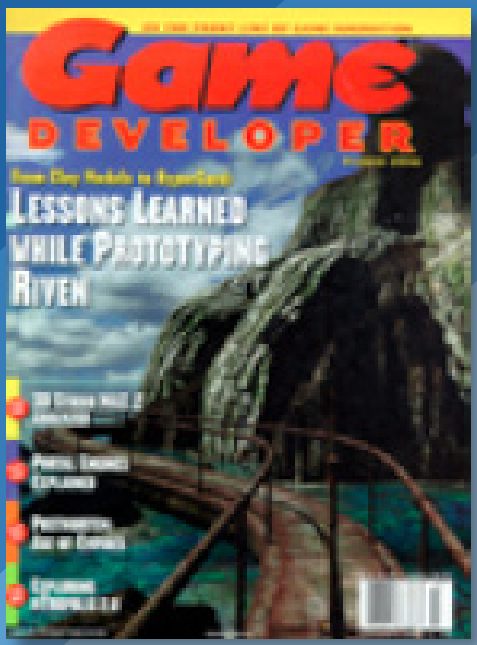
gd

GAME DEVELOPER MAGAZINE

MARCH 1998



UBM
Tech

Making Sense of Fahrenheit

Last December, the wraps came off a plan by Microsoft and SGI to integrate their graphics strategies into one unified vision, code-named "Fahrenheit." Unfortunately, many of us feel as if not all of the wrapping paper was removed from this early Christmas present: many of Fahrenheit's implications remain unclear.

Here is some of what's known thus far. Three new APIs will be developed by Microsoft and SGI, with input from Intel, Hewlett-Packard, and some other IHVs and ISVs along the way. These three APIs are:

1. A Low-Level API for accessing low-level hardware. This will supercede the current Direct3D and OpenGL APIs, yet will retain elements of both (with a dash of DirectDraw thrown in). This API will definitely be COM-based on the Windows platform, and Microsoft has said it will be an evolutionary step forward for current Direct3D users, just as Direct3D has evolved in various ways over the years. In fact, it might be helpful just to think of it as a future version Direct3D, since I bet that's what it will be under Windows. Approximate arrival date: 2000.

2. A Scene Graph API, consisting of a framework that sits on top of the low-level API and manages the geometry and data making up a scene. By adding a layer of abstraction away from the hardware, this API lets developers concentrate on application content rather than coding to the underlying system. According to SGI, the Scene Graph API will incorporate technologies from the company's Performer, Open Inventor, and Cosmo tools, as well as recent OpenGL advancements. Approximate arrival date: 1999.

3. The Large Model Visualization API. LMV is based on Hewlett-Packard's DirectModel and SGI's OpenGL Optimizer technologies. It's primarily for industrial-strength prototyping applications involving millions of polygons and very complex data sets — think automobile engineering analysis.

Fahrenheit documentation states that "developers can feel confident that there will be a high degree of functional consistency between their current OpenGL applications and the Fahrenheit Low-Level API." Yet Microsoft recommends going forward with DirectDraw and Direct3D from here on out, as the migration path from future OpenGL iterations (that is, after version 1.2) to Fahrenheit under Windows won't be as easy. This has called into question the graphics strategies of game developers for the coming years.

The result of this confusion? A lot of conjecture by game developers as to what the Fahrenheit announcement means and what their best course of action is. Some developers view it as SGI capitulating to Microsoft, and lament the passing of one of the last bastions of excellent technology not fully co-opted by the powers-that-be in Redmond. However, let's all take a deep breath and resist a knee-jerk reaction. Rather than fanning the flames on the Direct3D vs. OpenGL spat, let's put our energy into constructive dialog with these companies as they try to create a solid foundation for a unified graphics standard.

You can expect thorough coverage and analysis of Fahrenheit in *Game Developer* in the coming months. Meanwhile, write us at gdmag@mfi.com with your questions and reactions. We'll publish your letters in an upcoming issue.

Hook on Hiatus, Lander Joins Us

This month, we're joined by a new Graphic Content columnist, Jeff Lander. Jeff is taking over for Brian Hook, who imparted much of his considerable knowledge about graphics programming to us in 1997. Unfortunately for us, Brian got involved in a little side project called id Software along the way, and his column-writing time began to dwindle. We'll miss him, but he's going to stick around as a member of our advisory board. Fortunately, Jeff Lander is also a phenomenal developer as well as an excellent writer, and we're fortunate to land him. Welcome, Jeff. ■

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

EDITOR IN CHIEF Alex Dunne
adunne@compuserve.com
MANAGING EDITOR Tor Berg
tdberg@sirius.com
EDITORIAL ASSISTANT Wesley Hall
whall@mfi.com
EDITOR-AT-LARGE Chris Hecker
checker@bix.com
CONTRIBUTING EDITORS Jeff Lander
jeffl@darwin3d.com
Josh White
josh@vectorg.com
ADVISORY BOARD Hal Barwood
Noah Falstein
Brian Hook
Susan Lee-Merrow
Mark Miller
Josh White
COVER IMAGE Cyan Inc.

VICE PRESIDENT KoAnn Vikören
PUBLISHER Cynthia A. Blair
(415) 905-2210
cblair@mfi.com

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

un Miller Freeman
A United News & Media publication

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

www.gdmag.com

Third-Party Libraries

In response to Brian Hook's column in the January 1998 issue of *Game Developer* ("Graphic Content," p.11), in which he discusses perceived problems with using prepackaged software libraries — his assertions reflect a common attitude among some of the younger Internet crowd who claim to "know" all about software libraries that they've never tried.

Brian's conclusions are based upon false assumptions — that external libraries are buggy, require workarounds that take longer to implement than writing the code from scratch, are not optimized, are inflexible, are poorly supported, and are incapable of adding functionality except at the whim of the original author. He presents these "facts" with no evidence, in the spirit of "everybody knows it to be true."

I am the senior programmer at Ted Gruber Software, and I would like to address some of the myths surrounding software libraries.

We have been developing the Fastgraph libraries for more than 10 years. Our libraries are highly optimized in hand-written assembly language. They support most compilers and virtually all video cards. They are copiously documented and are essentially bug-free. The rare customer complaints we receive are given priority treatment, and if a bug is actually found, it is usually fixed the same day.

Brian asserts that if the library author isn't amenable to adding the particular feature that you want, then you are "in trouble." This is simply baffling. Fastgraph is modular, like a set of building blocks. If you need a block that's not in Fastgraph's block set, you simply add another block. You don't need to throw away everything you've built.

His claim that "The software component that you purchased might look great on paper, but when you drop it into your program and then spend a week looking for a bug that turns out to be a part of your new magic software IC, well, you tend to snap out of your dream world pretty quickly" is simply offensive. Not one of our customers has ever reported experiencing anything like that. This kind of polemic may sell to the masses, but it is inaccurate and damaging.

Finally, let me tell you something about writing games. They need to be written quickly and shipped the same year. If you spend 10 months of that year re-inventing the wheel, your competition is going to beat you out the door. id Software manages to stay abreast of technology, but they've been doing this for many years. If you're a new company with new programmers writing a new game, you need to hit the ground running. The tiny amount you spend on software libraries will shave months or years off of your development cycle.

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

Don't believe everything you hear on the Internet. When you are making a decision about whether to invest in software libraries, base that decision on facts, not newsgroup rhetoric.

Diana Gruber
Senior Programmer
Ted Gruber Software Inc.

BRIAN HOOK REPLIES: *I am baffled by the vehemence with which Diana Gruber objects to what are common sense assertions on my part. She misrepresents my comments appallingly. I did not state that I am familiar with software that I have never tried, nor did I claim that all external libraries exhibit all the negative attributes listed. My opinions are based on my experience working with many libraries and APIs over eight years. I have encountered bugs and performance issues with many different third-party libraries, including the run-time libraries that come with many compilers and DOS extenders, public-domain "freeware" libraries, window system abstraction libraries, 2D and 3D graphics libraries, and operating system APIs. Bugs are a part of life when you program for a living. Intel processors have bugs. BIOSes have bugs. Peripheral hardware has bugs. To believe that only a handful of third-party libraries and APIs possess bugs is naïve, and the implication that my comments are the ramblings of an inexperienced kid is offensive.*

If she wishes to assert that Fastgraph is bug-free, extensible, well documented, well supported, and fast — it sounds like a great product! I have no quarrel with that library — I've never used it, nor have I claimed such.

Even so, she states that Fastgraph supports "most" compilers and "virtually all" video cards. Which means that there is a real chance that someone who uses Fastgraph could run into problems when attempting to find support for a new or heretofore unimportant (for the end user) compiler or hardware device.

Stories such as this are not uncommon: a company is using Compiler ABC and thus gets library XYZ. However, 14 months into the project, the company switches to Compiler JKL because of unrelated technical issues, only to find that XYZ does not support JKL. I've been in this situation — it occurs often. It happened at id when we wanted to release the DOOM source code. We found that there was a dependency in the DOS version on a third-party library, which meant that the Linux code had to be released instead. While Fastgraph may be modular and extensible, most of the libraries that I have worked with are monolithic and most decidedly not *extensible.*

I agree wholeheartedly that games need to be written quickly — although shipped the same year is pushing it quite a bit. I haven't seen a professional game implemented and shipped in less than 12 months in quite some time. I haven't dismissed the notion of third-party libraries out of hand — I simply urge caution when considering committing to such packages. For example, id Software is a big proponent of OpenGL — a third-party library — and we use it extensively. As a result, we've endured poor hardware and worse drivers. Still, we feel that OpenGL's advantages outweighed its disadvantages. So it's not as if we're some kind of API Luddites who worship at the altar of NIH — we simply take into account the trade-offs that exist when selecting each of our development tools.

Finally, I don't understand her obsession with newsgroups or the Internet, but if you're thinking about buying into someone else's technology, you would be irresponsible not to solicit the opinions of others who have used that technology — whether those opinions come in the form of e-mail, print reviews, Web sites, Usenet posts, or Pony Express is irrelevant. Such commitments should be made with as much data as possible. I'm curious where "facts" should be drawn from — press releases and marketing collateral?

INDUSTRY WATCH

by Alex Dunne

HERE COMES THE GOD SQUAD. Mike Wilson has signed on a strong roster of game developers to the Gathering of Developers. Developers include Apogee/3D Realms, Epic Megagames, PopTop Software, Ritual Entertainment, and Terminal Reality. The squabble between GT Interactive and Apogee/3DR over who will publish the forthcoming MAX PAYNE title may not be the only speed bump that GOD encounters as it continues to lure developers with sky-high (about 40-50%) returns on sales, but that should be expected during a transition period like this. The high rate of return GOD promises its developers could cause some bidding wars by publishers over upcoming titles. Big winner? Developers. Incidentally, Wilson says that GOD will ship four games this year, and he's shooting to get seven out the door in '99.

THE LONELY GUY? The Sega layoffs announced on January 9 may have only trimmed 30% of the company'sworkforce as announced, but the layoff decimated the Sega of America division. Just one channel salesperson remains in that division. With Saturn sales in the can, and its next-generation system (Dural) a long way off, the company was left with few options for its sales force. I doubt that last guy is working on straight commission.

RIDING THE WAVES. AMD is weathering some pretty severe ups and downs right now. Up: it shipped close to 1.5 million K6 processors during the fourth quarter (thanks to its major deal with Compaq), nearly achieving the company's goal. Down: It has problems with declining liquidity and analysts are contemplating downgrading its stock. Up: AMD-3D technology is gaining buy-in from game developers like Dreamworks and Digital Anvil, and tool

Alias|Wavefront's Maya

ALIAS|WAVEFRONT is finally releasing Maya, its innovative computer graphics and animation application. It purports to be the next generation in 3D character animation and visual effects technology.

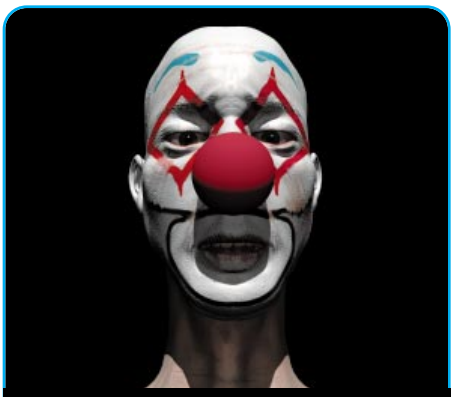


Image created by C. Landreth using Alias|Wavefront's Maya.

Alias|Wavefront developed many architectural innovations to maximize system performance. Maya combines object-oriented C++ design with a native OpenGL graphics implementation to provide high levels of throughput for computational and interactive tasks.Three architectural components give Maya power as an animation system. The first is the Dependency Graph, Maya's node-based architecture. The second is MEL, Maya's unique scripting language, which gives users flexibility in extending Maya. In addition, Maya's C++ API allows users to add custom plug-ins to tailor the program to specific production requirements. The application also incorporates a comprehensive set of features spanning modeling, animation, and rendering. Maya definitely focuses on ease-of-use; it allows artists to use a wide variety of character building tools and techniques to create digital puppets with embedded behaviors and high level controls. Once built, these digital puppets may be posed and animated by other artists without requiring them to understand all of the underlying technology.

Hardware requirements for Maya are: R4000 or higher processor, 24-bit graphics, OpenGL native preferred, and at least 128MB RAM. It runs on IRIX 6.2 or higher. Base price for Maya starts at $10,000; additional modules range from $5,000 for Artisan, to 10,000 for each other module.

■ Alias|Wavefront
Toronto, Canada
(416) 362-9181
www.aw.sgi.com/pages/maya_transition_web/

PowerMoves I

CREDO INTERACTIVE Credo Interactive unveiled Life Forms PowerMoves I, a 3D animation library that complements the company's Life Forms character animation software.

Life Forms is a stand-alone tool for creating, editing, and playing back character animation. It features a built-in walk generator, motion capture data support, graphical editing tools and its own animation libraries. PowerMoves I adds over 600 keyframed single and multi-figure 3D animation sequences to Life Forms, including office moves, sports activities, dancing, walking, and running. The PowerMoves collection of basic sequences serves as a starting point for animation projects, as these keyframed animation segments can be modified to get the specific look and

ASTS

OF GAME DEVELOPMENT

feel an animator requires.

Life Forms and PowerMoves work with Infini-D, Extreme 3D, ElectricImage, 3D Studio R4, Pixel Putty, and any program that supports BioVision or Acclaim motion capture file formats, or VRML 1 and 2 formats. Support for LightWave 3D and 3D Studio MAX is currently in development. Power Moves I sells for $249.

- Credo Interactive
  Burnaby, British Columbia, Canada
  (604) 291-6717
  www.credo-interactive.com

DeBabelizer 3 for Mac OS

EQUILIBRIUM just released the latest Macintosh version of DeBabelizer 3.

Already a standard graphics processor in desktop publishing, multimedia, and the Web, this new upgrade of DeBabelizer took "Best of Show" at January's Macworld Expo in San Francisco. DeBabelizer automatically and efficiently optimizes, translates, and processes graphics so that you can deliver content with the highest level of image quality across a variety of platforms. DeBabelizer 3 features a new, more intuitive interface that streamlines batch automation functions, allows an unlimited number of open image files, and grants more extensive drag-and-drop capabilities. The product also provides improved Web support. There are more powerful color reduction capabilities, including new built-in filters, enhanced plug-in filter support, and the ability to create SuperPalettes from any base palette. DeBabelizer 3 also features full CMYK support, more powerful scripting capabilities, and additional file formats.

DeBabelizer 3 runs on the Macintosh OS, and has a suggested retail price of $595.

- Equilibrium
  Sausalito, California
  (415) 332-4343
  www.equilibrium.com

The new user interface for DeBabelizer 3.

CodeWarrior in AMD-3D

METROWERKS and AMD have entered into a relationship, whereby CodeWarrior Professional will now support the AMD-3D technology.

CodeWarrior Professional offers developers a complete set of tools for writing programs optimized for the AMD-K6 processor and AMD-3D technology. CodeWarrior gives you inline assembly support for AMD's 3D instruction set. It will also allow you to optimize your code for the AMD-K6 processor. You can optimize for AMD-3D technology on a per-function basis, letting you exploit the speed of the new 3D instruction set. Finally, CodeWarrior utilizes vectorization compiler techniques to take advantage of the parallel instruction capabilities of the AMD-3D technology.

CodeWarrior Professional for AMD-3D technology supports C, C++, Java, and Pascal. It requires a 486 processor or higher, at least 16MB of RAM, Windows 95/NT 4.0, CD-ROM for installation, and 80MB of hard disk space. It sells for $449.

- Metrowerks
  Austin, Texas
  (512) 873-4700
  www.metrowerks.com

makers like Metrowerks. Down: Company executives have stated that Q1 may not be rosy financially, since it's switching over to .25 micron technology from its current .35 micron manufacturing process. This should be a telling year for the company.

NEW TITLES ON TOP. Those tag-team champions of the PC Data charts, RIVEN and MYST, were finally dethroned for the week after Christmas (Dec. 28 - Jan. 3) by QUAKE 2 and MICROSOFT FLIGHT SIMULATOR. The first or second spot on the monthly top seller list has been occupied by either RIVEN or MYST going all the way back to last June — quite an impressive streak.

CUC NOW CENDANT. As a result of a merger between CUC International and HFS Inc., CUC Software is now Cendant Software. The merger widens the scope of this already diverse company. Formerly, Blizzard, Sierra On-Line, Davidson & Associates, and Knowledge Adventure shared the same corporate roof with Days Inn, Howard Johnson, Ramada, and Avis. Now Century 21, Coldwell Banker, and ERA become corporate sisters too. No word as to whether low fixed-rate mortgages are now part of employee incentive packages at Cendant's game studios…

CHRIS' CAST NAMED. Cast members for Chris Roberts' upcoming *Wing Commander* flick have been named — not that you'll recognize them by name, necessarily. The stars include Freddie Prinze Jr. (*I Know What You Did Last Summer*), Saffron Burrows (*Circle of Friends*), Matthew Lillard (*Scream*), and Elise Neal (*Scream 2*). The film is now in production in Luxembourg and Germany.

IT WAS A GOOD YEAR. Bloomberg News has identified the game development industry as a "bright spot for investors in 1998." The respected financial news organization expects PC game sales to climb 30% this year to over $400 million. In addition, the market research firm, NPD Group, predicts that when all is said and done, 1997 will have seen an increase in the video-game business of up to 45 percent over '96.

Better 3D:

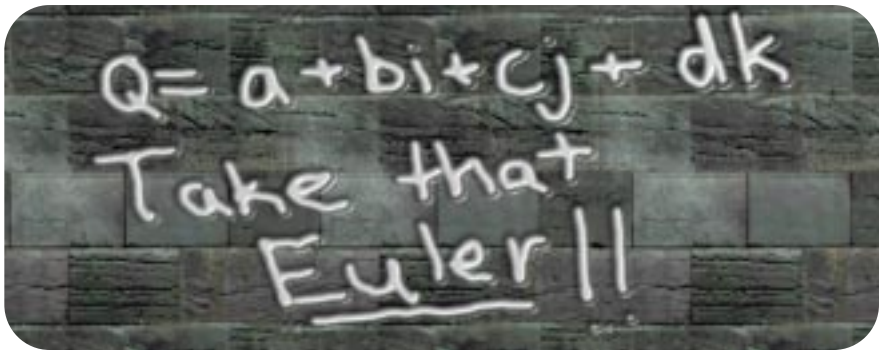
The Writing Is on the Wall

Have you ever come up with the solution to a problem at an unfortunate time? It can strike so suddenly. You know that if you don't write it down immediately, the solution may be lost forever. There are times when I wake from a sound sleep with the answer to a problem that's been bugging me for days. Occasionally, I have to jump up and crank on the computer to try it out right away. Most times though, I just roll over and go back to sleep. Epiphany or not, a guy needs his rest.

This phenomenon didn't begin with the computer age. In fact, long ago, the mathematician Sir William Hamilton found himself in the exact same situation. In the early 1830s, he was trying to extend complex number theory from 2D to 3D. Hamilton believed that a complex volume could be defined by one real and two imaginary axes. He worked on this problem for over a decade with no luck, and I might add, without the aid of a computer. Then, on October 16, 1843, while walking past the Broome Bridge to a meeting at the Royal Irish Academy in Dublin, the answer came to him. Hamilton realized that he needed three imaginary units with some special properties to describe the volume. He was so excited and so fearful of losing the answer for all time, that he carved the formula into the side of the bridge with a knife. It's a good thing he didn't roll over and go to sleep.

He called his solution a "quaternion," and it looks like this:

q = a + bi + cj + dk



Although I've never come up with something so good that I needed to carve it into the side of a bridge, I do have some ideas for animating characters that I've wanted to implement for years. Until now, I've had problems making these ideas come to life. Some of my problems are solved by better hardware and some are solved by just plain hard work. Today's problem has been solved by a combination of both. But what, you ask, do quaternions and my problems with animating characters have to do with game programming? Read on.

Now that we have the ability to create complex real-time characters, the challenge lies in bringing these characters to life. So how do I do that successfully? It really comes down to a problem of data storage and playback. What I need to store is the position and orientation of each "bone" in the character that I'm animating.

Last month, Nick Bobick wrote about using quaternions for camera animation and physical simulation ("Rotating Objects Using Quaternions," February 1998). We are going to apply the same concepts to character animation in OpenGL.

Who is this Jeff Lander dude, anyway? Jeff's a maniacal research nerd who loves the challenge of computer graphics programming when not mountain biking by the beach. He is always willing to learn more, and in fact, thrives on it. If you have anything you want to learn more about or something you would like to discuss, contact Jeff at jeffl@darwin3d.com.

Character Animation

Animation data is normally stored in keyframes. These keyframes are recorded periodically. For instance, motion capture data is often recorded at 30 frames per second (fps), the standard NTSC video playback rate. Obviously, such a high frame rate generates a lot of data. In order to save precious RAM, many game applications store animation data at a much lower rate, say 10 fps.

However, modern 3D action games try to run quite a bit faster than 10 fps. A game running at 30 fps will show three times as many frames as a 10 fps

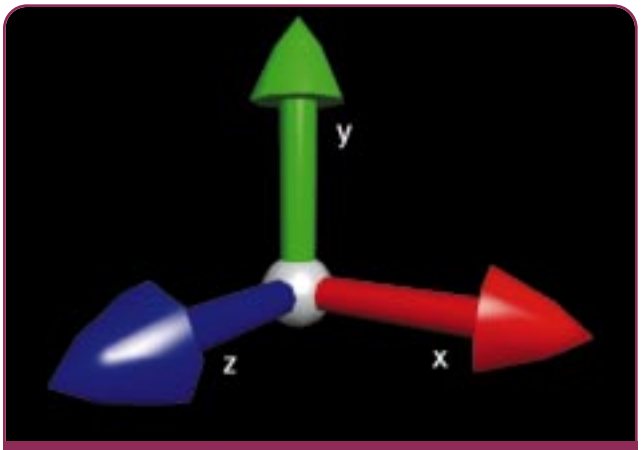


FIGURE 1A.

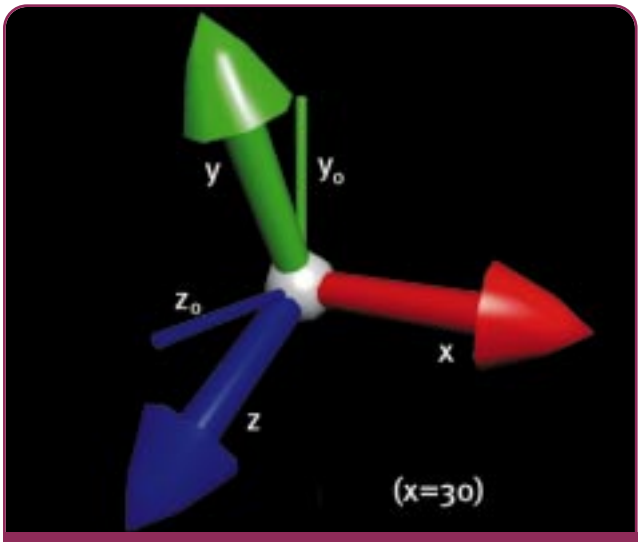


FIGURE 1B.

