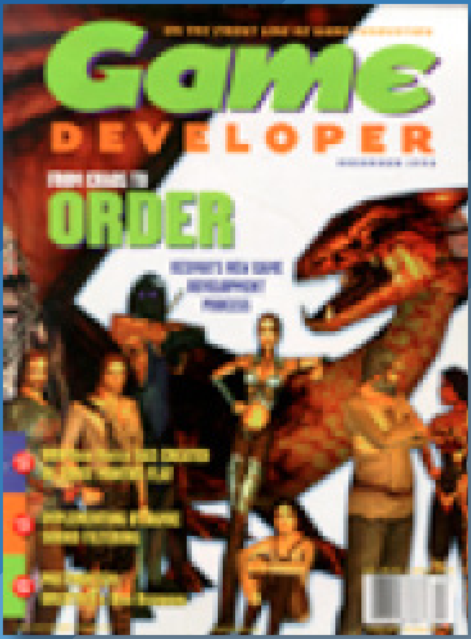
G A M E   D E V E L O P E R   M A G A Z I N E

DECEMBER 1998



UBM Tech

Back

Ditch the Death March

At the conclusion of World War II, Japan was nowhere near the industrial powerhouse that it is today. The management of Japanese companies was almost feudal in nature: plant managers were practically overlords, and workers were viewed as serfs. One of the people who helped turn Japan around in the '50s was American management consultant Dr. Edwards Deming.

Deming's lesson to managers in Japan and America was fairly simple. In a nutshell, he said that managers' infatuation with short-term thinking, merit systems, quotas, management by objectives (MBO), and so on, was all wrong. Instead, he said, special importance should be placed on the *process* of production, not the end product itself. If the production process was good, quality products would naturally flow from the company.

It worked. Take Toyota, for instance. The company's initial experience selling cars in America was disastrous; the Toyota Toyopet was a stark, underpowered vehicle that sold poorly. Later, armed with production processes based on the work of Deming, Walter Shewhart, and others, Toyota went back to the drawing board (literally). Using product testing, customer feedback, and studying the lessons of the then-successful Volkswagen Beetle, Toyota came up with the Corolla. This car sold well, and thanks to continued use of process improvement methodologies, Toyota's engineers were able to improve the product each year.

Of course, games are not pieced together on an assembly line. But many of Deming's lessons still hold true for game developers. In this month's cover feature ("Bringing Engineering Discipline to Game Development", page 37), Kesmai Studios Manager Gordon Walton explains how better development processes helped his company, and many of these experiences jibe with Deming's lessons. Many of you worked your tails off to complete projects in time for Christmas, so you're probably receptive to better methods of managing projects. Wasn't that death march fun?

As Walton explains in his article, Kesmai (like practically every game development company) experienced intense pressure to complete a number of games in time to guarantee their delivery to stores by Christmas. The burnout and frustration that this used to cause his teams prompted the company to find ways to gain better control of projects. And while Walton stresses that Kesmai's developers still encounter the occasional "crunch period," adopting formalized development standards and procedures for their projects has given everyone more confidence in estimating work duration, resulted in fewer bugs in their games, and made the process of developing a title more rewarding for everyone. Without a doubt, Walton's article provides the most detailed accounting of process improvement by a game development studio that I've seen.

"Sundance" Postscript

In the wake of my September editorial ("Where's Our Sundance?"), I received quite a number of letters from readers about various events that either already exist, or that are in the planning stages. First, Brooke Boynton of P.F. Magic reminded me that *New Media* magazine's Invision Awards (http://www.invisionawards.com) honor a wide range of multimedia and web products, including games. Second, it appears that Milia Games (http://www.milia.com), the French conference for interactive entertainment, will host a "New Talent Pavilion" at its upcoming event in February. Finally, the Game Developers Conference (http://www.gdconf.com) is launching the "GDC Independent Game Festival" at its show in San Jose, Calif., this March. In the interest of full disclosure, I must remind everyone that the GDC and *Game Developer* magazine are owned by the same company. Which, of course, made it very easy for me to twist their arms. ■

[illegible]

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

600 Harrison Street, San Francisco, CA 94107
t: 415.905.2200 f: 415.905.4962 w: www.gdmag.com

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

EDITORIAL

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

ADVERTISING SALES

Western Regional Sales Manager
Alicia Langer *alanger@mfi.com* t: 415.905.2156
Eastern Regional Sales Manager
Kim Love *klove@mfi.com* t: 415.905.2175
Sales Associate/Recruitment
Ayrien Houchin *ahouchin@mfi.com* t: 415.905.2788

