothing doesn't mean you're any less a nerd than the rest of us. It just means you're more annoying about it. Hey, you know that fat guy with the greasy, dandruff-laced hair and the Palm Pilot busting out the pocket seams of his sweat-stained button-down? He's a nerd. He knows it. He's just not trying to be the loudest nerd around. Handle your geekdom with quiet dignity. Oh, and in case I forgot to mention this, ship your damn game before you start copping a "Look, I'm a cool guy, I just happen to write games...but you guys are hopeless geeks" attitude at the GDC.

I'm not saying that getting some tattoos, piercing your anatomy, and wearing really cool anti-establishment T-shirts will necessarily prevent you from getting rich and famous. But delivering a quality product on time and on budget gives you a much greater chance in the long run of being able to present your vision repeatedly to the world. In the end, most visionaries would rather have their creation shared with the world than appreciated in solitude. The key to this is to take your art seriously as a business while at the same time deferring your delusions of being the next game developer-cum-rock star until after you've actually done something to warrant it. Ship your damn game, because screenshots don't count. ■