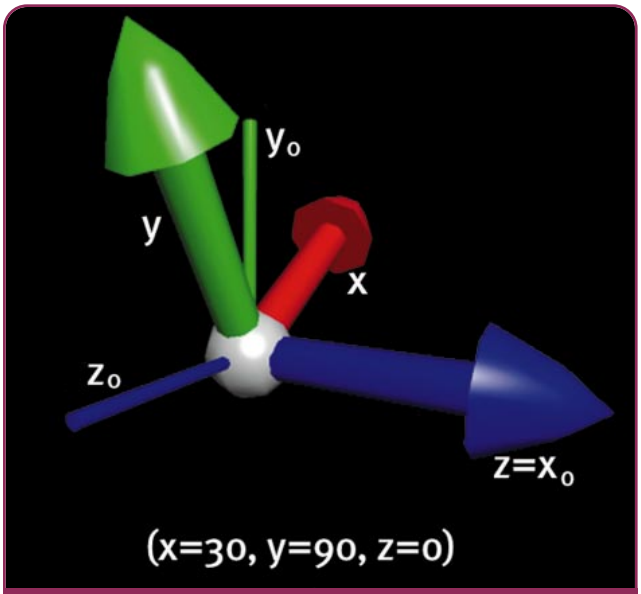


FIGURE 1C.

animation. This leaves you with a couple of options. The first option is to repeat or hold the same keyframe over three game cycles until the next animation frame arrives. This approach, however, can make your animation look a bit jerky.

The second option — and I consider this to be the better solution — is to create frames in real time that are in-between the existing keyframes. You generate these in-betweens by interpolating where the bone is at a given time between your keyframed data. Coming up with a method by which to calculate this in-between position is where I start hitting the wall (or bridge).

Euler Angles

The most common method of representing the current orientation of an object is through the use of Euler angles. This method represents the absolute orientation of an object as a series of three rotations about three mutually orthogonal axes, normally labeled x, y, and z. These Euler angles describe the three angular degrees of freedom. It's important to remember that by applying these rotations in different orders, you'll get different final orientations. So it's best to stay consistent.

If you saw the movie *Apollo 13*, you were probably amused by those guys whipping out their slide rules and complaining about something called "gimbal lock." Well, as strange as it sounds, gimbal lock can be a real problem. It's named after a situation that occurs in a mechanical gyroscope consisting of three concentric rings. Under certain rotations, the support for the gyroscope, called a gimbal, could lose of a degree of freedom.

This problem has a parallel in computer graphics. Bobick explained the situation in his February article, but it may help us to visualize it. Because Euler angles do not act independently of each other, it's possible to lose a degree of freedom. Therefore, a change to one of the angles affects the entire system. Let's look at an example. If I take the object in Figure 1a and rotate it 30 degrees about the x axis, I'll get the image in Figure 1b. I now rotate the object 90 degrees about the y axis, resulting in Figure 1c. As you can see, the current z axis is in line with the x_o axis. What we have now is gimbal lock. Any further rotation about the z axis affects the same degree of freedom as rotating about the x axis. I have completely lost the ability to rotate around the third degree of freedom. In many 3D modeling packages, it's possible to avoid this problem by creating a parent to this object and rotating the parent to gain back the additional degree of freedom. However, in a real-time 3D game, this isn't really possible.

The second problem with the use of Euler angles involves generating an in-between. In order to animate my character smoothly, I need to interpolate a new position out of the existing keyframes in my animation data. The fact that the Euler angles don't act independently of each other again raises a problem. I could represent the orientation of an object as (0,180,0) degrees about (x,y,z) respectively. This same orientation could also be described as (180,0,180). Now, what if I want to generate a position halfway between (0,0,0) and both orientations? In the first case, I would get (0,90,0) (Figure 2a), but in the second case I would get (90,0,90) (Figure 2b). Clearly, these two orientations are not

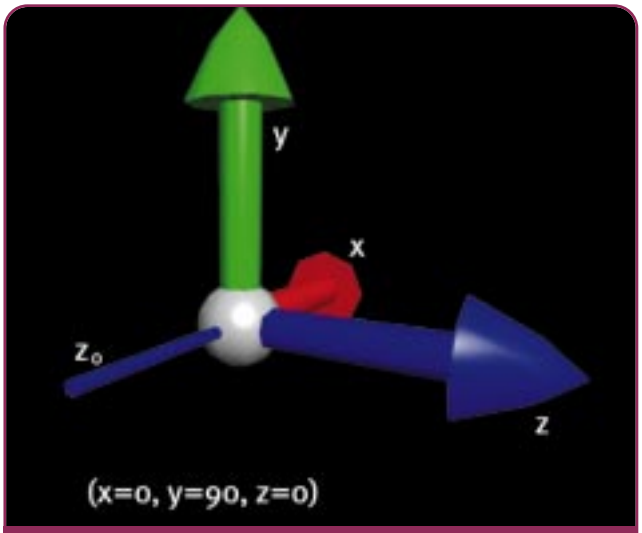


FIGURE 2A.

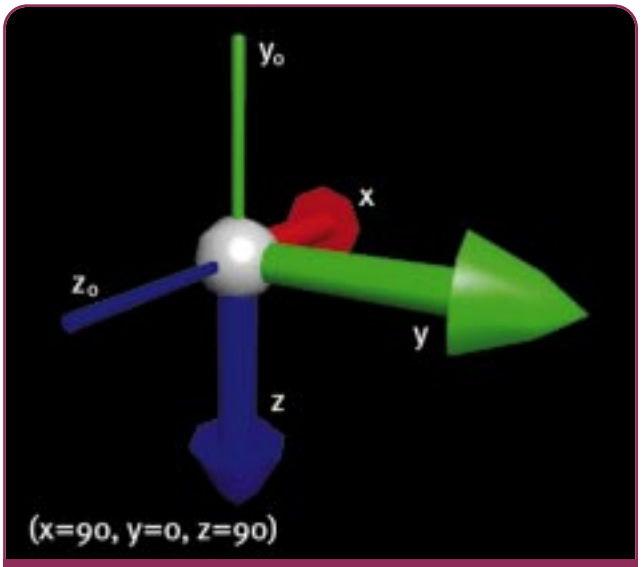


FIGURE 2B.

equivalent. These two problems with animating Euler angles lead me to look for a better way to store orientations for my animation.

Quaternions

By using quaternions to represent the orientation of an object, I can avoid the drawbacks of using Euler angles. A quaternion, when used to represent the orientation of a rigid body in space, is defined by two parts: a vector that describes the axis of rotation and a scalar value. When two quaternions of unit magnitude are multiplied together, they generate a single quaternion of unit magnitude — this is crucial to the use of quaternions for representing rotations. So, to use this system of animation, I have to convert my orientations from Euler angles to quaternions. For details on how quaternions are multiplied together I would suggest the Shoemake article that is referenced at the end of this column.

Conversion from Euler Angles to Quaternions

Rotations are defined as the following quaternions using [(x,y,z),w] notation, where (x,y,z) is a vector and w is a scalar value.

Euler ($x = \psi$, $y = \theta$, $z = \phi$)
$Q_x = [(\sin(\psi/2),0,0), \cos(\psi/2)]$
$Q_y = [(0,\sin(\theta/2),0), \cos(\theta/2)]$
$Q_z = [(0,0, \sin(\phi/2)), \cos(\phi/2)]$

LISTING 1. *Conversion from Euler angles to quaternions.*

```
////////////////////////////////////////////////////////////////////////
// Function:        EulerToQuaternion
// Purpose:         Convert a set of Euler angles to a quaternion
// Arguments:       A rotation set of 3 angles, a quaternion to set
// Discussion:      This creates a Series of quaternions and multiplies them together
//                  in the X Y Z order.
////////////////////////////////////////////////////////////////////////
void EulerToQuaternion(tVector *rot, tQuaternion *quat)
{
/// Local Variables ///////////////////////////////////////////////////
          float rx,ry,rz,ti,tj,tk;
          tQuaternion qx,qy,qz,qf;
////////////////////////////////////////////////////////////////////////
          // FIRST STEP, CONVERT ANGLES TO RADIANS
          rx =  (rot->x * M_PI) / (360 / 2);
          ry =  (rot->y * M_PI) / (360 / 2);
          rz =  (rot->z * M_PI) / (360 / 2);
          // GET THE HALF ANGLES
          ti = rx * 0.5;
          tj = ry * 0.5;
          tk = rz * 0.5;

          qx.x = sin(ti); qx.y = 0.0; qx.z = 0.0; qx.w = cos(ti);
          qy.x = 0.0; qy.y = sin(tj); qy.z = 0.0; qy.w = cos(tj);
          qz.x = 0.0; qz.y = 0.0; qz.z = sin(tk); qz.w = cos(tk);

          MultQuaternions(&qx,&qy,&qf);
          MultQuaternions(&qf,&qz,&qf);

          quat->x = qf.x;
          quat->y = qf.y;
          quat->z = qf.z;
          quat->w = qf.w;

}
// EulerToQuaternion ////////////////////////////////////////////////////
```

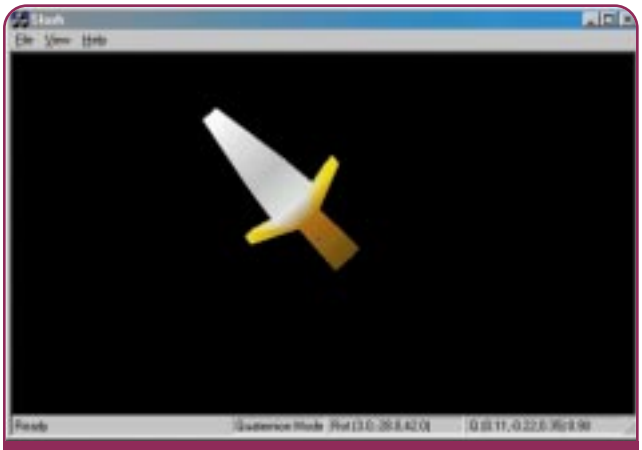


FIGURE 3.

By multiplying these quaternions together, I get a single quaternion representing the orientation of the object. The code for this conversion is shown in Listing 1.

This routine is set up to handle rotations in the order xyz. If your rotations are stored in a different order, you may need to be make some adjustments. By combining functions, it is possible to speed this up a bit. In the code that accompanies this article, I reworked it as a secondary conversion routine.

OpenGL Implementation

Since my eventual output is through OpenGL, I need to convert the quaternion back into something useful. The **glRotatef** command takes an axis to rotate around and the angle of rotation about that axis. This axis-angle representation was explored briefly in Chris Hecker's column "Physics, Part 4: The Third Dimension" (*Game Developer*, June 1997, pp.17-18). This representation is very similar to the quaternion form. The quaternion is defined as a rotation of ϕ about the axis (x, y, z) such that

$$q = \cos(\phi/2) + x \sin(\phi/2)i + y \sin(\phi/2)j + z \sin(\phi/2)k$$

Therefore, the axis and angle of rotation can be retrieved from a quaternion by simple algebra.

$$\phi = \text{acos}(Q_w)*2$$
$$x = Q_x/\sin(\phi/2)$$
$$y = Q_y/\sin(\phi/2)$$
$$z = Q_z/\sin(\phi/2)$$

This gives us the axis and angle that we need. To use it in OpenGL, the last thing to do is convert angle ϕ from radians to degrees, because OpenGL is expecting degrees (Listing 2). Since the data is now in the correct format for OpenGL, the actual display code is very straightforward (Listing 3).

Where Does That Bring Us?

The question is, "If quaternions are so great, why not use them all the time?" You certainly could. However, most people find it difficult to visualize the orientation of an object as a quaternion. People (at least 3D programming dweebs) are often more comfortable visualizing an orientation as Euler angles. This is an important consideration when you're creating a tool that will be used by artists and programmers. I'm sure people confronted with a dialog box asking them to enter the quaternion for a rocket ship would have some problems. At least Euler angles are a little more friendly.

The sample application uses the previously mentioned conversion routines to display an object using either Euler angles or quaternions. By switching between these display methods, you can see that both representations result in the same final display (Figure 3).

Using these routines, I now have a method for converting my Euler angle rotations into quaternions and then displaying them. Whew! This has been a lot of difficult stuff. But now we have the foundation. From here, I need to start animating these orientations. Next time, I'll add the interpolation code and build an application to display the in-betweens. ■

LISTING 2. *Converting the angle ϕ to degrees.*

```
////////////////////////////////////////////////////////////////////////////
// Function:        QuatToAxisAngle
// Purpose:         Convert a quaternion to Axis Angle representation
// Arguments:       A quaternion to convert, a axisAngle to set
////////////////////////////////////////////////////////////////////////////
void QuatToAxisAngle(tQuaternion *quat,tQuaternion *axisAngle)
{
/// Local Variables ///////////////////////////////////////////////////////
          float scale,tw;
////////////////////////////////////////////////////////////////////////////
          tw = (float)acos(quat->w) * 2;
          scale = (float)sin(tw / 2.0);
          axisAngle->x = quat->x / scale;
          axisAngle->y = quat->y / scale;
          axisAngle->z = quat->z / scale;

          // NOW CONVERT THE ANGLE OF ROTATION BACK TO DEGREES
          axisAngle->w = (tw * (360 / 2)) / (float)M_PI;
}
// QuatToAxisAngle  ////////////////////////////////////////////////////////
```

LISTING 3. *Using quaternions in OpenGL.*

```
// TAKE THE BONE EULER ANGLES AND CONVERT THEM TO A QUATERNION
EulerToQuaternion(&curBone->rot,&curBone->quat);
// QUATERNION HAS TO BE CONVERTED TO AN AXIS/ANGLE REPRESENTATION
QuatToAxisAngle(&curBone->quat,&axisAngle);
// DO THE ROTATION
glRotatef(axisAngle.w, axisAngle.x, axisAngle.y, axisAngle.z);
drawModel(curBone);
```

Hardware for OpenGL Developer Tools

When developing an advanced real-time 3D game engine and production tools, it's important to work on decent hardware. Likewise, when artists and designers work on those tools, performance means productivity. Now obviously, the 3D artists, modelers, and animators should have great systems. Any of the fully OpenGL-certified high-end graphics cards would also work well for production tools.

But what about the programmers, level designers, balance testers, and so on? I would like a great production environment for everyone. Luckily for me, certain "market pressures" have pushed some consumer graphics hardware towards OpenGL support. For me, OpenGL support means that the images that I see in my modeling and animation program and in my toolset are identical. The material and display parameters match exactly. This is very important at the design level no matter which final game application API is used (OpenGL, Direct3D, or proprietary). Low-cost OpenGL acceleration allows companies to get high-performance game production stations for all the team members.

To be a good system for game production, a graphics card should support acceleration in multiple windows, because most 3D development tools require more then one viewpoint. It should support most, if not all, display features of the target platform. It should also have decent performance in the tool, as well as being able to run the actual game application at a decent rate.

The 3Dlabs Permedia 2 chipset is an excellent choice for a low-cost OpenGL tool platform. 3Dlabs has much experience working with OpenGL through its Glint family of cards, and it shows. Its OpenGL drivers are stable and robust. The Permedia 2 adds acceleration of glLines, which really can help a development tool. The lack of some blending modes may be an issue when previewing the application. However, the price/performance ratio is great.

The new Nvidia Riva 128 chipset is a great performer. It has excellent potential as an all around perfect production card, as well as great speed and image quality and fast Windows acceleration. Right now, the OpenGL drivers are in testing, but I have high hopes for this card.

The 3Dfx Voodoo Rush chipset is very interesting. It's certainly the king of the hill (right now) as far as actual game performance. However, the OpenGL support beyond full-screen game applications is questionable. The alpha driver that was released was unusable in a development environment, and I have yet to see whether the new beta OpenGL driver will accelerate graphics in a window. That said, a Voodoo card to run the game application combined with either of the other boards for the production tools is a great environment.

The recent announcement that Microsoft is working with SGI to speed up and improve OpenGL driver development could change things dramatically. I will be watching closely.

—Jeff Lander

REFERENCES

I've only scratched the surface of the math behind quaternions. But as you can see, a whole lot of history rests behind them. People interested in reading more about quaternions and their use in computer graphics should read these sources.

Hamilton, Sir William Rowan. "On quaternions; or On a New System of Imaginaries in algebra." ***Philosophical Magazine*****, XXV:10-13, 1844.** This is the original paper describing the actual discovery. However, it's very mathematical and probably not necessary unless you have the desire to learn the underlying theory.

Shoemake, Ken. "Animating Rotation with Quaternion Curves." ***Computer Graphics (Proceedings of Siggraph 1985)*** **1:245-254, 1985.**

Shoemake, Ken. "Euler Angle Conversion." ***Graphic Gems II.*** **Academic Press: 222-229, 1994.** These papers were essential to my understanding of the conversion process. The paper in *Graphic Gems II* actually handles the general case of different rotation orders.

Watt, Alan and Mark Watt. ***Advanced Animation and Rendering Techniques.*** **New York: ACM Press, 1992.** This is a very good book that helped me with the background and comparisons between Euler angles and quaternions.

Laura Downs has written a proof that quaternions really do represent rotations. It's available at http://http.cs.Berkeley.edu/~laura/cs184/quat/quatproof.html. Thanks to Ian Wakelin of Rhythm and Hues Studios for turning me on to that site.

Also be sure to read Nick Bobick's article "Rotating Objects Using Quaternions," in the Feb. 1998 issue of *Game Developer* for other uses and a matrix-based approach.

Communication Triangles

From love triangles to children manipulating their divorced parents — if you get three people together, you usually get problems. My theory is that this triangle issue is behind a common problem that artists face: dealing with small team conflicts. Let's explore this by starting with an example.

Our Beloved Team: Albert, Dave, and Pam

Albert Artist is a contract artist for Dave Developer, an experienced, one-man game development "company." Dave's got a signed deal with Pam Producer, who works for an up-and-coming game publishing company and was just promoted from assistant to manager of her own project. Dave's worked with Albert before, but they didn't work closely together. Pam doesn't know either one very well, but has spent more time with Dave, negotiating the contract and approving the game design (which they've agreed is somewhat vague, but good enough, given the aggressive development schedule). Let's start with Pam, sipping her morning mocha and reading the day's first wave of e-mail:

```
from: Dave Developer (dave@gameguy.com)
to: Pam Publisher (ppublisher@bigdogs.com)
subject: Art Quality Control
As far as accepting artwork, I think this
should be my job, since I'm the one ultimate-
ly responsible for the product. I'm open to
other ideas, though.
-Dave
```

Pam sighs and leans back, wishing she could find the perfect outside developer for this project — some super-group with clean, efficient code and excellent artistic abilities and tastes, all the while affordable, available, and easy to work with. But this project is just a basic sequel, not exciting enough to attract the top-notch developers. In fact, her boss isn't jazzed about it either — that's why the budget's not huge and the marketing effort is strictly average, which is also why she's working with a small, unknown company such as Dave's.

Still, she's going to make the game as stupendous as possible. If she can get a really solid game (fun gameplay, good art, bug-free code, on time, and within budget) out of Dave and his contractors, she'll get good marketing and good distribution. With a little luck, it'll do pretty darn well in retail. That will be quite an achievement for her.

And Dave can do it — she knows he's good. The code's going to be great, but she is a little worried about the artwork: is it going to be any good?

Her coffee forgotten, Pam rereads the e-mail and ponders Dave's assertion. It makes sense, and she trusts Dave to do the right thing for the project, in general. The thing is, she's afraid that Dave can't tell first-rate characters from clip-art, so she's got to somehow ensure that the art is great. "I've got to do what's right for the project here," she reminds herself. This essential truth fuels her tickle of concern about the artwork as she composes her reply.

```
from: Pam Publisher (ppublisher@bigdogs.com)
to: Dave Developer (dave@gameguy.com)
subject: Art Quality Control
>>I think this should be my job since I'm the
>>one responsible for the product ultimately.
>>I'm open to other ideas, though.
I need some additional assurances that the
artwork quality is going to be adequate. I'd
like to see samples from the artist and get
some proof that this art is going to be AAA-
grade. You're in charge of the project -- I
just want to confirm the art quality.
-Pam
```

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

Hunched in his darkened workspace, Dave's thumb-drum solo takes on a louder, almost militant beat as he rereads Pam's reply. "Is Pam micromanaging here?" he asks himself. "I guess I can understand why she'd want to see proof of artwork." Still, a nagging feeling of doubt eludes his mental fishing. On one hand, her request makes sense — she's just trying to make sure it's a top-notch product. But it definitely bugs him that she's second-guessing his choices. He knows that if he's not in control of his own project, it'll be Disasterville.

But Pam isn't like that. So far, she's been really good about not micromanaging. Besides, it's not like he has a choice — she's the publisher, she's got the cash — and all she wants is sample artwork. Semi-reassured, he writes to Albert.

from: Dave Developer (dave@gameguy.com)
to: Albert Artist (artdude@cheapisp.com)
subject: Art Quality
Hey Albert,
So I need some samples to show to the publisher. They're curious what you do, how it looks, and such. Forward me a couple .JPG attachments, O.K.? They want to see nifty stuff, such as those SIGGRAPH demos or something all flashy. Whatever you've got handy. -Dave

Albert heaves a long-suffering sigh. Why? He's already shown Dave his demo reel, and Dave said, "Rockin' art!" and didn't bring up art skills again. So what's going on now?

He herds the cat off of his lap and leans back, rubbing his eyes. Something important is happening here, his intuition tells him. There's been a subtle shift, and now he's not sure who's calling the shots. What if the publisher doesn't like his work? Will he have a second chance? What if Dave likes something and this mysterious publisher doesn't? What then?

Albert's chair squeaks as he slowly spins 360 degrees. He knows that he and Dave can do good work together — Dave will give Albert the space he needs to make quality artwork, and Dave will help Albert make it fit into the technical limitations of the game. But with a third party involved, he's not so sure. Nightmares flit through his head: end of the project, Pam casually says, "Um, how about a more, y'know, cartoony feel?" He might be stuck begging for approval from the headless monster of their publisher's art direction, as Dave stands helplessly at his side. How can anyone make decent art under these conditions?

Options

In my opinion, this project is in trouble. "Hey, whoa!" you say, "What's the big deal? It's just a simple project, and all these people possess good intentions and reasonable judgment. Heck, nothing's even happened yet — they're only talking about a problem!"

First, the timing issue: On most bad projects, once we see actual problems (someone quitting, major design changes, and so on), it's too late to do much about it. The time for quick action is when the project is teetering on the brink — when everyone's talking about it, but nothing's happened yet.

Second, the most important risk to the project is loss of goodwill, not quality of artwork. If Pam isn't careful, she could easily give Dave and Albert the impression that she's an ignorant boss who wants to play developer and get her "touch" on the product. If Dave or Albert overestimate or get defensive about their skills, Pam will probably have to confront them with a truthful assessment of their roles in the grand scheme, and that might cause a very severe loss of trust.

So let's assume that our intrepid team is reasonable. Yeah, they're all good, fair, hardworking, and honest. They've promised to floss twice a day and to make every effort not to get pointlessly emotional about this. Even so, I still say that the project's goodwill is at serious risk. Let's appreciate the gravity of the problem by looking at some possible solutions, starting from Albert's point of view.

One option for Albert is to insist on taking art direction from Dave. That's how things currently are — legally Dave and Albert have a contract and Albert has no connection to Pam. Theoretically, if Albert keeps up his end of the deal with Dave, he has nothing to worry about. Unfortunately, that's not realistic, because Pam holds Dave's purse strings. If she's forced to work through Dave for artistic concerns, this will quickly prove frustrating for everyone: Pam would rather talk directly to Albert, and because Dave isn't an art director, he's not going to be good at, or enjoy, taking Pam's criticism and turning it into improved artwork. Poor Albert will labor to decipher Pam's intent through Dave's warped filter. This is a likely recipe for poor art quality, as well as frustration all around.

Another option is for Albert to work side-by-side with Dave, and for both of them to report to Pam. Pam gets direct access, and Albert takes responsibility for (and has more control over) the artwork quality. This is a big change. Albert will probably be paid by Pam. But Dave's control is weakened because Albert is now a peer, not a subcontractor. And Pam's job gets harder. No one person (besides Pam) is responsible for the project: if Albert's artwork is late, that's not Dave's problem anymore. So, Pam becomes involved in project management and takes over more of Dave's duties. This may create resentment between her and Dave. What if Albert isn't qualified for the job? Designing artwork is actually quite different from building art from a design. In fact, Albert himself may not want the job. Most production artists leap at art design/lead job opportunities, but happy production artists may turn down stressful situations such as this.

A third option is for Albert to try to compromise. Satisfy Pam by discussing art quality directly with her, while still working under Dave. Compared to the previous two options, this one sounds more reasonable, and it has the appeal

of "good faith." But it's high risk; it could combine the worst parts of the two previous options. Also, Albert loses because he would have the responsibility of art quality without the recognition, power, or pay of the actual job. For instance, it's likely that Pam will ask Albert to make art changes. Since she's not an art director, they may not be good ideas, and Albert isn't in a position to object. Another problem is simply that Albert now has two bosses. What will he do if Pam and Dave give conflicting assignments?

So, neither Dave nor Pam is really qualified to direct artwork, and Albert wasn't really hired to design the game's "look" — he was hired to be a production artist. This leads us to another seemingly reasonable idea: hire an art director (let's call her Ann). If someone can be responsible for art direction, it relieves the pressure on Albert and Dave, and Pam has someone to trust. Another important advantage is that Pam can use Ann's strong résumé to make the whole project look better, which will help get internal support for better budgets, marketing efforts, and so on.

But, of course, there are drawbacks. First, we have the obvious problems that come with a larger team (more complex communication and greater cost, to name just a couple). Game developers are well aware of how nightmarish huge projects can be, and it should be obvious that our little threesome is not predisposed to throwing people at a problem. But perhaps Ann could work only part-time, keeping costs low, and if she's good, the other problems may be minimized.

The real problem is finding someone good. In this case, "good" means an experienced, available, affordable art director who can also walk into this delicate situation and actually improve things. It would be very easy for Ann to give Albert orders that accidentally or carelessly conflict with Dave's previous direction, which is a quick road to distrust (and thus disaster). Art directors' jobs require debating with programmers. Because Dave is "in charge" in addition to doing the programming, the art director's going to have to be a darned good communicator to get Dave's trust and respect, earn Pam's confidence, and keep the art quality high, all without stepping on anyone's toes.

Solutions

Yuck! What a bunch of weak options. None of them is the nice, neat happy-ending choice for which we'd all hope. Let's step back and ask what our beloved team could do to make the whole situation better.

They need to realize how important this issue is, and put a lot of effort into permanently solving the problem. That means understanding the actual risks: damaging goodwill and thus, the project's future, as well as potentially having poor artwork.

Ideally, you don't have to choose, but I believe a few talented, cooperative, focused people with a little money produce better products than a team that has a pile of cash and a bunch of uncaring, uncoordinated workers. That's why creating and preserving goodwill is such an important skill. Unfortunately, goodwill is incredibly delicate. It's part of both Pam's and Dave's jobs to protect the project against such disaster.

How would awareness change the situation? Here's one possible outcome: Pam starts by taking a close look at what, exactly, worries her. What would be satisfactory art quality, and what would not? What kind of proof will be enough, given that she knows she's not an art director? If she's really good, she'll also probe for underlying "icky issues" — unpleasant but true motivations for her concern. Is the problem really art quality? Perhaps she's subconsciously looking for some direct control over production. Or perhaps she's interested in forming a line of communication around Dave because, deep down, she doesn't trust him completely. Once she addresses these questions honestly with herself (and let's assume that she's comfortable with her own motivations), she will use those answers to set concrete, achievable objectives.

Next, Pam meets in person with Dave and explains exactly what worries her and precisely what proof of quality artwork would make her comfortable. She might carefully use other products as artistic references (some developers hate this, but it works), such as "I think we need terrain like in MAGIC CARPET — see this screenshot? Like that, but maybe simpler — and with animation as good as ROAD RASH." If she uses reasonable references, Dave will likely be open and grateful for a specific goal rather than a vague, unsolvable concern. Again, Pam must be very careful not to ask for unreasonable goals such as "Art that is better than RIVEN, but real-time, and with flowing hair!" This will definitely not impress Dave, and may well lead to serious conflict. After this has been defined, Pam and Dave have to agree upon who approves the art. This is really Pam's decision — if she feels comfortable that Dave understands what she wants, she will ask him to decide. Otherwise, she'll put off the decision until she knows whether or not Albert can handle it.

Once Pam and Dave agree on what is "good enough" art quality, they'll explain it to Albert (who will find their explanations of quality artwork to be overkill). Pam and Dave together will ask Albert to show that he can provide that quality of artwork. As Albert prepares samples, they'll stay in close communication, exploring what is reasonable and reassuring each other that they're all striving for the same thing: the best possible product, in whatever way they can.

After Pam and Dave take a look at Albert's artwork, it's decision time. First, they'll decide whether or not Albert is capable of art direction. If they don't agree, Pam's got a serious problem. She may well decide to use a different developer rather than attempt to patch such a major rift.

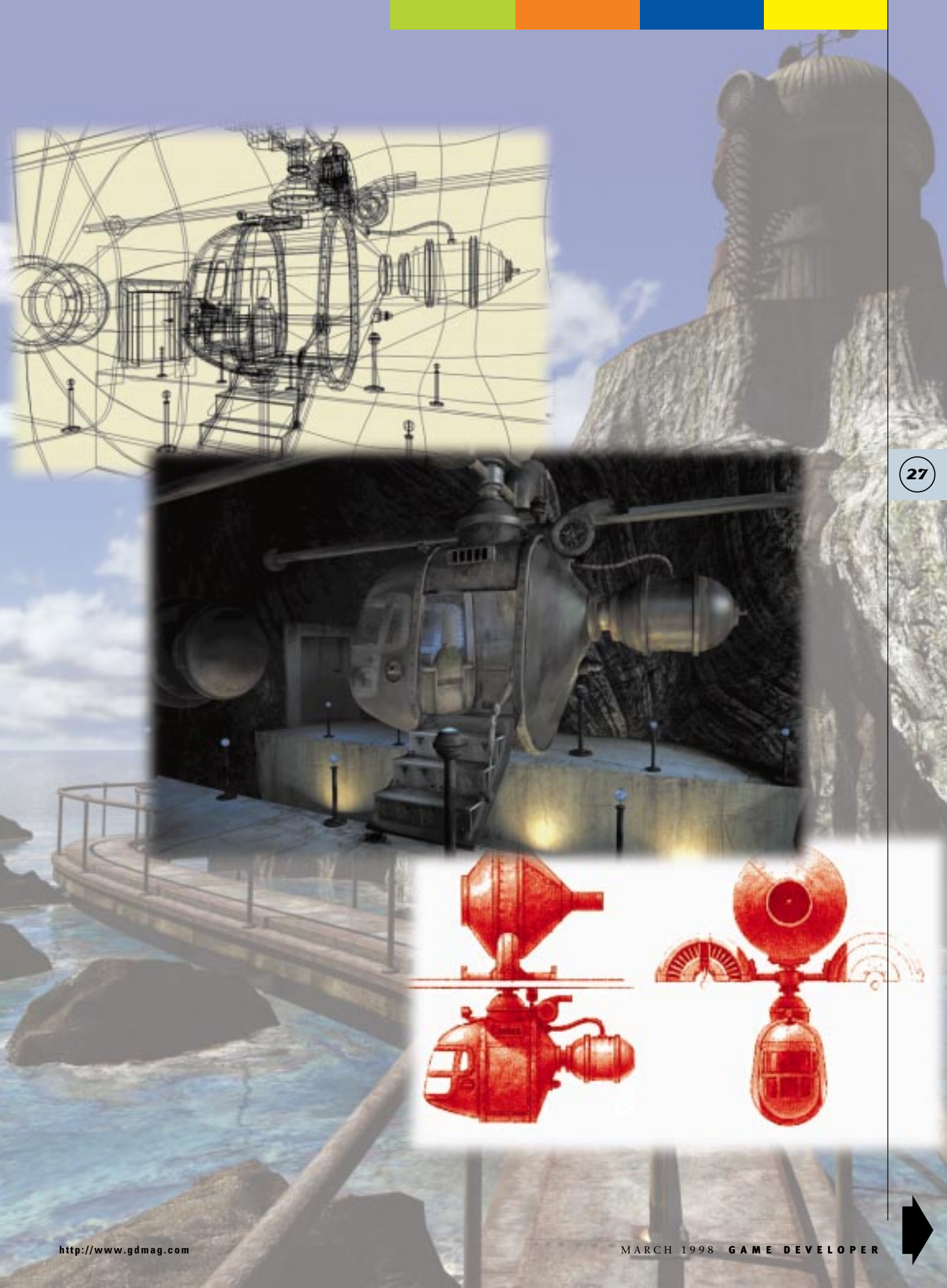
That should give you the idea. The decision process would continue in this way until they are all satisfied that the game is good enough or they give up on the project (a perfectly reasonable option — better than wasting time, money, and effort if the project is essentially flawed).

It's tough creating good games, and technical challenges are only part of the problem. I write about it often because I think that we all need more of this "psychological" problem-solving. Tell me what you think at column@vectorg.com. ■

PROTOTYPING 3D GAMES: LESSONS LEARNED FROM RIVEN

To get the most bang from your prototyping buck, consider these two essential guidelines: Prototype Early and Prototype Often. By definition, prototypes allow you to demonstrate new ideas in a rudimentary form, examining and considering different possibilities before committing resources to those ideas. Waiting until the release of your alpha to test your first prototype is generally a useless gesture: your alpha represents most of your final game. If you discover at this point that the design requires a major overhaul, it will probably be too expensive to undertake. Instead, create your first prototype for the initial design meeting — and then never stop. According to Rand Miller of Cyan, "Whether you're showing it to someone or just using it

BY NICOLE LAZZARO



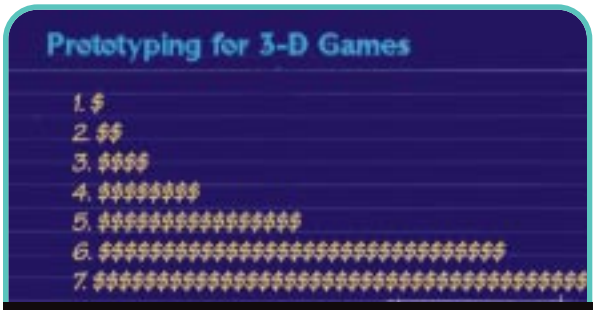


FIGURE 1. *As the game develops from stage to stage, the cost of making a change increases exponentially.*

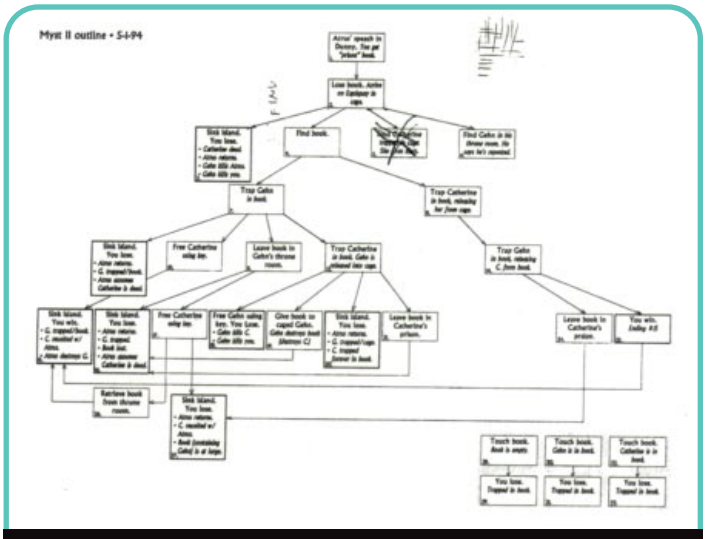


FIGURE 3. *Cyan used flowcharts like this one from 1994 to refine RIVEN's more complex story structure.*

for yourself, the interesting thing about a prototype is that the more feedback you get, the easier it is to fix things before you lock them down" (Figure 1). In light of these principles, this article will take an in-depth look at cheap — and not so cheap — prototyping techniques used to design RIVEN: THE SEQUEL TO MYST.

During the early days and weeks of a project, the design is small and prototypes are simple to make and revise — clearly, most of your prototyping should be done at this point. Think of your design process as a spiral that iteratively visits the four process quadrants of Design, Prototype, Test, and Learn, starting with your kick-off meeting at the center and time (and costs) increasing outward with each loop (Figure 2). The more cycles that you complete earlier in development, the tighter the spiral becomes and the more money you save. Since each iteration represents the validation of good design ideas and the elimination of bad ones, you can be assured that your project matures on the inherent merits of its design and not merely on its creators' enthusiastic ideas of what it should be.

Obviously, elements such as art direction and code performance won't be prototyped until later in the design process. However, general game concepts, basic character interaction, and what the player will experience can — and should — be worked out before you've purchased your first pixel. Since you haven't yet made a design commitment, your prototypes can be very rough and cheap as you work out these important details; later on, as your designs solidify and become more complex, they will become much harder to prototype and test.

FIGURE 2. *The Design Spiral: separate the good ideas from the bad early in production while they are still cheap to fix.*

Prototype the Concept and Story

All great adventure/mystery games focus on the story. Prior to puzzle design, you should explore your game's structure and the player's goals. Your new world won't have an integrated adventure until it has plot twists and major characters. Nonviolent 3D games are driven by exploration; without a unifying narrative, the experience is of walking from one obtuse puzzle to the next.

Simple, one-page flowcharts excel at prototyping during this stage because they hold just enough detail to represent the story's structure. For RIVEN, designer and co-director Robyn Miller created flowcharts to map out the different endings. These flowcharts offered RIVEN's designers the flexibility to depict the consequences of the player's actions and the multiple ways to end the game (Figure 3). Later on, when the Cyan team designed the puzzles and environments that surrounded these plot points, the skeletal flowchart prototype was easier to use than a 40-page treatment.

Once you've established the outline of the story, your 3D game needs a place where the action happens. As he did for MYST, Robyn sketched maps of the different islands on graph paper, labeling the paths and buildings that the player would encounter along the way. During initial game design meetings, Robyn could erase and redraw as

Nicole Lazzaro, president of XEODesign Inc. and production manager for the French and German versions of RIVEN, has for eight years combined her knowledge of cognitive psychology, video, origami, HyperCard, and the Xerox machine to create customized 3D environments. XEODesign provides design services to numerous companies including Broderbund, Mindscape, and Maxis. For more information on prototyping and a schedule of upcoming seminars (including this year's Computer Game Developer's conference) visit www.xeodesign.com or send e-mail to nlazzaro@xeodesign.com.

much or as little as required during the evolution of the design. "There was an awful lot of scribbling, scratching, and erasing at those first meetings," remembered Robyn. "But by the end of a few weeks, we had recorded the basic game play for the world, though it wasn't too pretty."

The overhead view concentrated design on the location of the puzzles in each environment and how those puzzles related to the others on the five islands. Done with pencil on graph paper, conundrums such as the location of the waffle iron puzzle could be planned with a minimum of fuss (Figure 4). If Cyan's artists had already modeled the islands on the computer, those same changes might have taken weeks instead of minutes. Pencil sketches of the big picture kept the ideas flexible and easy to change.

Prototype the Fun

To invent a new game paradigm, prototype the fun. In order to go where no other title has gone before, you may need to experiment with the game play outside of the computer — especially when your game's technology hasn't been invented yet. Putting your design on the computer before the basic game concepts are in place increases the likelihood that the end result will resemble someone else's good idea. Leaving behind the electronic limitations of a flat screen and a mouse frees you to discover interesting interactive possibilities.

Ideally, prototyping game play should spawn "outside the box" thinking; knowing what makes an activity entertaining will help create its analogue within the game. Get the creative juices flowing by using a number of different prototyping tools. Don't limit yourself — a video camera, a cell phone, an office cubicle, a roll of butcher paper, and even a toy car can assist in prototyping (Figure 5). Your imagination and creativity will take care of the rest. "If you're talking about prototyping," said Robyn, "your tools should allow you to be flexible. They should allow you to change things right up to the last minute." Just remember that the cost of those changes increases as the design progresses (Figures 6 and 7).

Given the fact that RIVEN's primary storytelling relationship is between the player and the environment, Cyan's designers prototyped the fun before creating any art by hosting virtual tours of the RIVEN maps to test the basic outline of the game. "We sat down with people from the company and verbally walked them through the game, told them what they saw, and asked them what they'd do — almost a role-playing kind of thing — before there were visuals," Rand explained.

This verbal prototyping provided flexible and cheap feedback on story and game play because all that was necessary were maps and puzzles ideas. "In my mind," said Rand, "it was more to see how people responded to exploring this place… you're surprised a lot of times by what people get and what they don't get." Before designing any of the art, Cyan's creative team needed to know what puzzles people would find interesting, which ones were too hard, how players built the story from the environments, and what sorts of actions players found fun to do. Because they role-played the puzzles, RIVEN's designers could see in the players' faces what was fun about exploring and discovering the story and world, and they could pursue ideas created spontaneously by the group. Robyn

FIGURE 5. *Playing games away from the computer can prototype what makes something fun.*

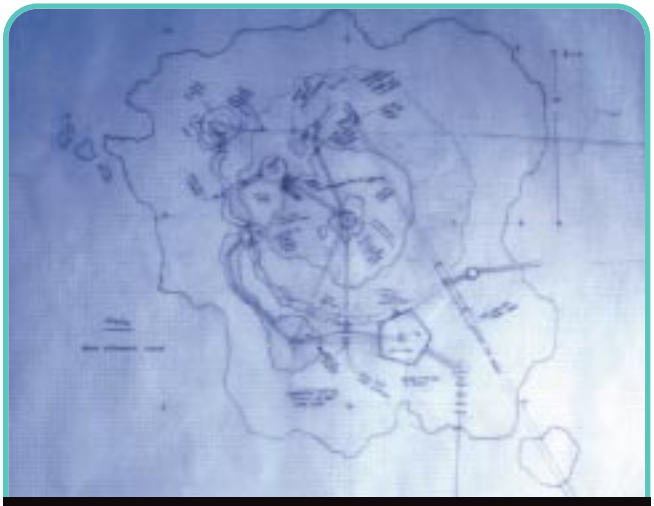


FIGURE 4. *The power source for the Fire Marble domes always involved the waffle iron puzzle, but its location changed from the Book Assembly Island to Temple Island.*

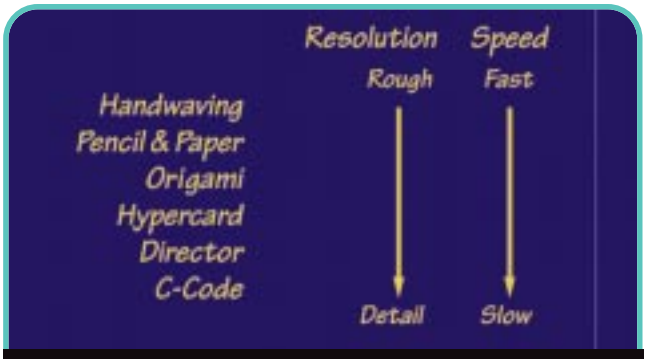


FIGURE 6. *Prototyping tools that cannot represent a lot of detail are generally faster than those that can more accurately represent the final product.*

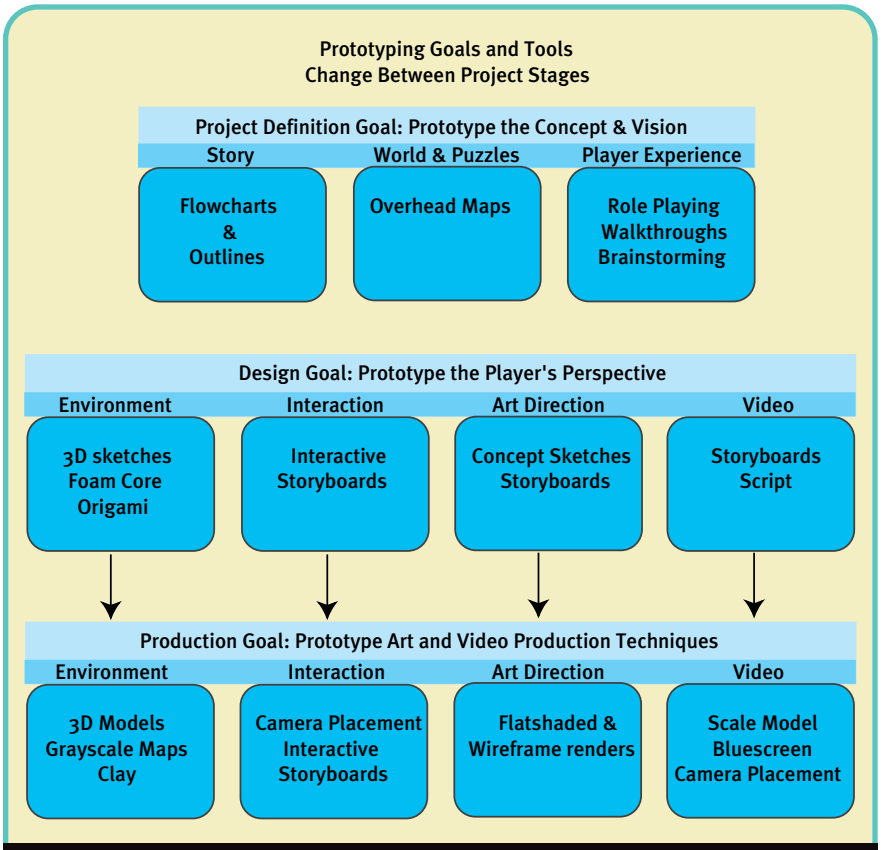


FIGURE 7. *As a game goes from concept to design to product, different aspects of the design require different prototyping methods.*

remarked, "It worked really well, and we did it earlier on MYST, too. At least it gave us an idea that what we had designed was going to work."

Prototype the Art

After outlining the essential story, location, and game play, the focus of your design should shift to what the player will see and hear. Rough line drawings can refine how objects and terrain will work together to build the environment and story. At this stage, concept sketches on paper will help you explore ideas more quickly and create a consensus of the story world to be built in 3D (Figures 8 and 9).

Low-resolution prototypes have two benefits: they are quicker (and thus cheaper) to make, and they focus discussion on broad concepts rather than on details such as color and texture maps (Figure 10). By limiting the number of items and keeping the drawings rough, you can make 50 black and white sketches for the cost of a single RGB image.

Unlike paintings in a museum, game art unfolds in response to the player. As RIVEN's designers discovered, paper-based sketches of objects and terrain didn't provide enough feedback when the game design focused on player interactions; these elements required new prototyping methods.

Just as role-playing can prototype player interaction from maps, interactive sketches and storyboards make designing art that responds to player behavior much easier. Scanning storyboard-style sketches into the computer and making them interactive is an economical way to create useful 3D environments early in the prototyping stage — line drawings in HyperCard are easier to change than Softimage-rendered images. With an electronic storyboard, the designer can click through the environment and make changes as easily as erasing lines on paper.

In RIVEN, game play changed the look and function of one particular water-filled room several times during art production; at one point, even the waffle iron puzzle floated here. Evolving the design after the room was finally rendered required throwing out several screens, updating the geometry, adding new textures, shaders, and lighting, and rendering new art (Figure 11). By comparison, switching between different concepts while still in sketch form would have sacrificed a few pieces of paper and some pencil lead. "Next time," said

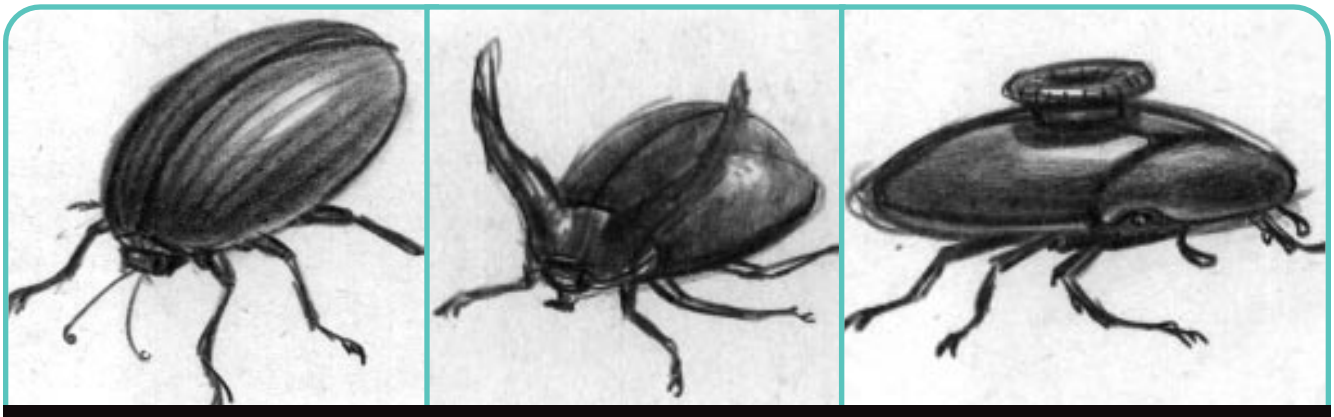


FIGURE 8. *Richard Vander Wende started the design for Gehn's inkwell by drawing many bugs.*

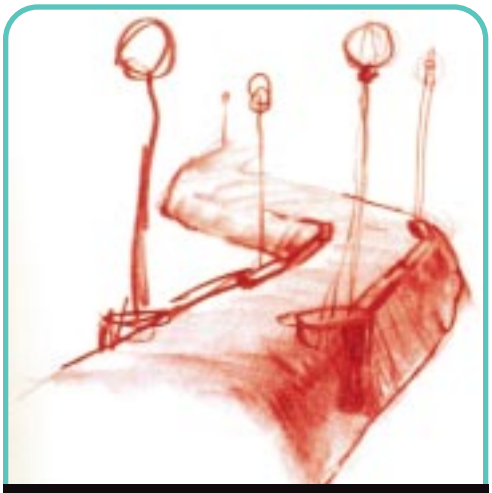


FIGURE 9. *A hand-drawn sketch at this resolution can be created faster than working with a 3D modeling package, even in wire frame.*

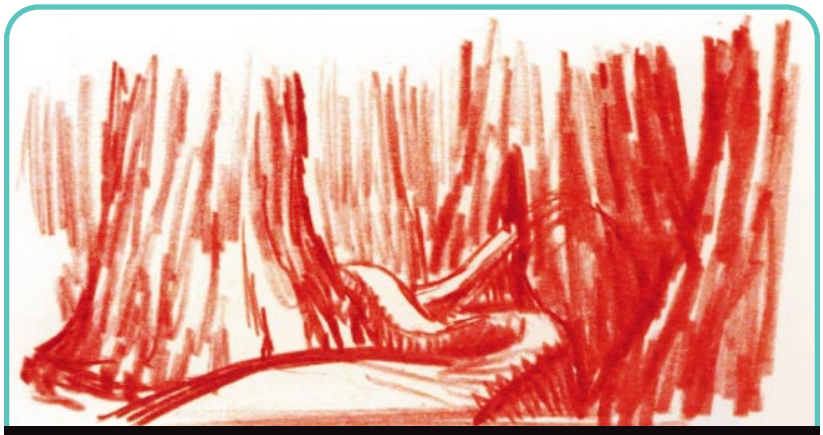


FIGURE 10. *In this Riven forest, rough art quickly prototypes the relationship between the player's path and the trees.*

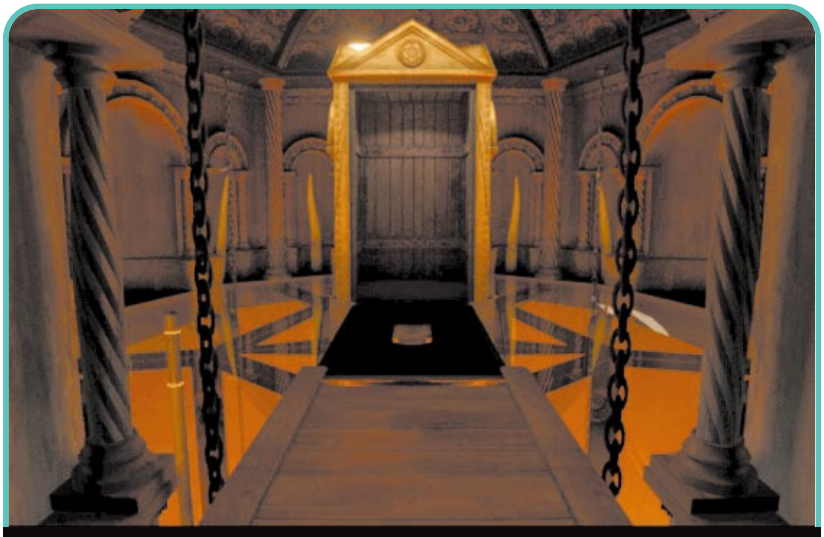


FIGURE 11. *This room holds a submersible elevator and went through several artistic and functional changes. Changing the purpose of this room during the rough sketch stage would have involved fewer production hours — or days.*

Robyn, looking back, "I personally would want to set up the whole thing with storyboard-style sketches, so you could have a playable or semi-playable game in rough-sketch form. I think it would save time in the end because we spent so much time changing this thing, which was for naught because we threw stuff away as we changed it."

Rand Miller and designer and co-director Richard Vander Wende, however, disagreed somewhat with Robyn and cautioned that too much detail in a rough-sketch prototype interferes with the final art direction. As Richard explained from his previous experience in animation, "Many painters and illustrators I know will produce preliminary sketches, thumbnails, or even drawings. Quite often, they adhere strictly to those drawings, and the final version lacks a certain freshness. [With RIVEN], there was an element of spontaneity — we didn't always know how it was going to work. We weren't always trying to copy exactly some sort of prototype version, which I think was great."

Prototype Media Production Techniques

With RIVEN's style of interface, the settings for the 3D camera position, height, tilt, and viewing angle all determine what the player sees. Done well, a sequence of still screens creates a sense of motion that draws the player in; done poorly, however, it invites player confusion that even wonderfully rendered 3D art cannot dispel. As Richard pointed out, "You just can't put the camera where the mannequin's eyes would be all the time. More often than not, the camera needs to be cheated one way or another to allow you to see more of the stuff that you would naturally be focusing on — sometimes in small rooms, it's even pulled back through walls to create a wide enough view to contain everything of interest."

"Although it seems fairly natural when you click," said Rand, "we didn't take for granted how much you move — a lot of work went into how far we set the next camera position and what angle it would be placed at, and how much you would see out of the corner of your eye to inspire a turn."

Although the design team didn't create full storyboards for RIVEN, they did develop an ingenious world assembly tool in HyperCard called WorldStarter, which made positioning the camera in the computer-generated models easier. To save time and to get more iterations, they started with wireframe and flat-shaded views along a path or around a room and then modified the camera position and angle until the navigation looked smooth (Figure 12). Because the artists could click through the images just as a player would, they could easily grasp how it felt to move

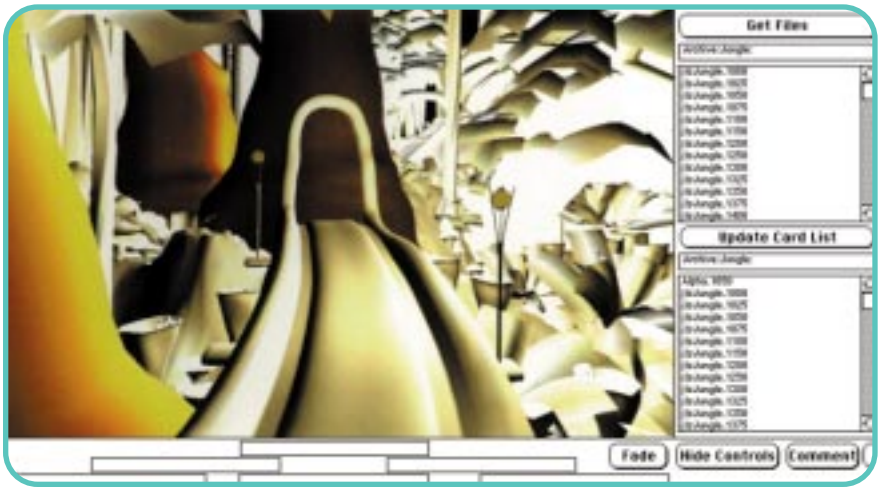


FIGURE 12. *Flat shaded images made these prototypes faster by creating images from the models without complex lighting and textures.*

through the space (Figure 13). By building RIVEN in this fashion, subtle problems became apparent at an earlier stage of the art production. When inconsistencies arose, the designers could make changes and tweak more confidently than they would have otherwise, as their prototypes provided more context. As Rand remarked, "Probably most people don't realize the extent to which every camera position was cared for there, and what we went through to create them."

In time, more screens were added. Detailed renders replaced the flat-shaded views (Figures 14 and 15). "If we had pictures that we needed to add," Rand said, "we'd walk through to that area, get to that dead end, and then just start adding in the new pictures. Since the feedback's immediate, we weren't afraid to tweak things, which was very important to us."

Prototype Terrain

All too often, production prototypes show you what not to do. For MYST, Robyn created a grayscale topographical map for each island that was then extruded in Strata, with the lighter-shaded areas being higher in elevation than the darker ones. He considered using the same process for RIVEN, but he had some concerns: not only were RIVEN's islands much bigger than MYST's, the puzzles required more precise control of the elevation (Figure 16).

As a production test, Robyn modeled Gehn's Survey Island in clay and bought a 3D pen to digitize its contours (Figure 17). "The only way we could get it to look neat on the computer was to use tons of geometry, so much so that the computer wasn't handling it very well," recalled Robyn. "We thought that if we gave the rock more of a stylized look, we could use a lot less geometry. But when I began to digitize the thing, it was just a complete failure. It was just horrible working with the digitizer; it wouldn't read the points at all, or it wouldn't put them together correctly." After struggling with the clay model, the digitizing effort was shelved, and Robyn and Richard went back to the grayscale extrusions.

Scale Models for Actors

Sizing models created by different artists so they link together seamlessly is but one challenge of building a 3D space. It was for this reason that Cyan added another valued team member: Harold (Figure 18). "A long time ago, we had created a very simple mannequin that we would put into our environments to make sure that the scale felt right," said Richard. "We built all of our environments to scale, but in addition to that, just putting a person there — or a computer-generated model of one — really helped out." Harold was soon cloned and his expertise shared among all of the artists. Being a virtual six feet in height, he could be brought on the scene to test the player's line of sight, as well as the scale of the surrounding terrain. With all of the separate objects that were imported into the model to create an environment, the Harold prototype proved to be valuable aid in keeping everyone's work in the same ballpark.

Position Lights, Camera, and Talent for Bluescreen

Live action in 3D computer graphics requires precise positioning of the cameras and lights used to photograph the actors against a bluescreen background. In most cases, you can speed the production process by preparing your models ahead of time and using them to prototype your final composites while on the set and before shooting a single frame. Cyan's designers brought to the stage their Softimage models on an SGI and a Macintosh for video compositing. Having built everything to scale before the shoot, they spent most of the day setting up the lights and camera positions to match those in the computer-generated background — and discovered that the integration would work better than they had imagined. "Because we had built our models to scale and we had lighting objects in the computer," commented Richard later, "we knew where the light sources were: they were six feet off the floor and were spaced exactly five feet apart. We could duplicate that on the set and that worked really well." Using their models for reference, a video frame was placed over a still from the model. Through this iterative prototyping process, the camera position, lens angle, and lighting were adjusted until the effect was seamless.

Prototype Animations

Depending upon your needs, you can prototype animations very cheaply by creating storyboards that can be used by themselves or scanned into a computer for simple animation. If your team is beyond the storyboarding stage, you can mock up your environment in a software package that yields low-resolution walkthroughs,

such as Virtus Walkthrough or the QUAKE level builder Deathmatch Maker. Both of these packages quickly set up simple texture geometry that players can walk through. As your animations become more polished, you can take advantage of some of the simple visualization tools that most 3D packages contain: wireframe animations can be played in real time, and for more detail, flat-shaded animations will render in a fraction of the time required by full versions. You can use these low-resolution images as references for making changes before the final render, which may take many hours or days to complete.

"Softimage's tools are really flexible, and are one of the biggest strengths of that whole package I think," said Richard of the 3D application used to create RIVEN. "A lot of these animations were so complex in terms of the geometry that we knew we were only going to have one shot at fully rendering this thing — it was just going to take so much time. So we really tried to make sure that we had seen it as many times as possible in its various primitive stages, including wireframe and shaded views." For example, even with four SGI servers, the submarine adventures at the bottom of Jungle Island Bay lived in the queue for months because the atmospheric shaders and reflections caused the animations to creak out very slowly, frame by frame.

The fast playback of the wireframe animations allowed RIVEN's designers to adjust the player's experience before committing to a lengthy full render. Rand explained: "Especially for the major animations, we would always render flat-shaded views — or we would render out the wireframes so that they could play at full speed, just so we could see the action and make sure. Even though it's a subtle effect, if you get to the end of that lumber car ride and there is no pause before falling out of the bottom of the car, it just doesn't feel right."

Prototype Navigation: Click-Through or Virtual Bubble?

Early on in Riven's design, the Cyan team prototyped different navigational systems by doing some test renders. These production prototypes compared the click interface style from MYST with several that provided more freedom of movement, such as QuickTime VR. The designers found that while a VR experience more closely resembled moving through a real world, they lost control of how the clues in the environment were revealed to the player. Richard remarked, "Since the presence of other characters in these games is still pretty

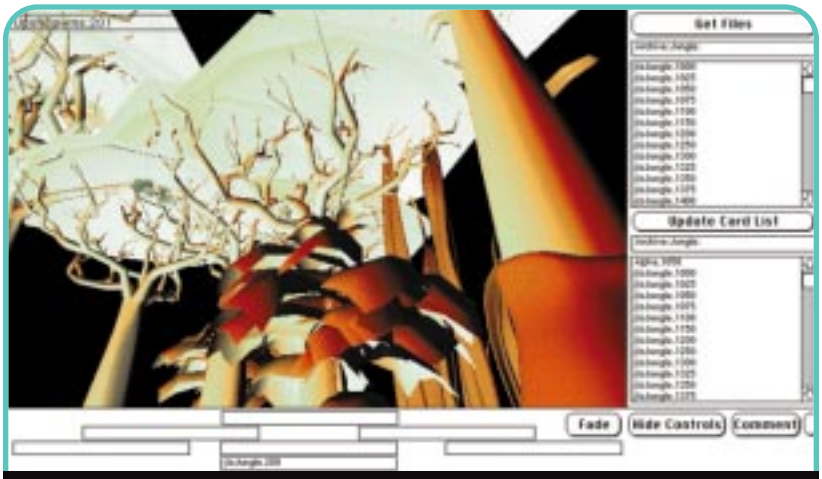


FIGURE 13. *Artists could make changes as they walked through the world, even if that world was a low-resolution one.*

FIGURE 14. *Interactive walkthroughs provided feedback on how the final art would look together.*

FIGURE 15. *Using HyperCard artists could quickly define and modify buttons to fit a particular screen and drag new screens from a list into the fields below the view area.*

slight, the primary relationship is between the participant and the environment. So we tried to imbue that environment with as much meaning as possible to maximize the richness of that relationship between the environment and the participant. If you're just running through the game and not seeing things and not hearing things, then you're cheating yourself."

The step-through interface created a more contemplative pace, encouraging the player to look for clues. "The click-through interface, to a certain extent, slows you down and allows you to digest what you're seeing and what you're hearing better than if you were just running through it at marathon speeds," said Richard. RIVEN's designers wanted players to really look closely at the environment around them to notice the subtleties. By comparing some tests in VR-type environments with the more traditional click-through interface, they decided that a virtual bubble would constrain their game design too much. The primary drawbacks were:

1. It degraded quality of the art (because of compression or rendering on the fly).
2. It led to a loss of control over where the viewer looked (as if one were making a film without edits).
3. It did not support multiple states. (For example, an open and closed door required two different bubbles instead of just two different screens.)
4. The VR interface encouraged the player to speed through the terrain, emphasizing the speed at which they could solve the game's puzzles. (Click-through interaction slowed the viewer down, emphasizing the importance of observing.)

Being Flexible versus Being Finished

Prototypes are guide books rather than detailed road maps. Designs can change frequently, even at the end of the title's development. "The one thing I have to say about the way we did things was that it was a lot of flying by the seat of our pants," said Richard. "The fact is that you can have stuff planned out on paper, but making it real and detailed and complex and functional often necessitates changes and adjustments in the contiguous objects and spaces." No matter the extent of your prototyping efforts, recognize that storyboards or even the "final script" never matches the release. "Flexibility is what allowed us to keep tweaking and twisting and making it better," said Robyn. "At the same time, flexibility is what prevents you from ever being done. So as long as you understand that and can work within those constraints, you can get the best possible production available given the amount of time and money you have."

Prototype to prepare your ideas — seed the creative terrain and make it fertile. Use design prototypes to become familiar with the space, so that when production starts and design decisions lock down one by one, your team will understand the opportunities and share the same vision. ■

THANK YOU

Special thanks to Rand and Robyn Miller, Richard Vander Wende, Bonnie Staub, and Richard Corley for their help, hospitality, and access to RIVEN's prototypes.

FIGURE 16. *This grayscale map of RIVEN's Temple Island shows the higher elevations in lighter colors. The colored areas indicate areas to be kept at the same elevation.*

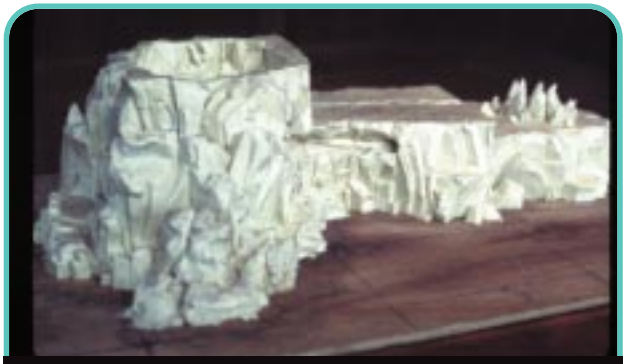


FIGURE 17. *Robyn created a clay model of Gehn's Survey Island in the hopes that it would be easier to digitize than to extrude from grayscale.*

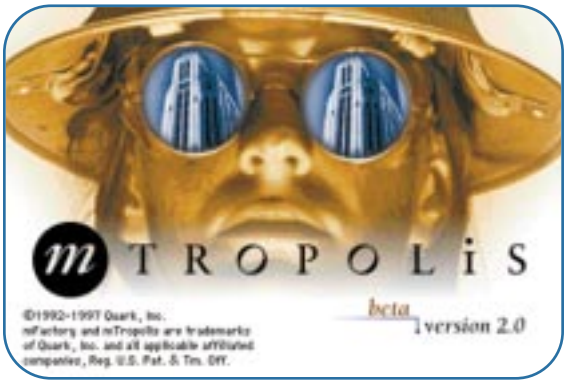
FIGURE 18. *Harold, Cyan's cyber stunt person, (whose head and shoulder can be seen in the lower right corner) provided reference for the scale of models and camera placement. How big were objects compared to Harold? What could Harold see from different places along the path?*

Pigs in mTropolis 2.0

by Tzvi Freeman

Authoring environments that provide visual representations of a program structure that are more conducive to a creative state of mind can make a dramatic difference in enhancing your creativity. Years back, I remember sitting my three-year-old son on my lap while at the computer, showing him what I had just made using HyperCard and asking, "What would you like now?" I implemented his demands then and there. The results were way beyond what I could have imagined on my own. Apparently, Craig Hickman built Kid Pix the same way. Of course, eventually we had to rewrite everything in a *real* programming language, but for project take-off, HyperCard was fine.

More recently, fun-oriented development has become a nearer reality with the advent of mFactory's mTropolis, now under the wing of Quark. Note that I haven't said the fun has arrived. But mTropolis certainly gives us a window on what the future will bring.



Enter mTropolis

As a beta site for mFactory, I've been working with mTropolis since it's alpha stages five years ago. I've used it to create several major prototypes and lots of experiments and toys. mTropolis has an attractive feature list, including a true object-based structure with its own extendible object model (the mFactory object model, or MOM). It compiles impressively efficient code. The graphical drag-and-drop interface provides a clear abstraction of what's going on, at least in general terms. It's cross platform. It has convenient ways to manage and relink your resources.

But the most empowering feature is something called BOP. That's how visionary Norm Gudmundson, the creator of mTropolis and chief engineer of mFactory, refers to his paradigm for software development, Behavior-Oriented Programming.

BOP!

For a better concept of what Behavior-Oriented Programming provides and how to take advantage of it, I'd like to use a parallel to our concept of the universe. At one time, everyone thought of the solar system, and of the universe, as having one absolute center. Whether that was the earth, the sun, or somewhere else, was subject to debate.

But this absolute center could only be in one place.

Then Albert Einstein came along and proposed a new way of looking at things. Now, as his students have explained, anywhere that you wish could be the center. You just need to create an appropriate model.

Similarly, procedural code is all about getting from a beginning to an end. The procedure that takes you there is the absolute center of everything, branching out in routines and subroutines, but all in one path with one vortex.

Object code begins to break out of that paradigm, but it reaches it fruition with BOP. Here, every time you work with any of your objects, you must see all of the rest of the world that you've created from the perspective of that object. How will that object react to the rest of its world? What will it do with the information that it receives from "out there"? What will it tell its world about its own internal state? In fact, every object becomes the center of its universe.

Reusability is the most salient advantage of BOP. A BOP programmer ensures that all of his objects are as self-sufficient as possible and avoids being context-specific at all costs. He does this principally by designing his objects for as generic an environment as is practical and by establishing consistent protocols in all his worlds. An object that sends messages that mean one thing in one world and something else entirely in another is not very valuable. A valuable object operates similarly and provides meaningful data in a wide variety of contexts.

Adaptability is another point chalker for objects. A "rigid" object, such as a scroll bar that remains the same size in every environment, is a rather useless asset. A good object, then, receives messages about its environment and knows what to do with them. A good world, it follows, is one that provides its objects with meaningful information.

Complicated objects that cannot be broken into smaller, self-integrated components are a pain — a pain to work with and a pain to debug. A BOP programmer starts off by acquiring many small and simple self-reliant objects. Once he has a rich library of those basic components, he can carefully build them into larger "organisms" that have more specific functions. If he builds them well, another programmer will be able to use them in his worlds without having to know much about their inner contents.

BOP is a realistic simulation of the real world. Imagine how your world would be if it had been made by conventional programmers. From a passive viewpoint, everything might look O.K. But then, let's say you grab a bucket, walk over to a pond, and lift out some water. Within the context of the bucket, the water would probably behave entirely differently than it did in the well. Bring the bucket into your kitchen, and it may just suffer a fatal error and crash your entire reality.

We generally see our world as being made of many pieces, each having their own properties and interacting with each other in specific ways. BOP means thinking about your project in the same way.

True BOP mastery still eludes me, but in many instances, I've managed to create a wonderful and valuable animated object with multiple behaviors in one project, dragged and dropped it into an entirely different project, and after a few minor tweaks, have it do all the same tricks in an appropriate mode for its new environment. mTropolis 2.0 has made some important fixes that make such moves more reliable.

Rocket BOP

This is not a tutorial, just a loose description. I'm using a Power Macintosh with mTropolis 2.0. I've just opened a new project window. That three year old mentioned above, who acted as a lap consultant on my first children's game, is

mFactory's Official Position on 3D

I asked Norm Gudmundson, creator of mTropolis and chief engineer at mFactory, what he was planning to do about 3D in mTropolis. Here's the e-mail response:

"mFactory has always been very interested in integrating 3D technology into mTropolis. Unfortunately, when we tried that a couple of years ago, there was no standard 3D technology to build a successful twitch 3D game (in my opinion at least). This was because most successful twitch 3D games use a proprietary 3D engine or a heavily modified standard technology designed specifically for the game. It's my feeling that 3D twitch games will always exist on the very edge of the technology wave, just beyond where mFactory can reach.

"Does this mean that mTropolis will not incorporate 3D technology? No way! I think real-time 3D is strategically important to all 'software solution composers' such as mTropolis. However, the objective of 3D in mTropolis will be to stress real-time 3D simulation and possibly prototyping 3D games, not to act as a 3D twitch game development environment. I think we will all have to leave that to the whiz kids hacking away in the back rooms. At least for a while yet.

"With real time 3D simulation integrated into mTropolis, BOP could really take off. After all, the whole point of BOP is to break down software problems into the fundamental components we find in the "real" world. I can't think of a paradigm that mates as perfectly with real-time 3D as BOP, and since mTropolis is the only environment that uses BOP so far, the marriage just has to be made."

I also asked Norm if he thought that those "on the edge" 3D programmers would be able to extend mTropolis's future 3D capabilities using MOM. He was honest and answered that he can't know until the alpha and beta phases of 3D integration. But, "mTropolis has always stressed the need to extend its supported features and 3D would be no exception."

—Tzvi Freeman

Tzvi Freeman teaches Multimedia and Game Design at the University of British Columbia Multimedia Certificate Program. He has designed several commercial games and acted as a consultant on many more. He is interested in hearing your comments at tzviF@aol.com.

FIGURE 1A. *The Layout Window in edit mode...*

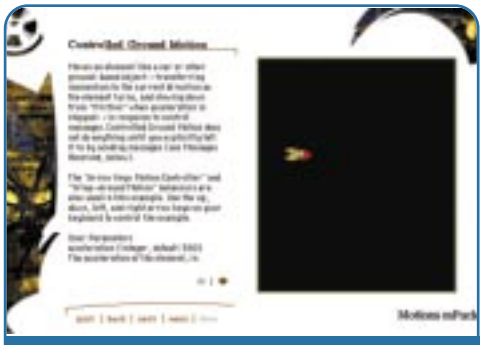


FIGURE 1B. *...and in run-time mode.*

now ten. He falls into my home office at 3:20PM, looking for what's cool today. Great, just in time to help with this article.

When asked, "What do you want to create today?" his response is a typical 10 year old's unintelligible, hyper-babble, which I interpret as, "I feel like making things blow up."

My son reminds me that we have a wonderful outer-space-object exploding animation that works well with a small explosion sound effect he created. I have that object exploding in another project — all the necessary behaviors are already set. So I open that project, find that object, and drag and drop it into my new project. Now all it needs is something to send it a message that says, "**explodeInSpace**," and it'll go up in solar luminance, leaving behind a burnt crust.

I need something to attack the space object. The CD that mTropolis comes on contains a few libraries of ready-made behaviors, as well as some objects that use them. There's a rocket in one library that's propelled by the arrow keys. It also has momentum and drag built in.

I drag and drop the rocket into my project (Figures 1A and 1B). Then, I drag a small icon from the Modifier Palette — this one is called the Collision Modifier — configuring it with a few mouse clicks (Figure 2). I set the rocket object's behavior so that whenever it hits another object for the first time, it will tell that object **explodeInSpace** (Figure 3).

I've just dragged two objects out of two entirely different projects authored by two different people who are in no way coordinated with each other. If they are well-built objects that follow all the BOP guidelines, they will work together and we'll have an explosion on the screen.

Hitting one stationary object in open space doesn't quite compete with the neuro-traffic in my kid-consultant's brain. I dig up a neat random-motion behavior to drop into our space-object, setting it to terminate when the **explodeInSpace** message arrives (Figure 4).

For even more excitement, I repeatedly [Option]-drag the object to make multiple copies. Since the behaviors are all "aliased," I can still modify the behavior of all of them by modifying any one of them. But since the variable that sets velocity is specific to each object, I allow the consultant to adjust each one to a different setting. He also adjusts size and color for each object, no typing necessary. Total time to construct this prototype: approximately 15 minutes.

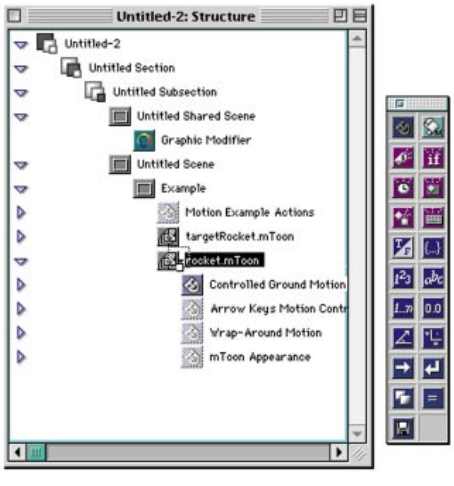


FIGURE 2. *Dropping a new Behavior Modifier from the Modifier Palette into my rocket.*

BOPping the Head Against the Wall

As can any technology, BOP can be quite frustrating. The good news is that most of the hassles are things that have visible solutions to them. Here are a few of my major gripes:

The first two are problems with the scriptor interface. It's wonderful, especially for chronic typo-slobs such as myself, to be able to just drag, drop, and click an entire program together. But the convenience — at least in its present incarnation — doesn't come without severe compromise. For just one example, double-clicking, typing, then closing 20 icons to adjust 20 variables can get kind of tedious — apart from the fact that you can't view them all as one list. Of course, you can use a List Modifier to set them all, but that was implemented as an awkward afterthought, and it's very clumsy to use.

The Structure View provides a kind of outline of the entire program, similar to an indented outline that you might use for an abstract of an article you'd like to write, or the contents of a book. Establishing good naming conventions for yourself helps make this view all the more meaningful. The problem is that there's only so much that you can see on a single screen. In a long or complex project, the script view resembles watching a football game through a tiny crack in the fence. mFactory could easily alleviate matters by allowing multiple views of the same window, and at the very least, providing printing capabilities. A print out could provide detailed information that's not available in the structure window.

Speaking of reviewing your script, whatever happened to programmer's comments? The only place for these are in the miniscript modifiers. If the Structure View is meant as a window on your project's architecture, then why can't I annotate it?

A Behavior Modifier has its own view — quite a clever one. I won't bother with all the details here, but the window is too easy to clutter and, again, has no place for comments other than in miniscript mod-

FIGURE 3. *Selecting a message for the Collision Modifier to send.*

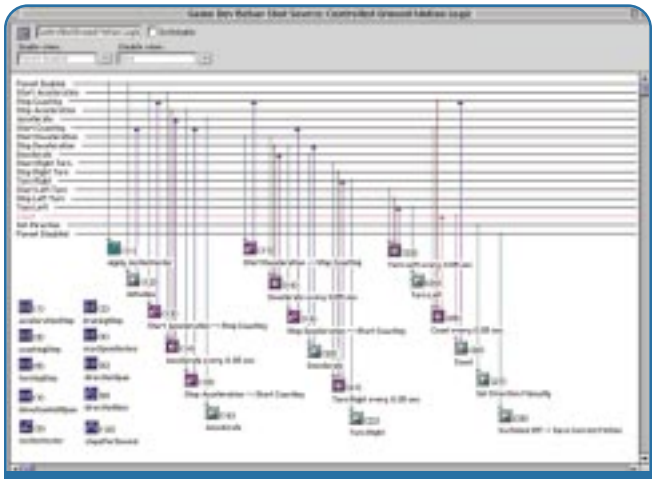


FIGURE 4. *The Behavior Window for the motion logic of the rocket. By selecting the Coast message track, any modifiers associated with that message are easily distinguished.*

ifiers. Going into a complex behavior created by even the best mTropolis designer and trying to adapt it to your project can be a nightmare. Yet that capability should be one of the salient features of the tool.

As far as problems inherent to the concept of BOP, they fall into two categories: issues associated with stereotyped behavior, and the extreme difficulties that are loaded into the programmer's brain in conceiving all the dynamics of his creation.

Consider my rocket zooming about telling everything it hits to **explodeInSpace**. That's fine when there's only a handful of objects and most of them are responsive to that message. But, just as in our reality, not everyone wants to hear you say, "Have a good day!" every time you bump into them. In most games, such a behavior could have undesirable consequences. For one thing, the CPU is overloaded with processing this message and all the instances of it as it travels down the parent-child hierarchy-tree.

One solution is to make your messages more discerning and specific. This message's transmitter could get some information about an object before engaging it in conversation. This information would help the transmitter determine whether the object might be responsive or not. But then, you're just trading off one overhead for another.

mTropolis allows you to limit the range of a message's transmission. You can send it to the whole project, to an object's parent, to siblings, or only to a specific object. You can even stipulate that this message doesn't "cascade" down the hierarchy. In my experience, however, these solutions severely compromise the reusability of your objects. And reusability is what BOP is all about.

Taking another cue from real life, the solution to problems generated by stereotyped behavior is intelligence — some sort of AI. An object has to be able to learn its new environment, to adapt to become as efficient as possible there, and to store all that data separately from its basic behavior data, so that it won't necessarily apply it to other environments.

BrainBOP Overload

BOP, as it exists now, is not meant for dummies. It excels when the task is to build something simple using a few prebuilt objects and some of the more straightforward modifiers. But as soon as your project has reached any level of complexity, it can become a major brain-stretcher. As mentioned earlier, you're no longer looking at things from one point of view. You're forced to see things from the perspective of each of your objects and then imagine how all of them will relate to each other.

mFactory is working hard on their debugging tools. Progress with naming conventions will also help. In the meantime, mTropolis is not scriptor-proof.

Is mTropolis a tool for nonprogrammers, such as animators, writers, and others? The fact is, there are plenty of artists using the tool right now, quite successfully. Not having to memorize a set of commands and syntax helps. Most of the learning curve is just in twisting your head around to think in new ways. Nevertheless, it helps to have a code hacker around to create extensions when you need them, or to deal with the mathematics and algorithm matters.

For generating good ideas and trying out new behaviors and interdynamics of elements from past projects, nothing beats mTropolis. But beware, your project will be molded by mTropolis's limitations and demands, as any good programmer always falls for the most elegant solutions within whatever environment he's working.

If you're new to the game development scene and don't yet have an engine all your own, mTropolis may be the place to start. If you have an engine that you're satisfied with already, you might consider fusing it with mTropolis using MOM. Or just use mTropolis as a prototyping tool. ■

mTropolis 2.0

mFactory/Quark
Burlingame, Calif.
650-548-0600
www.mFactory.com

Price: $995 estimated street price for version 1.1. Free upgrade to version 2.0 for new 1.1 purchasers.

Systems Requirements: The minimum configuration recommended for authoring in mTropolis is a 68040 or Power Macintosh with
- 6MB RAM (for 8-bit color) or 8MB RAM (for 24-bit color)
- System 7.x
- QuickTime 2.5
- Sound Manager 3.2

The minimum configuration recommended for playback is a 68030/68040/Power Macintosh or 486/Pentium PC with Windows 3.1/95/NT and 2MB RAM.

Peeki g Th gh P al

by Adrian Perez

In the few eons of cyber-time since BATTLEZONE, it seems that computers have finally become fast enough to handle complex 3D geometry that approaches what we see in the real world. Every new QUAKE/DUKE/TOMB RAIDER engine seems to have more and more polygons in an average scene. Rasterization speed in particular has gone through the roof in the last few years. With installed 3D accelerator cards in the millions and many computers this past holiday season shipping with them preinstalled, a high fill rate doesn't seem to be as much of a pressing problem today. Many developers are likely breathing a sigh of relief — at least those who hate trying to squeeze cycles out by tweaking assembly routines.

However, a perfect world does not a high fill rate make. As worlds become more complex and game players kick and scream for more features, a much bigger problem is beginning to arise: hidden-surface removal. Effectively managing large data sets of polygons, culling the polygons that are completely out of view, clipping the polygons that are partially out of view, and drawing the polygons that are in view is no simple task. Also, while game developers generally agree on the "best" way to draw texture-mapped or Gourad-shaded triangles, hidden-surface removal is still an area of great research. There really is no best way to do it, only a number of different techniques that have their own strengths and weaknesses.

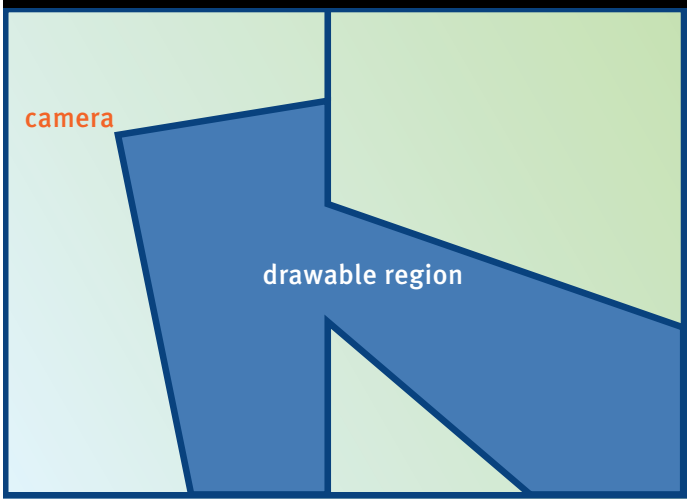
Binary space partition trees (BSPs), which have been used in game development with increasing frequency lately, have a lot to offer. For the sake of discussion, a BSP tree is a binary tree that divides an n-dimensional space at each node with an (n-1)-dimensional hyperplane until the world is divided into a set of convex subspaces (at which point they can no longer be divided). Most importantly, BSPs give you a perfectly sorted order from front to back or back to front of any scene with an astonishingly O(n). This means that as the number of polygons (n) increases, the processing time (represented with an O) increases linearly with n. Straight polygon sorting, on the other hand, is represented as $O(n^2)$, meaning that this method gets much slower as the polygon count increases. The drawback to this technique is a one-time, relatively time-consuming BSP tree-building phase. BSP trees are a complex and abstract topic in and of themselves. A more complete description than I could include here can be found in *Computer Graphics: Principles and Practice* by Foley, van Dam, et al. (Addison Wesley, 1996).

However, BSPs are not a cure-all. Constructing a BSP tree is time consuming, so much so that it can only

Adrian Perez attends Carnegie Mellon University, majoring in Computer Science with a focus on graphics and artificial intelligence and a minor in Mathematics. He is currently working in his spare time on a hybrid BSP/portal engine he calls Zeta. You can contact him via email at amperez+@andrew.cmu.edu.

FIGURE 1. *Sliding doors in DOOM is one way of working around the restrictions associated with a static data set.*

FIGURE 2. *Portal rendering techniques allow you to draw only pixels that are visible from the camera's viewpoint.*



realistically be done before executing the game, which completely restricts you from modifying the data set in any way apart from limited workarounds (such as the doors that slide up and down in DOOM, shown in Figure 1). Static worlds are fine, but eventually they'll cramp designers, and players will demand more. Also, BSPs don't have occlusion built in (I think that they do support this feature, but the important thing is that nobody knows how to do it yet).

For example, imagine you're drawing the inside of a house. You generally won't be able to see any polygons upstairs from any point downstairs besides the stairwell. However BSPs cannot cull those upstairs polygons automatically. DOOM got around this by having a front-to-back buffer — every scanline was drawn once and only once, and it traversed back until nothing else could be drawn. Although amazingly effective, this method translates very poorly to 3D. Edge sorting took care of the overdraw in QUAKE, but no edge sorter in the world can handle all the polygons for every frame of a typical 3D world at any reasonable speed. So id incorporated a potentially visible set (PVS), where the visibility at each node is precomputed and stored, so at run time only a small list of nodes needs to be processed. In practice, the PVS is not a BSP technique. It's really a concept belonging to portal rendering.

Portal Rendering

Portal rendering is an alternate way to treat 3D scenes. It's been around since 1971 and has been (or will be) used in games such as DESCENT, DUKE NUKEM 3D, UNREAL, and PREY. It operates under a surprisingly simple premise: if I'm drawing two rooms that are connected through a doorway, after drawing the room that I'm currently in, the only part of the screen that hasn't been drawn yet is the area that can be seen through the doorway; if I restrict the drawing space to the dimensions of the doorway, I'll only draw those pixels of the next room that I can actually see, giving me zero overdraw (Figure 2). The world is represented as a collection of convex hulls, connected with polygons that are tagged as portals. Convex hulls are collections of polygons that have no concavities. You can draw a line from any point inside the hull to any other point inside the hull and, by definition, that line will never leave the dimensions of the hull. Cubes, for example, are convex hulls that can be considerably deformed before they become concave (DESCENT used a connected cube scheme to accelerate portal rendering). To draw a scene, you start with a base portal with the dimensions of the screen (or the area that you're drawing) and draw the current hull you are in. Every time you encounter a polygon that's tagged as a portal, you recurse into the next cell, clipping the new portal against the current portal until you cannot see any new portals.

Portals work remarkably well when the conditions are right. The Build engine (used in DUKE NUKEM 3D, REDNECK RAMPAGE, BLOOD, SHADOW WARRIOR, and others), for example, uses 2D portals to a great effect. Instead of having a front-to-back buffer as DOOM has, Build games draw convex sectors, clipping the scene to portals as it recurses backwards. The Build engine achieves moving sectors by performing virtually no precalculation on the scene. It is thus dynamic: doors can swing out, gears can spin around, and walls can explode. In addition, as a pure portal system, the Build engine allows floors above floors, another effect that wasn't possible in a straight BSP system such as DOOM.

DESCENT uses portals to reduce overdraw to zero and to accelerate culling (Figure 3). A DESCENT level is a collection of convex six-sided polytopes (3D polygonal objects). Each polytope is connected on each side with, at most, one other polytope. To draw the scene, you simply recurse and clip the portals to each other as you step down the list of cubes. Both DOOM and DESCENT run on fairly low-end machines, yet feature an amazing amount of eye-candy for the horsepower.

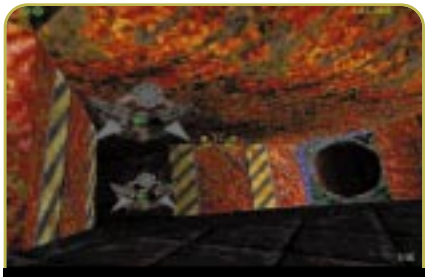


FIGURE 3. *DESCENT uses portals to reduce overdraw to zero and to accelerate culling*

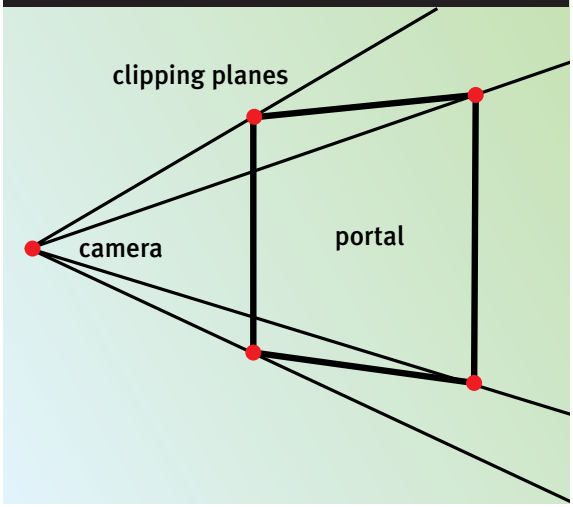


FIGURE 4. *Each portal creates a new set of clipping planes based on the camera location and the locations of the portals vertices.*

Implementing Portal Rendering

To implement a portal-based engine, you have several options. The most fundamental design decision is whether you want to use 3D clipping or 2D clipping, each of which has its advantages and disadvantages. 2D clipping is typically done on a per-scanline basis, and since most triangle raster code has built-in clipping support, this isn't a very tall order to fill.

Start drawing your image with a table of extents, so that each scanline has a start, end, and active field. As you reach portals, modify the table, clipping each scanline and calculating the new extents to determine if the current scanline is covered by the polygon. At this point, drawing the polygon is simply a matter of drawing only the active scanlines and only the appropriate pixels in those scanlines.

The good news about this method is that drawing polygons requires only a few lines of logic code within the function, as well as a couple of **if** statements in the code for each scanline. The bad news is that this method translates very poorly to 3D hardware, which generally draws a triangle at a time, rather than scanlines. Also, although a few extra **if** comparisons may not seem like much, they tend to stack up when you're rendering at 640×480 resolution and clipping in three dimensions rather than two — especially when you're drawing large sets of polygons. In practice, portal rendering using 3D clipping may be faster, especially as resolution, portal count, and polygon count increase.

When implenting 3D clipping, start with a clip volume that consists of the area visible on the screen. When you encounter a portal, clip the polygon to the 3D volume and create a new set of planes based on the two vertices of each edge and the camera location (Figure 4). The good news is that this isn't much more computationally demanding than traditional 3D clipping, although it requires a little more time to generate the new frustums. The problem with this method is that you're clipping more often, and thus, you have to perform a few **sqrts()** to normalize the new planes when you clip them to the portal and generate the new frustum. Each **sqrts()** requires about 100 cycles each — not too good for performance.

This is the method that the included engine (available on the *Game Developer* web site) uses. The file it loads has polygon information for all eight cells (four rooms and four doorways), and all polygons have an ID tag. This tag is -1 for nonportal polygons (the walls, the ceilings, and the floors). For the clear portal polygons that represent the doorways, the tag contains an index into an array, indicating which cell in the data set lies on the other side of the portal.

FIGURE 5. *A view of the Device Independent Bitmap from the engine that accompanies this article.*

Here's how it works. At each frame, the camera is moved according to user input. The camera is bounds-checked to make sure it's in a cell and the index of the cell is noted. Then a new frustum is generated. Finally, a single call is made to draw the current cell with the current frustum. The function automatically clips the frustum to any portals and recurses as needed. It draws the view into a Device Independent Bitmap (DIB) and then blits to the screen (Figure 5).

It's important to note that the initial clip volume only has four planes. A far clip plane is unnecessary because drawing will stop before any appreciable distance. A near clipping plane is not recommended when using this rendering method, because as you get very near a doorway, the portal itself becomes clipped and the next room is

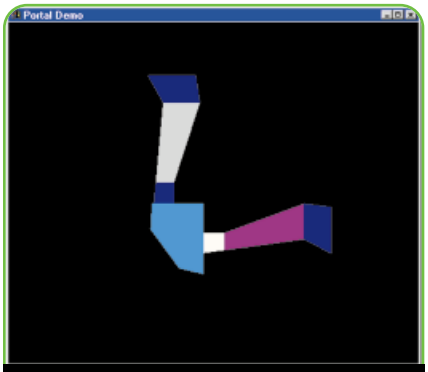


FIGURE 6. *An overhead view of which polygons are drawn and which are culled in this article's sample application.*

FIGURE 7. *The QUAKE engine does not implement portal rendering because each level contains a prohibitive number of portals.*

not drawn until you enter it. When I created the engine that accompanies this article, I kept it restricted to flat shading and didn't use any lighting or mirrors, which cut down the code complexity. There is also an overhead view (Figure 6) so you can see the polygons being dynamically clipped as the camera moves around.

The Drawback of Using Portals

Just as BSPs (and, in fact, as any hidden-surface removal algorithm), portal rendering has its faults. Clipping portals is not a computationally simple thing to do, and in most portal rendering systems, it ends up being the limiting factor. This is why, for example, DESCENT slows down considerably in large, complex rooms: the clipping of literally hundreds of portals bogs down the game. John Carmack once told me that he had tried implementing portal rendering in the QUAKE engine, but found the overhead from clipping portals to be prohibitive. He said that in a typical QUAKE level, there would be literally twice as many portals as polygons (Figure 7) — the geometry alone would kill you.

Implementing Good Portal Effects

However, portals are not without their perks. You can achieve an amazing amount of special effects with portals, effects that are difficult (if not impossible) with other hidden-surface removal methods.

LIGHTING. Real-time shadows are always difficult to accomplish in 3D graphics, but portal rendering offers a very elegant way to do it. For every light in the scene in your general area, perform a portal render using the light as the virtual camera location and its spotlight as the initial frustum. When that's done, you have a set of polygons visible from the light. Tag those polygons that were trivially accepted as lighted and those that were trivially rejected as unlighted. Make note of any split polygons and store this split information in the global polygon list (perhaps using pointers in the polygon structure to a temporary polygon pool that stores the lighted/unlighted members of split polygons). Perform these actions for every light. Then, when the scene is drawn, you proceed as normal, using the lighted/unlighted/split information for drawing. When a conservative number of lights is used in a full-portal environment, you can implement moving, turning spotlights that light the scene in real-time.

MIRRORS. A real-time mirror is another feature with which portals can help. Some portals can be tagged as mirrors, which means that instead of recursing into the cell on the opposite side, we recurse again into the original cell, using mirrored camera and frustum information. To mirror the camera, use a dot product of the camera location and the plane's normal to find the distance from the camera to the origin (Figure 8). You subtract from this the distance between the plane and the origin, which gives you the distance from the camera to the plane. Then double this value (to get the distance from the camera to a point on the opposite side of the plane that is the same distance from it), multiply it by the normal (giving you a vector that goes from the cam-

FIGURE 8. *To mirror the camera, use a dot product of the camera location and the plane's normal to find the distance from the camera to the origin.*

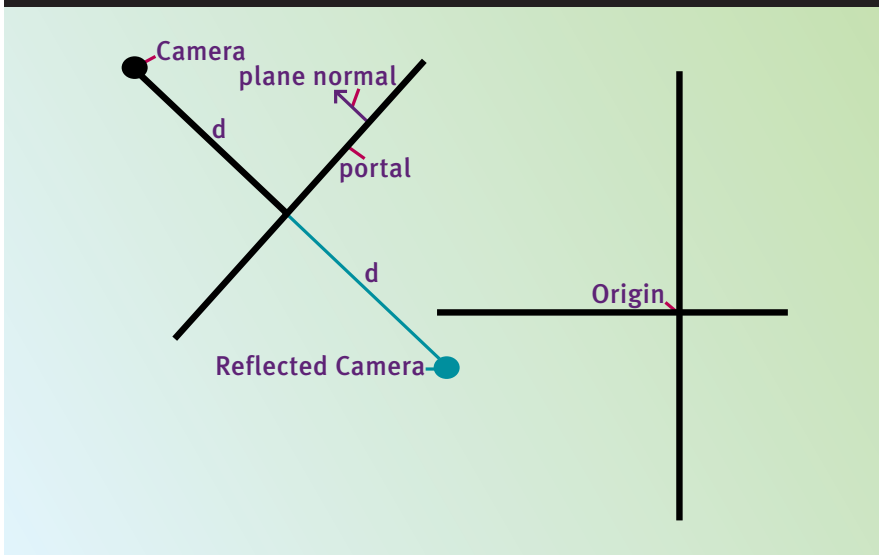
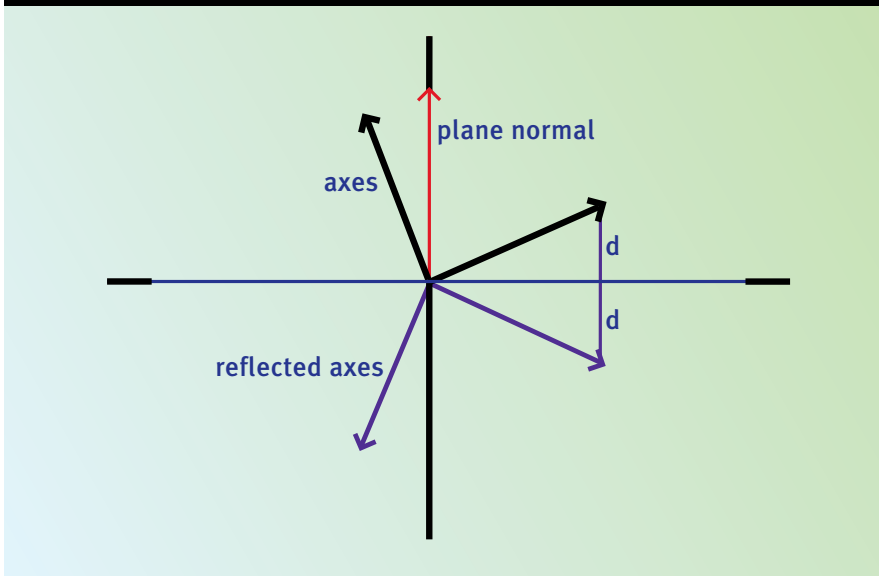


FIGURE 9. *To reflect the camera axis, treat the plane normal and all normals of the axis as if they were coming out of the origin.*



era to the reflected camera), and add the displacement to the camera vector.

To reflect the camera axis, you use a similar procedure. The most notable difference is that you treat the plane normal and all normals of the axis as if they were coming out of the origin (Figure 9). You find the distance from each vector to the plane, double it, multiply it by the plane normal, and displace it. Caution must be used here, especially when two mirrors can see each other, as an infinite loop can occur. One way to deal with this without losing much visual accuracy is to treat the mirrors as imperfect, slightly darkening the image on each reflection and ending recursion after enough reflections make whatever you draw black. You can reflect lights off of mirrors with the same approach, creating some incredible special effects.

Displacing Portals

You can displace portals using many of the same ideas that I've discussed with regard to reflecting portals. Instead of just reflecting the camera about a certain plane, you move it to a completely new location. You can try this for yourself in DUKE NUKEM 3D. Bring up the automap, zoom all the way out, and then jump into some water. You'll see yourself jump to a different corner of the map as soon as you enter the water. Portals such as this can be used for closed circuit TV displays, for quickly jumping to different areas of the map, or for an eerie floating doorway that leads into a completely different room. If the source and destination portals are parallel, you can simply displace the camera by their difference when you traverse the portal in the drawing sequence. Nonparallel portals are a bit trickier, requiring more matrix math than I have space to discuss.

A lot of buzz is in the air regarding half-portal techniques. QUAKE 2, for example, uses what John Carmack calls "area portals" to help with hidden-surface removal. In this technique, levels are sectioned off into areas connected by doors, and the parts on the other side of closed doors are not considered by the engine for drawing. As soon as the door opens, a flag is raised, signaling that the far area is drawable.

Learn Something New

Whether you're programming games, Internet security systems, or ugly scientific visualization programs, you should focus on good (meaning abstract and hard to visualize) algorithms. There's no "right" way to do anything, merely a bunch of good ways (and a whole lot more bad ways). As the programmer, it's your responsibility to look at what you want to accomplish and to choose the best algorithm to reach your goal. To do this, you need to know about as many algorithms as possible, including their strengths and weaknesses. You need to be able to maximize the good and minimize the bad by tinkering with these algorithms. Portal rendering has a lot of very good points and a lot of not so good points, and it deserves consideration when you're analyzing possible hidden-surface removal techniques. ■

FOR FURTHER INFO

Visibility Computations in Densely Occluded Polyhedral Environments by Seth Jared Teller, University of California at Berkeley. Available at http://sunsite.berkeley.edu/NCSTRL/

Database and Display Algorithms for Interactive Visualization of Architectural Models by Thomas Funkhouser, University of California at Berkeley. Available at http://sunsite.berkeley.edu/NCSTRL/

3D Engines List. Available at http://cg.cs.tu-berlin.de/~ki/engines.html. The list has information on many types of 3D engines (including mine). Some of these are based on portals.

Homepage of Jacco Bikker (aka The Phantom) at http://www.geocities.com/CapeCanaveral/5402/index.htm. This page has the source code to a scanline-clipping portal rendering engine and some information on how he built it.

Homepage of Mark Feldman at http://www.geocities.com/SiliconValley/2151/. This is the home of the in-development Win95GPE, which has documentation about (among other things) portal rendering, DIB programming, and Direct3D 5 Immediate Mode.

Ma im m De h: Ki e i 3D S di MAX 2

by Josh White

Because many of us game artists will see more of 3D Studio MAX 2 in 1998 than we will see of our spouses, we'd better take a really hard look at the upgrade before we commit. As with anything else on which we spend lots of time and money, we need fair, honest opinions that are closely based on first-hand experience. My own work focuses on low-polygon, real-time 3D modeling and animation, not pre-rendered work. For example, we RT3D artists deliver to our clients 3D models (such as VRML .WRL files), not .AVIs. So new rendering effects such as lens flare don't rock my world. I love things such as better interfaces, vertex editing improvements, and really good import/export options.

Regardless of how you have or will use MAX 1, you may wonder if the $795 upgrade for MAX 2 is worth it. The quick answer is that if MAX is your main 3D tool and you're not poor, you should buy the new version. This upgrade is a big one — the scripting language alone will be worth the money. And there are plenty of other major reasons to upgrade.

Let's take a look at it: Figure 1 is the obligatory screen capture. Compared to MAX 1, it hasn't changed that much. But despite the similar look, MAX 2 has a whole lot of changes, some of them very substantial improvements. Many small, annoying problems in MAX 1 have been fixed in MAX 2. Kinetix has hyped this point heavily, and the company is right to do so. Everywhere you turn, existing features are easier to use and more powerful. For example, the align tool now works on subobjects; that means you can align vertices now. I'd be curious to know how many bugs and features improvements were implemented for MAX 2; I bet it's in the thousands. For me, these many small fixes make up one of the most compelling reasons to upgrade.

Most of the major improvements and rebuilds fit into a few general categories: modeling; materials, rendering, and animation; interface; display improvements; documentation and support; MAXScript; and miscellaneous. For each category, I'll highlight a few of my favorite features, discuss the ones that I've actually used in detail, and list others that I think are interesting but didn't actually use.

Josh White (josh@vectorg.com) has been building real-time 3D models for games since 1990. He runs Vector Graphics, a company devoted to real-time 3D; cofounded the CGA, a community of computer game artists (www.vectorg.com/cga); wrote Designing 3D Graphics, *the first book on real-time 3D modeling; is involved with the CGDC; and writes about computer artists… but he admits that all this really just kills time between soccer games.*

Modeling

One of my favorite improvements is MAX 2's more holistic approach to editing model vertices, edges, and faces. This is big news for real-time modelers. Editing vertices in MAX 1 felt as if I was tinkering under the hood of the model where I didn't belong. I had

FIGURE 1. *MAX 2 compared to MAX 1: evolution not revolution.*

to keep the stack collapsed all the time or I'd get spurious problems. MAX 2's interface for subobject editing is cleaner and easier to use, though I haven't used it enough to be reassured that the stack problems have been addressed.

The addition of NURBS modeling is the other big news. Many of the MAX users that I talked to were very excited about this feature. After playing with it, I think it's a pretty nifty improvement as well. You can update the NURBS surface in real time, watching the surface change on the screen as you move a control point or curve. Unfortunately, you can't create holes in a NURBS surface, and I found parts of the interface clunky (Figure 2).

The manuals have some good background on NURBS modeling, but conspicuously lack any detailed how-to information for basic NURBS procedures in MAX. I didn't see any printed tutorials on NURBS either. Regardless, I stumbled through the procedure, and once I had a NURBS object on the screen, things got easier. I especially like the Refine tool: as you move the icon toward the control lattice, new vertices magically appear, just where you hoped they would. At least, that's how it happened for me when I played with it; I have yet to build anything really complicated with NURBS.

I also like the new Preserve modifier, which constrains a mesh to a certain volume. It can also constrain the curvature of a surface by enforcing minimum angles between face normals (in much the same way that Autoedge does).

Shape modeling has several excellent improvements, including Slice, which generates a planar cross-section shape of any 3D object. This is really handy for merging objects. Normally, I use the Boolean with a box to get a cross-section, but that's problematic. The text shape generator also sports some really good improvements, such as kerning, multiline text, and so on.

UV mapping has a handy new feature: the Unwrap UVW modifier allows you to move the mapping coordinates individually in UV space (Figure 3). This unwrapping concept is common in 3D paint programs, but it's nice to have it integrated into MAX 2. The interface is simple: you "unwrap" the mesh onto the texture, then just move the points around and watch the texture mapping update on the 3D object. UnWrap UVW lets you fix mapping that needs to be carefully aligned to the underlying mesh — an eyebrow, for example. I had some difficulty selecting coincident-mapped vertices, such as the vertices at the top of the cylinder shown in Figure 3. I'd like to be able to select 3D vertices along with the vertices on the unwrapped texture.

Materials, Rendering, and Animation

The materials editor got a fairly major rebuild in MAX 2. Many new features abound. One feature that I think is great for prerendered work is cropping — the ability to use part of a larger bitmap as a texture. For example, if you have an image with stripes on one half, you can define a region in the striped area and assign only that region as a texture. Previously, of course, you had to create a separate bitmap.

Cropping is useful for a technique that I call texture clumping: grouping textures into sizes and shapes that are friendly to real-time graphics engines. Many graphics cards and rendering engines require square textures that are 8, 16, 32, 64, 128, 256, or 512 pixels on a side. This is an annoyance for RT3D artists. We're always trying to squish nonsquare textures onto square surfaces, no simple matter when you have a long, thin texture. One approach is to "clump" several long, thin textures into a single image, then use clever mapping to get a particular part of the texture onto the model.

At first, I thought that was the purpose of the cropping feature in MAX 2 — but it isn't. The UVs on the mapped object aren't modified when you change the cropping settings in the material editor (or at least the UVs that are exported in the VRML output aren't modified). I'm guessing it saves a "transform" for UV coordinates on that material. That works fine for prerendered work, but it's ignored when MAX 2 exports VRML.

A lot of the work that was done to develop good rendering tools for the original release on MAX 1 was carried through to its next logical step in MAX 2. Some of it worked well, some not so well. For instance, MAX 2 offers space for more rendering sample windows — up to 24, with lots more options including custom background and object selection. Raytraced reflections and refractions are here — a welcome addition to this release. MAX 2 allows selective raytracing (per-object), which could really save time. MAX's particle systems improved a lot. Users can now

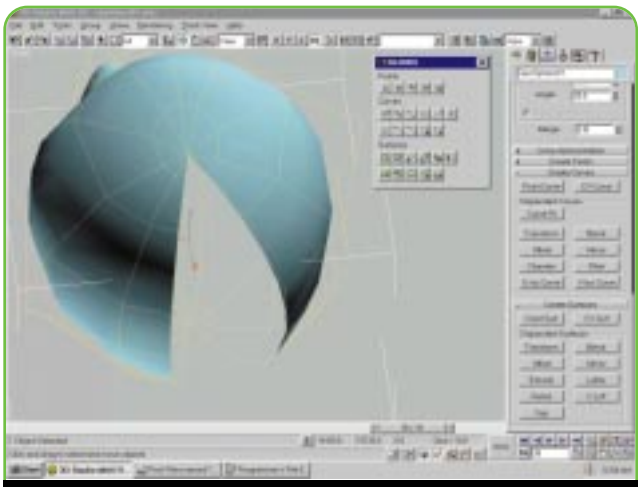


FIGURE 2. *The elusive modeless NURBS dialog.*

use arbitrary objects for the particle, including meta-particles — blobby particles for flow simulations. Particles also behave according to the new physics system, bouncing off surfaces and so on. There's support for Photoshop and Premier filters, which is kind of a no-brainer, but you have to appreciate the inclusion anyway. The new Blinn rendering looked a pretty nice, but it wasn't revolutionary. Blinn rendering is variation on Phong. As with Metal rendering in 3D Studio 4, Blinn is another scan-line rendering variation.

Since I'm not an animator per se, I'll just give a quick summary of the new animation features that sound most interesting. Subobject animation lets you animate vertices, faces, and edges. Additionally, the built-in Bones system looks easy to use and straightforward. MAX 2 supports motion capture data, including real-time motion input and key reduction. It's interesting that support for motion capture is part of MAX 2; Character Studio 2.0 (demonstrated at SIGGRAPH '97) is supposed to offer some very similar features. MAX 2 also offers dynamic simulations, including basic physics features such as fall, slide, collide, and bounce.

Character Studio 1.2 for MAX 2 is included on the CD with MAX 2, but it won't fully install unless you already own the previous version. The features are basically the same; as with any other plug-in, Kinetix updated it so it would work with MAX 2.

Interface

I think MAX 2's weakest point is its user interface (UI). On the plus side, it's easy to learn and several of the underlying ideas (good use of context menus and some 3D interface innovations) help make work easier.

That's about all the good points that occur to me. On the negative side, MAX 2's UI is inconsistent and inflexible. The mouse-only approach is slow and error-prone, which would be forgivable if I could just edit the toolbar and access everything with keyboard shortcuts, as I can with Microsoft Office applications. Alas, MAX's UI is about as flexible as the immutable forces of nature. That's bad in any application and unforgivable for a product that leads an industry.

What's inflexible about MAX's UI? One of my biggest complaints is that the input is dependent on mouse movement, and the keyboard shortcuts are substandard. The list of key-assignable commands is frustrating: it's unorganized and lacks many major commands. For example, there's no way to assign a keyboard shortcut to the commonly used editing command Move Vertex. Moving a vertex requires a varied sequence of precise mouse clicks.

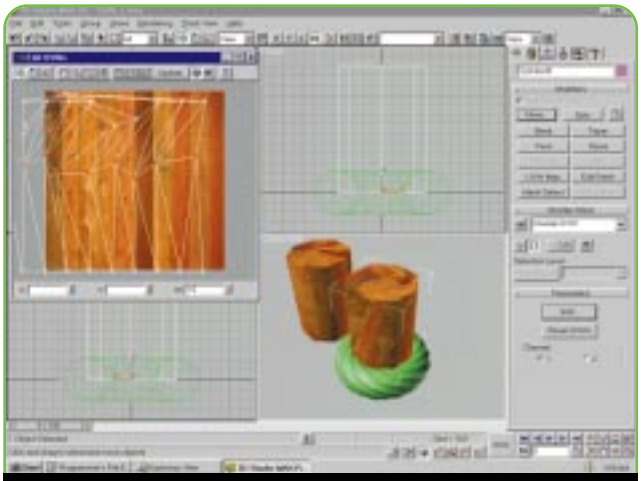


FIGURE 3. *Unwrapped UVs.*

Unlike most modern software, MAX 2 doesn't allow the user to customize the screen layout. Modeless dialogs are not resizable, the menu structure is frozen, and Kinetix's sidebar is certainly not customizable. The user is stuck with Kinetix's idea of perfection (prominently featuring the Percent Snap button) for all eternity. Furthermore, MAX doesn't have the standard Window main menu item; it was combined with the View menu, so opening the Materials Editor window is under the View menu. Also, don't change your screen color to a black background: MAX's material selection font is always black.

Again, most of those layout issues wouldn't be worth mentioning if MAX allowed the user to change them — but it doesn't. Heck, I'm even annoyed at $25 shareware that doesn't allow me to drag and drop toolbar buttons; when my $2,500 software package doesn't have a customizable UI, I'm horrified. It especially burns me because Kinetix obviously spent so much time and energy on MAX 2; how could it ignore the UI?

Other UI problems are abundant, though less critical. White space is poorly used. Some of the side-bar buttons resemble Windows 3.1 dialogs: huge fonts with poorly designed white space. Some of the most confusing dialogs (for example, the snap setup) are much improved, but the interfaces to newly introduced features are not. For example, the modeless NURBS floater window (which is begging to be a docking toolbar) consumes my precious desktop with useless white space, and I can't resize it.

Nontiling 3D windows would have been nice. According to Kinetix documents, the display tiles for performance reasons. Why not let users decide that trade-off? Some of us have darned fast computers; we'd rather have our interface flexible than fast.

And then there's Strokes, a movement-recognition system that reminds me of the Apple Newton's handwriting-recognition system: cute and innovative, but poorly implemented. I found that it worked for the preset functions, but adding new functions wasn't practical. The UI didn't let me edit the stroke shape, and its recognition was very primitive. My initial impression was that it's a pure gimmick, but when I realized Strokes could assign keyboard shortcuts, I got all excited: "Strokes be damned, if this thing lets me set keyboard shortcuts to anything, then it's great!" Nope. It offers the same crippled list of assignable functions as the standard MAX interface, with a only a few others thrown in. Sigh.

In a few cases, Kinetix has some good, forward-thinking interface designs in MAX 2, especially in the 3D interface area. Most notably, the snap is now predicted by a light blue 3D cursor, underneath the 2D cursor and showing where the grid (or endpoint, or whatever) is. This works far better than MAX 1, although it's a little sluggish and sticky-feeling (even on my P2-266). Another nice 3D interface is the highlighting of available edges on NURBS surfaces — as with the snap, the appropriate 3D object lights up as you

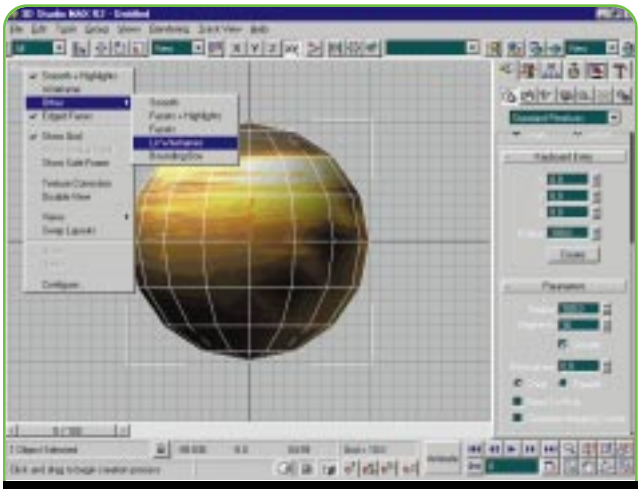


FIGURE 4. *MAX 2's Edged Faces view.*

pass the cursor over it, a quick, intuitive interface that works really well.

Kinetix can afford to be playful with its 3D interface since there aren't many decent precedents. However, if Kinetix wants to replace standard features such as docking toolbars, the solution has to improve upon the original (as in uncluttered, flexible, fast, and intuitive).

Display Improvements

I love the new Edged Faces view (Figure 4). It's a combination of wireframe view and smooth plus highlight view, and it probably took Kinetix about 10 seconds to implement (not that we users care). Real-time 3D modelers especially appreciate this one: we often need to see face boundaries while working with textured surfaces. To work around this problem in MAX 1.2, my coworker Lisa Washburn realized that the poor lighting in MAX's rendered views could be a feature, not a bug. She put a bright light in the scene and viewed the model in facet plus highlight mode, which textures the model with simple lighting. The severe lighting changes at the face boundaries revealed the edges of the faces. Of course, this workaround has drawbacks: not all edges are always visible and it's difficult to see the original texture.

One of MAX 2's most touted features is its direct support for OpenGL and Direct3D. Apparently, Kinetix decided that Heidi needed alternatives — understandable, given such stiff competition. I hope it works for others, but I didn't have much luck in my own test. I'm using STB's Velocity 128 AGP graphics card in my P2-266 workstation, and I couldn't get MAX 2 to make use of it. Neither the Direct3D nor the OpenGL options worked.

I embarked upon a little journey through Kinetix's tech support and documentation. In summary, I found the following: Kinetix has a web page that lists compatible cards (www.ktx.com/3dsmaxr2/html/graphics_cards.html), but as of this writing, there wasn't much on there — they didn't test the Rendition Verite, for example, which is one of the most common hardware accelerators available. Somebody in tech support told me that MAX 2 requires 8MB of video RAM for Direct3D support (I'm not sure that's true). Furthermore, I learned that the Direct3D support requires the DirectX 5 drivers — often, manufacturers are shipping DirectX 5 with DirectX 3 drivers. For OpenGL support, I needed to get drivers for version 1.1 or later from the card manufacturer (STB doesn't have any OpenGL drivers on their site). After some checking, Kinetix's tech support told me that using OpenGL under Windows 95 is "not possible," but that he'd check into it further and get back to me.

At a local users' group meeting, I learned that several people had successfully set up MAX with OpenGL and various graphics cards. One user reported that the Permedia with the OpenGL driver distorted textures, but that the problem disappeared when the software driver was used. Another user reported that the TrueFX Pro card also had distortion with OpenGL. Others reported that oddly enough, hardware acceleration didn't make that much difference to display performance; in one case, the software Z-buffer was 30 percent faster than the hardware-accelerated OpenGL driver. However, because OpenGL can easily be set up to bypass the hardware, I think it's possible that this report is inaccurate. No one else at the meeting had used MAX with a Riva 128-based card, so I didn't get any insights into my own hardware woes — and my card still won't accelerate MAX.

Regardless of hardware acceleration, some of my fellow artists report that MAX 2 has improved their display performance significantly. I tested this by creating a simple benchmark test: about ninety simple wireframe screen redraws and view transforms. Here's what I did:

1. Start MAX clean and make sure it's maximized. Create a Sphere object (16 segments), apply an edit mesh modifier, and Go to Sub-object/Face.
2. Change to the front view. Zoom extents, and make sure it's a wireframe view, with no grid showing and degradation override enabled. Select a face on the equator that has the right edge aligned with the axis of the sphere.
3. Choose Arc Rotate, click on the sphere, and start timing as we hold down the right arrow key. When all the red edges of the selected face have rotated so that they're all invisible, measure the elapsed time.

After entering my ultra-clean-secure-turbo test environment (a clean boot of my usual computer), I ran the test on the slowest MAX-capable machine I have: a Compaq Armada 4130t laptop (P133, 48MB RAM, 800×600×16 display, and built-in graphics) running Windows 95. Of course, with this low-end hardware, both MAX 1 and MAX 2 can only use their basic software display drivers.

TABLE 1. *Wireframe 16-Segment Test Results.*

Software	Time
MAX 1.2	35 seconds
MAX 2.0	30 seconds

TABLE 2. *Wireframe 256-Segment Test Results.*

Software	Time
MAX 1.2	73 seconds
MAX 2.0	60 seconds

TABLE 3. *Texture plus Highlight 16-Segment Test Results.*

Software	Time
MAX 1.2	35 seconds
MAX 2.0	31 seconds

Table 1 shows the results of the initial test. As you can see, MAX 2 was a little faster than MAX 1 (17 percent improvement). The test seemed to reflect my experience in using MAX 2 (that is, a small performance difference). But it made me curious: Was there a big performance increase with geometry handling, while my little test only looked at screen redraws? I reran the test with a great big 256-segment sphere. This monster had thousands of faces in the same screen area as the other sphere.

Table 2 shows the test results for the 256-segment sphere. The 22 percent improvement shown in this test made me think that MAX 2 has improved more in geometry handling than it has in its software-rendering pipeline. But what about nonwireframe objects? For this test, I created a sphere with a default material and added a bitmap ("sunset90.jpg") to it. I deleted a face for reference and applied mapping as well. The results are in Table 3 and, again, show a small performance improvement (13 percent, roughly).

I can almost hear the developers who sweated out those 10-20 percent performance increases grinding their teeth when I call them "small improvements." Let me explain: while I applaud any performance improvements, they don't really change the way I work with MAX — especially since I can get a cheap hardware card and get a 100 percent or better improvement in performance. If I had the choice, I'd prefer improvements in accuracy, not speed. MAX 2 still has those really nasty interpenetrating object borders in smooth plus highlight Z-buffering, for example.

Documentation and Support

Fortunately, the problems that I encountered trying to get my cheap graphics card to accelerate MAX gave me an opportunity to test Kinetix's support system. First, I started where all good users start: the manuals. When it comes to Direct3D and OpenGL support, I found a very terse, single-page discussion hidden in the back of the manual and one paragraph about OpenGL on a loose card in the box — that was it as far as printed documentation.

The online help is all new, and it's based on Microsoft's Internet Explorer instead of the standard Windows help system. I couldn't see what advantages it offered over normal help: It was different, but not clearly better. For example, it lacks the "<<" and ">>" buttons that I like for browsing neighboring topics. It has slick features such as smooth scrolling rather than snap-jumps when you push the [Page Down] key, and the simultaneous index/window interface was both good (convenient to have the topics always present) and bad (on a low-resolution screen, I'd prefer to see more text than half a window). In any case, the built-in help files had more information than the printed documentation, but didn't help with my driver issues.

The problem with the documentation isn't simply that my particular issue isn't adequately covered. Other users have complained strongly about MAX 2's documentation, saying that the manuals aren't as thorough as they once were. I agree — MAXScript, for example, has almost no printed documentation, and help with NURBS is difficult to find. Also, the online help files are not fully integrated — some are in the old Windows help system and some are in the new Internet Explorer-based system. I couldn't find any reason for the discrepancy.

Next, I spent an hour or two browsing Kinetix's online support site (support.ktx.com). I am so thankful that Kinetix online support has finally moved off of CompuServe (which I find difficult to use, expensive, and generally outdated) and onto the free, easy-to-access Web. The Kinetix support site is a sophisticated bulletin board system that is fairly easy and comprehensive, but you'll spend a lot of time waiting for pages to load.

Kinetix's support site has one unusual feature: not much privacy. The system publishes the number of times you log in and how many posts you've made. I didn't mind it personally, but it definitely felt weird and encouraged snooping. ("Is Jane Doe working? She's logged on 20 times in the last two weeks, with huge long posts every time.") Content-wise, there's lots of good stuff on there — users, both newbies and professionals, help other users, and the sysop has a very good attitude. Distracted, I browsed the MAXScript section, where I was very impressed to see John Wainwright (the MAXScript programmer) present, chatting amicably with the adoring masses.

Still in pursuit of a solution to my display acceleration problem, I browsed all the relevant areas that I could find on the support site. Surely, other users are struggling with the same issue, but I found no major answers.

Still lacking a satisfactory solution to my display problem, I called tech support. MAX 2 comes with ten days of free phone support, which is intended mainly for set-up issues, not general support. The lines closed before 5:30PM, which cost me a day. There's 5 ½ techs, and they're within shouting distance of the QA people. The support people say they're very active on support.ktx.com — someone's on there every day, answering the harder questions (and letting users answer each others' easy questions). I was very glad to hear that.

Unfortunately, tech support couldn't really answer the question. The person I talked to said lots of people are calling with the same basic issue: how best to use MAX 2 with cheap graphics hardware. At this point, I'd spent about three

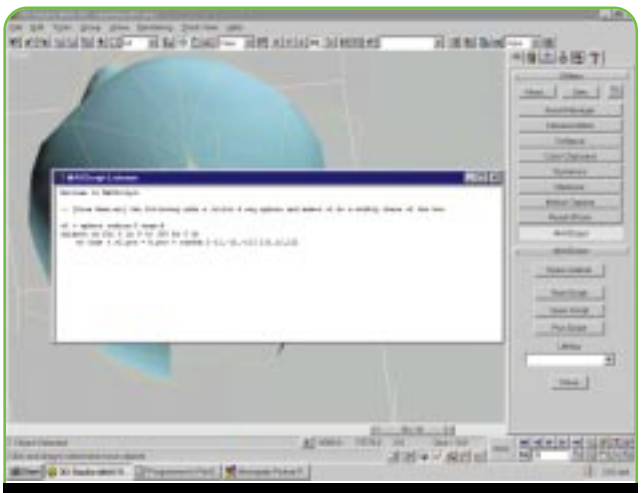


FIGURE 5. *MAXScript certainly doesn't look overwhelming... but stand back.*

hours on this problem and if I wasn't writing about it, I would have given up. But I have a cheat: Kinetix has special people that regularly contact product reviewers and offer to help answer questions. When my special person contacted me, I asked her about the difficulties I was experiencing with my graphics card. She told me that yes, Riva 128 has been tested and works via OpenGL. As of this writing, that round of communication is still in progress, which means I still don't know whether or MAX 2 is really going to work with this card.

MAXScript

MAXScript is truly great. That dull little button in the Tools menu will change the way we use MAX. What does it look like? Not much. As Figure 5 shows, it looks quite pathetic compared to today's programming workbenches. As an artist, however, I appreciate a minimum of confusion when I have to confront programming tasks. I wish MAXScript included a few more debugging aids, a better text editor, and better help (where are my printed reference manuals?), especially for MAX-centric commands and actions. Still, the existing setup basically works. The example code, DEMO.MS, is a nicely written and easy-to-understand introduction to writing scripts for MAX, but it leaves the user wondering where to go next.

I've done a little noodling around with MAXScript. I read the manual and wrote a simple function or two. But I haven't yet used it enough to evaluate its power (or limits thereof). Compared to other built-in languages, (I'm experienced with Word 97's built-in macro language, and I spent way too much time in AutoLISP, AutoCAD's built-in language), I think MAXScript looks really good. The syntax is simple, as it should be, and many of the annoying parts of programming (declaring variables and garbage collection, to name just a couple) are seamlessly hidden. At first, I wasn't impressed by the functions that interact with specific MAX objects in a scene; their names and syntax were confusing.

MAXScript seems quite well integrated with MAX. It can interact with MAX's database, UI, and (to a degree) the file system. It can also access plug-in functions, which is very impressive, and it can even use OLE to integrate with external applications, which is bloody amazing from a user's point of view.

I spoke with a plug-in developer about the power of MAXScript. He said that his commercial product could have been written in MAXScript and that he foresees MAXScript being very useful as a prototyping tool for plug-in developers. This is good news for starving artists; I hope we see cheaper (albeit slower) MAXScript versions of commercial plug-ins.

From what I can see, MAXScript is great, but there's always room for improvement. I'd like to see multitasking. I want to write scripts that run concurrently with MAX. I'm also concerned about security. MAXScript's power is quite broad, and scripts are easy to download and execute. Many users aren't experienced programmers and don't know how to review code for malicious behavior. I worry that MAXScript is an ripe environment for dangerous viruses (much like Word macro viruses, only worse). I think Kinetix should take a proactive role in preventing disaster for its users.

3D Studio MAX R2

Rating (out of five stars): ✪✪✪✪

Kinetix Inc.
San Francisco, Calif.
800-789-4233
www.ktx.com

Price (list/street): Full version - $3,495/$2,899; Educational version - $995/$899; Upgrade from MAX 1 - $795/$695 (commercial to commercial)

Requirements: At the minimum, you'll need a Pentium 90 with 48MB running Windows 95. Kinetix recommends a Pentium 166 with 128MB running Windows NT.

Technical Support: Ten free days from your first call. Other support varies by dealer (some will relay tech support issues to and from Kinetix).

Return Policy: The dealer says there are no returns of any kind allowed. The license that comes with MAX 2 says that you can return MAX for a full refund within 15 days if you don't agree with the license terms.

Pros:
1. MAXScript allows quick, easy customization and simple tool creation.
2. All the little bug fixes (especially VRML exporter) make this product more stable and easier to use.
3. OpenGL/Direct3D support allows users to use inexpensive graphics cards.

Cons:
1. "Frozen" (noncustomizable), cluttered, mouse-centric interface slows down experienced users.
2. Incompatible .MAX file formats make sharing data near-impossible with MAX 1 users.
3. Assorted documentation problems (inadequate tutorials, multiple online help systems, and so on).

Competitors (with estimated prices): Softimage 3D ($7,500); Softimage 3D Extreme ($15,000); Lightwave ($1,995/$1,795); Alias PowerAnimator; Electric Image ($2,495/$2,395).

.MAX File Woes

There is one major issue you should be aware of: MAX 2 writes a new kind of file format that's incompatible with MAX 1. Kinetix has confirmed that this is true, and hasn't announced any plans to fix it.

Who cares? More people care than you might think — specifically, anyone who uses MAX 1 and MAX 2 on the same project. For example, you buy MAX 2, but your customer/client/art director hasn't upgraded their MAX 1. How do you show them your artwork? You're stuck with using lame .3DS files to get data out of MAX 2 and back into MAX 1. You'll lose data because .3DS files don't store things such as multiple UVs per vertex.

Why did Kinetix do this? Is it an evil plot to force all MAX 1 users to upgrade? Or did Kinetix innocently say, "Let's make the .MAX file better," without considering how MAX 2 will be used with MAX 1? I think Kinetix (or some other plug-in developer) should

fix this discrepancy by creating a MAX 1-compatible .MAX file exporter.

New Features Replace Plug-ins

A number of new features overlap existing plug-ins. The most obvious is the morph blending feature. This allows the user to set multiple morph targets and blend between them. It offers features very similar to MorphMagic, Smirk, and Mix (there's a good review of these three in the December 1997 issue of *3D Artist* magazine, though it mistakenly says that MAX 2 doesn't offer this feature). Users report that morph blending works fairly well, but not as smoothly as the plug-ins do. A bunch of other features also invade plug-in territory: Bones, Lens flare, and the Sun System, which creates correct sun positioning given a latitude and longitude.

It's interesting to watch Kinetix attempt to balance its desire to leave room for plug-in developers with its attempts to increase MAX's feature set. The message behind MAX 2 is that Kinetix has no problem stepping into its developers' markets if it will make MAX better. That's good for users, but bad for developers — it reminds me of Microsoft's tactics.

Gossip

After I'd played with MAX 2 long enough to form an initial impression, I asked some of my peers about it. First, I e-mailed a few artists; their comments are scattered through this review. Then I went to the local San Francisco 3D Studio Users' Group meeting.

About half the people at SF3DSUG had used MAX 2, and they seemed to think it was worth the money, especially since many of MAX 2's built-in features saved them the cost of having to buy certain plug-ins (such as lens flare). Not everybody was totally thrilled, though. There was some heartfelt complaining about the printed documentation. "We want more than reprinted MAX 1 manuals," was a refrain that I heard more than once. Another hot topic was the attempt to get MAX 2 working with cheap 3D accelerator cards (sound familiar?); many varied opinions were bandied about and some good data was collected. Notably absent (or perhaps incognito) were Kinetix employees — I would have expected someone from Kinetix to be there, but at that particular meeting, no one was. There was a MAX dealer who seemed to speak on Kinetix's behalf/defense, which was helpful.

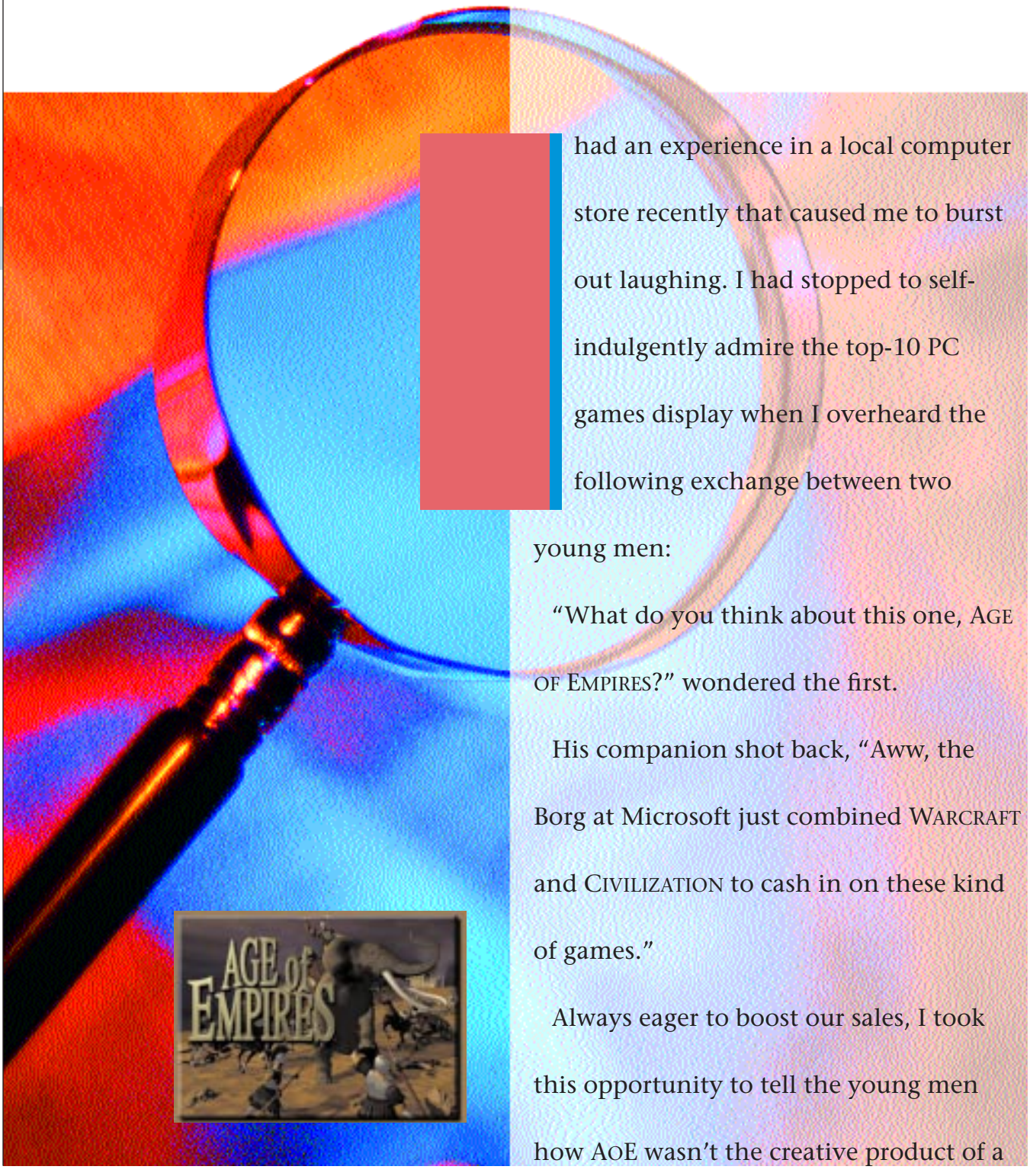
The Drum Roll...

MAX 2 is better than MAX 1. Is it $800 better? That depends on what 800 bucks means to you. If you're a hungry freelancer who barely came up with the cash for MAX 1, then you can get by without MAX 2. It's better, but it isn't going to change your work enough to justify the cost.

On the other hand, most art departments and freelancers with healthy incomes should definitely upgrade: you could save the cost of the upgrade just by using cheap 3D graphics hardware (if you can get it working) and plug-ins (if you haven't already bought them). ■

E emble AGE OF EMPIRES

by Matt Pritchard



I had an experience in a local computer store recently that caused me to burst out laughing. I had stopped to self-indulgently admire the top-10 PC games display when I overheard the following exchange between two young men:

"What do you think about this one, AGE OF EMPIRES?" wondered the first.

His companion shot back, "Aww, the Borg at Microsoft just combined WARCRAFT and CIVILIZATION to cash in on these kind of games."

Always eager to boost our sales, I took this opportunity to tell the young men how AoE wasn't the creative product of a

The AGE OF EMPIRES development team. The author is second from the right in the row of guys who are kneeling.

giant corporation, but of a small group of talented people living right in their own backyard.

For us, AGE OF EMPIRES was not only a game of epic proportions, it was an epic journey for a small band of people determined to turn an idea into a real game company. Along the way, we laughed, we cried, we consumed pizza and caffeine, and we learned a great deal about making games.

Designing the Past Perfect

Obviously, AGE OF EMPIRES didn't start out looking like the final product. Despite some accusations, DAWN OF MAN (AoE's original title) wasn't created to be a WARCRAFT II clone. (In fact, WARCRAFT II wasn't released until after AoE's development was well underway.) Instead, the final design was evolved and refined over time, with a few significant design events along the way. One of the best things I think you can have in a game company is a staff that plays a lot of different games. This was true of our staff at Ensemble, and was helped in no small part by programmer Tim Deen's habit of buying and actually playing almost every new PC game as it came out. It was Tim who brought WARCRAFT II to the attention of the rest of the Ensemble staff. At that time, many of AoE's game elements, such as resource management, empire building, and technology research, were taking clear shape. However, we didn't really know what to do about combat. WARCRAFT II was a splash of cold water in the face, waking us up to how much fun real-time combat could be. Several times a week, the staff would stay late to play multiplayer WARCRAFT. These impromptu events continued until AoE reached the point in development when it became more fun to play than WARCRAFT.

Another major shift occurred a little over halfway through development, when the designers were looking at AoE's localization plans. They realized that AoE would be sold in Asia, but didn't include a single culture from that region. We held a company-wide meeting and decided to add early Asian civilizations to the early European, African, and Middle-Eastern tribes that we'd already developed. Though the addition would create more work for the artists and designers, the enhanced appeal that the game would have in Asia would make this a profitable decision.

All of these changes occurred because the game's designers weren't afraid of taking design input from the rest of the staff. Making a game that everyone would be proud of and would want to play was something that got more than just lip service at Ensemble. Perhaps the best example of this core value is the Wonder, the penultimate building that a player can build and use to win the game. In early 1997, AoE was great for slugfests, but everyone felt that the game play needed to offer something more. Numerous ideas were tried and rejected. Then Mark Terrano, our communications programmer, hit upon the idea of building an "Armageddon Clock" that would force players to drop what they're doing and respond to the new challenge. AoE is chock full of little ideas and touches that were thought up by the artists and programmers. This participation tangibly increased the sense of ownership and pride that we all took in the game.

One of things that is truly underappreciated about the designer's job is play balancing. The designers spent months and months adjusting costs, strength, speed, and many other statistics in an effort to create a game that was fun to play and yet didn't offer any loopholes or cheats. At this point, I realized that Tim Deen was truly a gamer's gamer. During development, if any of the various iterations of AoE's design opened up a way for one player to take advantage of another player and thus make the game one-dimensional, Tim would find it. And when we didn't believe his assessments, he would promptly show us by using the loophole to kick our butts at the game. For the better part of a year, play balancing was a prominent task, and it paid off in giving AoE more staying power and better game play than many others in the recent crop of real-time strategy games.

Blazing the Multiplayer Path

Multiplayer support was an integral part of the early design, and AoE was structured in such a way that most of the game could not differentiate between human and computer players. When DirectX first came out, it appeared that DirectPlay would be the best choice for providing communications over the widest variety of connection types.

To support a high number of moving units, we went with a modified game synchronous model, where the entire game is simultaneously run on all machines. Only moves, changes, and commands are communicated to other machines. This approach has the advantage of minimizing the amount of data that has to be sent. The unan-

Matt Pritchard is busy trying to be a modern renaissance man. He designed the graphics engine for the hit game, AGE OF EMPIRES, and is currently working on the next generation of strategy games. He can be reached at mpritchard@ensemblestudios.com.

ticipated danger of using this model is that it can generate a condition where the game on one machine becomes out of sync with the game on other machines. This caused some very hard-to-reproduce bugs with AoE.

Load metering, the process of determining how much bandwidth the game updates required, was done before the computer AI was completed, and was based on the data flow model taken from human players. As a result, we initially missed the fact that computer players would sometimes issue large numbers of commands in quick bursts. We did, however, address this oversight with the first patch. An area where AoE's communications worked out better than expected was the game's ability to dynamically adapt to latency. A sliding window delays the actual game time when a command takes effect, so that all players receive the command and do not have to pause before executing it.

The problem of handling players who have dropped from a game presented Mark Terrano with difficult challenges. Since drops are unpredictable, usually there is no way to know what happened. The problem could be the game logic, Winsock, the physical connection, or the ISP, and could exist on either the sender's or receiver's side. Getting the game to handle drops by anyone at anytime required a great deal of work.

One of the lessons learned from the multiplayer experience was to make full use of communications testing tools, such as automated logs and checksums. Also, we discovered that creating a simple communications data flow simulator program can provide great benefits and isolate the communications code from the rest of the game.

DirectPlay also turned out to be problematical. Difficult-to-reproduce bugs, quirky behavior, and poor documentation made the going more difficult. Guaranteed packet delivery for IPX was one of the more notable examples. At the CGDC, Microsoft promised to deliver this feature with DirectX 5 and even included in the beta. However, when DirectX was actually released, this feature was nowhere to be found. The cost of that one missing item was the extra time we had to spend writing our own guaranteed delivery system and a bad packet generator program with which to test it.

All of the 2D sprites in AoE began life as 3D models.

Painting the Scene

AGE OF EMPIRES contains 20MB of in-game sprite graphics. Even though all of the displays are 2D, we decided early on that all of the graphics in the game would be taken from 3D models. We used 3D Studio and 3D Studio MAX for art production. Because 3D rendering is so time-consuming, each artist was issued two machines each, both usually 200MHz Pentium Pros with 128MB of RAM, which at the time was better equipment than the programmers were using. The objects in the game were created as 3D models that had anywhere from a couple thousand to 100,000 polygons. The models were then textured, animated, and rendered out to a .FLC (Autodesk Animator) file with a fixed 256-color palette.

So far, the process I've described is identical to that of many other games. At this point, however, the artists added another time-consuming step. The .FLC files were handed off to a 2D specialist, who took the animation apart frame by frame and "cleaned up" each image with Photoshop. The clean-up process involved sharpening detail and smoothing the edges of the irregular shapes. Since most of the sprites in AoE had screen dimensions of only 20 to 100 pixels in each direction, the visual quality improvement that we realized was significant. When AoE was shown at the 1997 E3, the artists received numerous compliments on their work from their peers at other companies.

The choice to go with 3D models for the in-game objects provided benefits for other art needs, as they were readily available for use in the static background scenes that appear on the menu, status screens, and the various cinematics. The cinematics, including the three-minute opener, were a full-time project unto themselves.

The 256-color palette (actually 236) used in AoE was something of a problem. The palette was chosen and set in stone at the very beginning of the project, before most of the models and textures had been created. As a result, it turned out that some portions of the color spectrum, such as browns for wood textures, had too few available colors to get the best visual quality. The palette wasn't revised during the development process because that would have required rerendering and touching up every image in the game — far too expensive time-wise. On the other hand, the palette did have a wide and balanced range of colors, which contributed to the overall bright and cheerful look of the game's graphics. If we do another 8-bit color game, we'll generate the palette at a point further along in the development process.

Going for Speed

Performance is an issue for all but the simplest of games, and it certainly was for AoE. When I joined Ensemble, the game was still in an early form and slow. The two biggest problems were the graphics engine (which was just plain slow) and various update procedures, which produced occasional pauses of up to a second in game play. If we were going to sell to anything but the most cutting-edge systems, some serious optimization was in order. The story gets personal here, as I did a great deal of the work on this part of AoE.

I started by trying to get a handle on what the over 100,000 lines of C++ code did (the source would rise to over 220,000 lines before it was finished). After spending a few weeks studying

what we had, I proposed a major overhaul of the graphics engine structure, including writing a major portion in assembly. AoE's original design team asked if the frame rate of a benchmark scenario could be raised from its current 7 - 12 fps to 20 fps. I told them yes. Inside I was sweating bullets, hoping that I could deliver that much improvement.

I couldn't just go ahead and rip out the old graphics engine, as it would hold up everyone else, so I spent the next five months working mostly on the new engine. Along the way, I managed some incremental improvements that upped the frame rate to 10 - 14 fps, but the big breakthrough came during an all-nighter, when the last piece of the new design was put into place. Much to my surprise, the benchmark scenario was now running at 55 fps. It was exciting to come back into the offices the next day and see the formerly sluggish animation running silky smooth. But my work was not all on the graphics engine. I also spent a great deal of time identifying and optimizing myriad processes in the game. Since the game was real-time, many improvements involved spreading out a process over several turns rather than of stopping the game until it completed. In the end, the optimizations paid off handsomely and allowed us to raise the default resolution from 640×480 to 800×600.

A practical lesson that I learned from this experience was how much additional overhead and slowdown a game can acquire as it approaches completion. Often, early in a game project the engine will show great performance — but the game's not finished. When you replace the simple test levels with the complex final levels, add all the AI, UI, and bells and whistles, you can find a world of difference in actual performance. This was true for AoE as well. As we approached completion and all of the loose ends were tied off, many of the performance gains were traded in for new features.

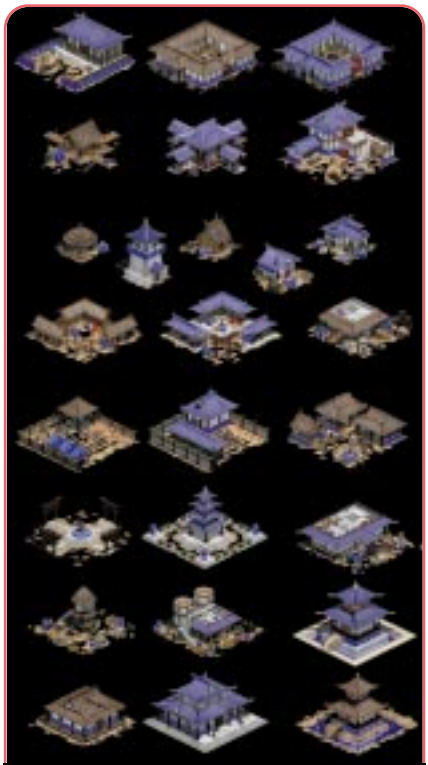
Things That Worked Out (S)well

1. The game was broken into separate engine and game components. About halfway through development, there was concern that the code base had expanded far enough beyond the initial design in some areas that every new change and addition would look like an ugly hack. Lead programmer Angelo Laudon and Tim Deen took two weeks and separated the code base into two separate sections, the general engine (Genie), and the game itself (Tribe). The conversion was very successful and allowed the AoE programmers to retain a nicely object-oriented design. The benefit here was that it made the code much easier to modify and extend, saving the programmers a great amount of development time.

2. We made the game database driven. Thanks to the object-oriented design, almost nothing about any object in AoE is hard-coded into the program. Instead, huge tables of information describe every characteristic of every object that appears in the game. The designers used a system of over forty Paradox tables to control and shape the game. As a result, they were able to constantly update and tweak the game, and then test their changes without having to involve a programmer.

3. We stayed in close contact and working together with the publisher. I can't say enough good things about how the close contact with our publisher, Microsoft, helped the development of AoE. By keeping them "in the loop" on our progress, they worked with us instead of against us as things happened. The best example of how this relationship aided development is the way we handled schedule slippage. Each time something took longer than expected or new problems cropped up, we effectively communicated the delay to Microsoft. With a clear understanding of what was happening and why, they reaffirmed their commitment to assist us in producing a quality game, whatever amount of time that would take. So instead of being panic-stricken and whacked out, we remained professional and focused on our goals.

4. We played our own game. While this may sound simple, it's very important. Throughout the development process, every person in the company not only play tested, but played AoE with the purpose of having fun. As a result, we were very in tune with why the game was fun, and what people were likely to get out of it. We had 20 guys who were determined not to let the game play be compromised or watered down.



Early Asian civilizations had to be added when Microsoft announced plans to distribute AoE in Asia.

5. Performance was good. Performance truly means a lot if you want your game to have broad appeal. Age of Empires can adequately run an eight-player game in 16MB of RAM on a P120 system. Contrast that with Total Annihilation, which requires 32MB and at least a 200MHz CPU for an eight-player game. Achieving this level of performance required a group effort. The programmers expended extra energy on keeping memory con-

sumption in check and identifying bottlenecks, while the artists culled extra animation frames and reorganized the graphics to maximize free memory.

6. The company respected its employees. I have to say something about the way Ensemble Studios treated its employees and their families. It is well known that software development, especially game development, involves great sacrifices of time and can be hell on personal relationships. Ensemble's thoughtful management respected that by going out of their way to include families at numerous company dinners and other events, and to make them feel welcome to come by the offices at any time. Additionally, after crunching hard to meet a milestone, they would insist that employees take a couple of days off to catch up with their families. People were allowed flexible schedules if they needed them, and flowers or other tokens of appreciation were sent to the families periodically. The result of this deliberate action by company management should be obvious; company morale and loyalty was higher than I have ever seen in fourteen years of software development. My wife loves my job as much as I do.

7. Localization really worked. From the beginning, we knew that Microsoft wanted to release AoE in six different languages, including Japanese. About halfway through development, we updated our code base to provide full localization support. This required stripping out and replacing all text references in the source code and maintaining all game text in an external resource file. It also placed severe restrictions on how we could draw and display the text. We had to use the Windows GDI exclusively, something most games shun like the plague. It also meant that interface items such as buttons had to be sized to hold the largest possible translated form of their captions, limiting the clever things one could do with the design of the user interface. But we buckled down and did it, following the guidelines exactly. And to our pleasant surprise, the conversion was swift and painless. We felt even better when the translators at Microsoft told us that localizing AoE was the smoothest and most pain-free project they had ever done. The benefit to us was enormous in that we had a single executable program file that was the same for all translated versions of the game, thus avoiding the huge headache that comes with tracking bugs and releasing updates for multiple versions.

8. We worked as a team that respected all its members. A project of AoE's size required that we all work together in close quarters for extended periods of time. One of our expressed criteria in hiring new people is that we must be able to respect each other. This respect, complemented by the company's actions towards its employees, fostered an excellent sense of family and team spirit among everyone. We avoided having different groups develop a sense of isolation from the project, and as a result, attitudes and spirits remained high even during the worst crunch time. Had tempers flared and cliques developed, I honestly don't believe that AoE could have made it out the door in 1997.

Things That Went Wrong Or We Could Have Done Better

1. We held the beta test too late in the development cycle. A public beta test of AoE was held in August 1997, but we didn't come near to exploiting the full potential of it. We were too close to the end of the project to make any game play changes, despite the wealth of useful feedback we received. Manuals were already set to be printed, and most of the design had been set in stone. All we could really do was fix any bugs that were found. For any future projects, we vowed to hold the beta testing earlier.

2. There was inadequate communication with the QA people at Microsoft. For most of the project, bug reporting was handled through a database and developers didn't directly communicate with the testers. As a result many bugs wound up taking much longer to resolve, and new features went untested. An intermediate database was simply not enough to let testers and developers know what the other was really thinking. In future projects, we would like to assign a specific tester to each programmer and have them communicate by phone every couple of days. Near the end of development, this did happen for a couple people — for them productivity with testing and bug resolution was drastically improved.

3. We sometimes failed to coordinate development through the leads. Yet another area where personnel communication could have improved the development was among our own teams. Each team — Programming, Art, Game Design, and Sound — has a lead person who is responsible for keeping track of what each member of his or her team is doing. The lead is the go-to person when someone outside has new requests for the team. As the development of AoE progressed and the pressures rose, adherence to this system broke down as people went direct to get their needs filled quickly. We paid a price for it. People didn't know about programming changes or new art that was added to the game, and the level of confusion rose, creating a time drain and distraction. We all had to stop at times just to figure out what was going on.

4. We failed to adequately test multiplayer games with modem connections. One problem with our development environment is that it isn't comparable to the typical end user system. During the course of development, the multiplayer portions of AoE were tested extensively. When we played a game in the office, our fast machines, stuffed full of RAM, communicated with each other on our high-speed LAN. When we tested Internet play, our communications were handled through the company's T1 line. One thing that we failed to realize in our testing was the fact that most players would be using dial-up modem connections to commercial Internet service providers. And it wasn't just us; the same situation applied to the testing labs at Microsoft. As a result, modem connection games were under-tested. Bugs that were harmless when ping times were low resulted in dropped games for users on slower Internet connections. And our high-speed network masked the fact that under certain not-so-uncommon circumstances, AoE could require 15K of network bandwidth per second — six times what even a 56K modem can provide on the uplink side. As a result, we were taken a bit by surprise when reports of multiplayer game problems rolled in. Though our first patch fixed this problem, the situation was unacceptable. Now, each developer has a modem and several different ISPs are used, as modem testing is a big part of our testing process.

5. Portions of development relied on products that were not delivered on time. There was a second reason that modem games were under-tested; problems with the delivery and quality of DirectPlay from Microsoft. Features that were promised, and even included in beta releases, weren't present when the delayed final release was delivered. Direct X 5a wasn't available to us until a month before the game shipped. In the meantime, our communications programmer was burning the midnight oil writing the functionality that was expected but not delivered. Waiting on promised drivers and SDKs is one of the harder things that developers have to deal with, even those with Microsoft as a publisher have no control over them.

6. We did not plan for a patch. The version 1.0a patch, even though it was a success, was problematic in that as a company we had not planned for it. The general argument is that if you know you are going to need to release a patch, then you shouldn't be shipping the game in the first place. While one can take that stand, it's also a fact that no game developer's testing can match that of the first 50,000 people who buy and play the game. Your customers will do and try things that you never dreamed of, while running on hardware and drivers that you never heard of. Looking around, nearly every significant game released this year has had a patch or update released for it. Rather than deny this reality, we would like to allocate resources and expectations in the future so that we can release any necessary updates days or weeks after our games ship, rather than months.

7. We didn't manage "surprise" events as well as we could have. During the development period, we experienced several sudden events that caused us, as a company, to stop what we were doing. These events included the creation of a demo version of the game and materials for press coverage of AoE. While most of the events were beneficial to the company, we weren't very good at handling them, and they threw off our schedules. These disruptions mostly came late in development, when their impact was felt the most. One of our goals for future games is to minimize the impact of unplanned events by giving advance notice when possible and restricting them by minimizing the number of people that have to respond to them.

8. We didn't take enough advantage of automated testing. In the final weeks of development, we set up the game to automatically play up to eight computers against each other. Additionally, a second computer containing the development platform and debugger could monitor each computer that took part. These games, while randomly generated, were logged so that if anything happened, we could reproduce the exact game over and over until we isolated the problem. The games themselves were allowed to run at an accelerated speed and were left running overnight. This was a great success and helped us in isolating very hard to reproduce problems. Our failure was in not doing this earlier in development; it could have saved us a great deal of time and effort. All of our future production plans now include automated testing from Day One.

Patching It All Up

Once AoE was shipped off to production, we threw ourselves a big party to let off some stress. It turns out we were a bit premature in our revelry. Shortly after AoE arrived on store shelves we began receiving feedback on problems with the pathfinding, unit AI behaviors, population limits, and multiplayer play. Additionally, some bugs were found that a player could exploit to gain an unfair advantage in the game. Management was stirred to action at both Ensemble and Microsoft, and the decision to do a patch for AoE was made.

Although time was taken away from other projects, and it took longer than desired, the patch was a success; it vastly improved the quality of multiplayer games, fixed the bugs, and addressed the game play concerns that had been raised. And that brings the development of AoE to where it is today, which I hope is somewhere on your hard drive… ■

by John Podlasek

Don't Lie on Your Resume

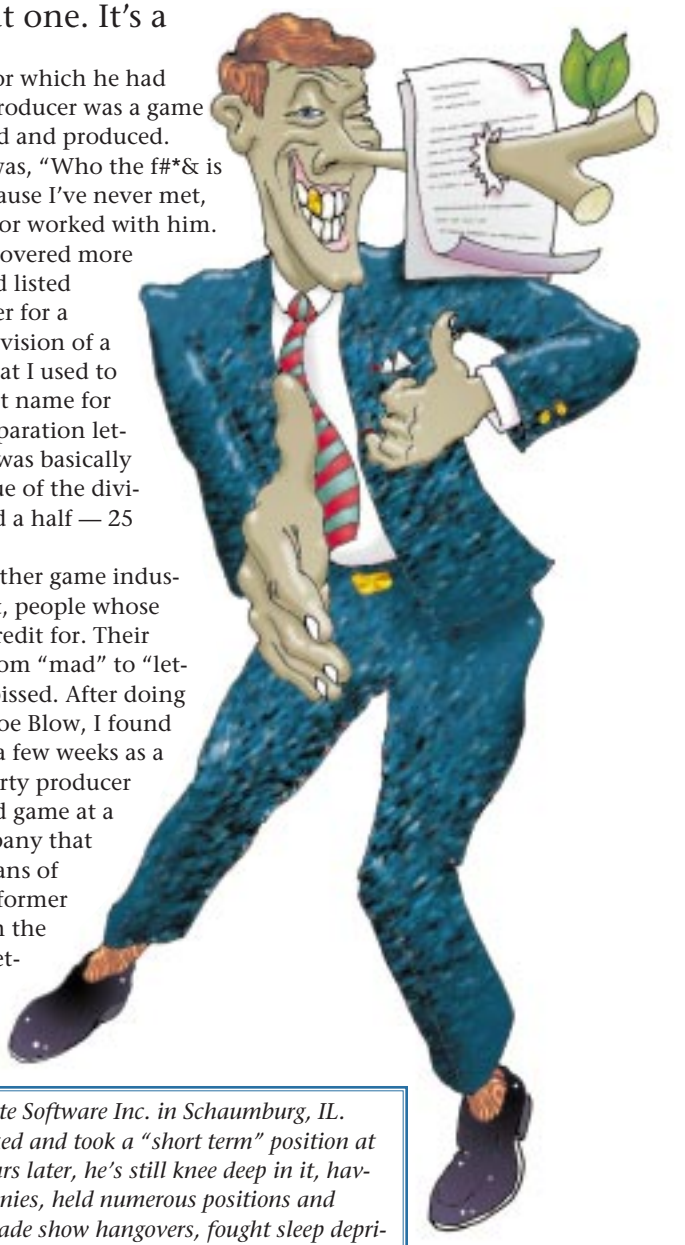
Résumés are a funny thing. They're a pain-in-the-ass to write, but you can't get a job (that doesn't require you to wear a name tag) without one. It's a time-consuming, frustrating, necessary evil, especially in the shaky and volatile world of the game development industry. At Blue Byte Software, we get around a dozen résumés every week from interested job seekers. It's pretty easy to see who's got their act together within the first few minutes, based not just on presentation, but on background, experience, and education. The company president passed along a résumé to me a few weeks ago that shocked me unlike anything I've read before.

It started with the sheer size of it — six pages. Yes, you read that right, six pages. Unless you're say, Winston Churchill, I don't see how you could logically come up with enough accomplishments to fill four pages, let alone six. So I read on, more for amusement than from any desire to "get this industrious go-getter in for an interview."

After three pages of objective, summary, and experience, Mr. Joe Blow (the name has been changed to protect the clueless) got to "Titles as of July 1997." Three pages of games followed, listing title, platform, position, and genre (an amazing 154 total — many listings were for the same game on various platforms and/or never-published titles). When I got to the last page, however, my jaw nearly hit the floor. One of the titles for which he had listed himself as producer was a game that *I* had designed and produced. My first reaction was, "Who the f#*& is this nimrod?" because I've never met, spoken, e-mailed, or worked with him. As I read on, I discovered more titles where he had listed himself as producer for a defunct gaming division of a big corporation that I used to work for (and can't name for fear of hungry "separation letter" attorneys). It was basically the entire catalogue of the division's last year and a half — 25 titles in all.

So I told some other game industry people about it, people whose work he'd taken credit for. Their responses range from "mad" to "letter-bomb-ready" pissed. After doing some digging on Joe Blow, I found out that he spent a few weeks as a worthless third-party producer on a nonpublished game at a division of a company that had short-lived plans of merging with my former division. Although the connections are getting confusing, the point is that

Continued on page 63.



John Podlasek is project manager of U.S. development for Blue Byte Software Inc. in Schaumburg, IL. While trying to break into the advertising world, he got side-tracked and took a "short term" position at NEC's TurboGrafx-16 gaming division. Over eight and a half years later, he's still knee deep in it, having worked for both monster mega-corporations and small companies, held numerous positions and titles, battled know-nothings and knuckleheads, nursed wicked trade show hangovers, fought sleep deprivation every summer and fall, and been involved with boring, average, great, and a few outstanding titles. He can be reached at podlasek@bluebyte.com.

Continued from page 64.

these are games that he didn't produce for a company that he didn't even work for. Amazing.

Anyone who's been in this industry for even a short period of time knows that it's more incestuous than an Alabama family reunion. Between the layoffs, mergers, and millions lost, plus the relatively small work force, people get around. They talk, have friends that left to work for other companies, share drinks, attend seminars and shows, and so on. It's ridiculous to think, "No one will find out."

Some may ask, "Why make such a big deal about it? The division isn't around anymore, the games weren't that popular, and he needed to beef up the résumé."My response is, "He lied!" Besides the obvious ethical question, stealing credit for games that you didn't work on cheapens and belittles the late hours and hard work put in by the people who did. It's disgusting to think back on all the headaches, frustrations, and late nights put in on a game to just have someone take the credit from you. Don't get me wrong. I'm not complaining about the hard work. It's the nature of the beast in game development. Besides, I know of a million more difficult ways to make a living than making games. The point is that when you put this much effort into a project (of any kind, really), it magnifies the attachment. When the level of effort, intensity, and time required for game development is expanded, you take your titles personally.

One other point stands apart from the ethical question and the attachment factor: survival. Having been out in the job market a little over eight months ago, I wondered, "What if this yahoo got a job over me because of the titles he lied about?" Most likely, any company dumb enough to hire him is someplace you don't want to work for anyway, but the possibility is cause for concern. Besides my own personal reasons, I have honest, hard-working friends out in the job market competing with this schmuck. It makes me even angrier knowing that good people may lose out on an opportunity because some guy thought nobody would know or care if he lifted a couple dozen game titles for the résumé.

But our friend Joe Blow isn't the only one guilty (just the most stupid and blatant). He's more of a poster boy for this dirty little secret that nobody talks about. And it's not even limited to résumés. I've read articles and web pages published by people with whom I used to work that twist, exaggerate, and simply lie about past titles for which they're claiming undue credit.

My advice? Keep your résumé well under six pages (unless you helped save Western civilization) and don't lie about the games that you didn't work on. The industry is small and well-connected, and it may come back to haunt you down the road. Who knows? Someone may even write a magazine article about how stupid you are someday. ■

ADVERTISER INDEX