ADVERTISING PRODUCTION

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

MILLER FREEMAN GAME GROUP MARKETING

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

CIRCULATION

Vice President Cirulation Jerry M. Okabe
Assistant Circulation Director Mike Poplardo
Circulation Manager Stephanie Blake
Circulation Assistant Kausha Jackson-Crain
Newsstand Analyst Joyce Gorsuch

INTERNATIONAL LICENSING INFORMATION

Robert J. Abramson and Associates Inc.
President Libby Abramson
720 Post Road, Scarsdale, New York 10583
t: 914.723.4700 f: 914.723.4722
e: *abramson@prodigy.com*

Miller Freeman
A United News & Media publication

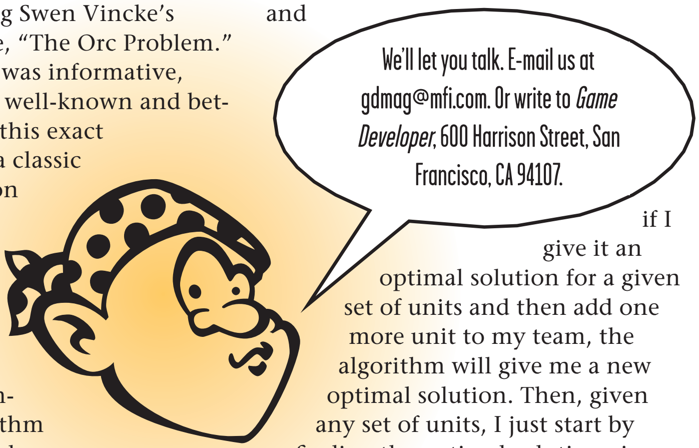
Chairman-Miller Freeman Inc. | Marshall W. Freeman
President/COO | Donald A. Pazour
Senior Vice President/CFO | Warren "Andy" Ambrose
Senior Vice Presidents | H. Ted Bahr, Darrell Denny Galen A. Poss, Wini D. Ragus, Regina Starr Ridley, Andrew A. Mickus, Jerry M. Okabe
Vice President/SD Show Group | KoAnn Vikören
Senior Vice President/Systems and Software Division | Regina Ridley

BPA International Membership Applied for March 1998

www.gdmag.com

Solving the Orc Problem

I just read through the October 1998 issue of *Game Developer* (all of us here at GameFX... err THQ... fight over the latest issue of *Game Developer* every month), including Swen Vincke's interesting article, "The Orc Problem." While the article was informative, there are, in fact, well-known and better ways to solve this exact problem. This is a classic resource allocation problem that can be solved quickly and easily using a technique known as dynamic programming. This algorithm is also sometimes known as Viterbi decoding or Hidden Markov model decoding. These algorithms were developed in the 1960s to assign nuclear missiles to targets.


We'll let you talk. E-mail us at gdmag@mfi.com. Or write to *Game Developer*, 600 Harrison Street, San Francisco, CA 94107.

You want to avoid greedy solutions for simple fitness functions if they are not the optimal choice. If you have a simple fitness function that only takes into account the two units involved, (that is, if unit *a* attacks unit *b* you get one score, and then if you also attack *b* with another unit *c*, the score for attacking *b* with *c* is not affected by the fact that it has already been attacked by *a*), then the "greedy" solution is optimal. That is, get each of your units to attack the unit that it gets the highest score for attacking. Because the pairwise evaluations are independent, you need to compare each of your own units against each enemy unit one time each. If you have *n* of your own units and *m* enemy units, this can be calculated in time that scales as *m***n*. In an eight on eight fight, this is 64 pairwise evaluations. In a battlefield game where the units are usually spread out and the scoring function takes into account the distance between the units involved, this method can work pretty well. This is probably less expensive than a single call to Vinke's fitness function. But the trade-off is that it won't give you as accurate an answer.

Dynamic programming is the solution for the harder fitness functions. If you have a fitness function where the score for attacking unit *a* with unit *b* depends on what other units are attacking it (or the configuration of units as a whole) then you have to use an algorithm that's just a little bit smarter. The trick here is that if I have an algorithm, and if I give it an optimal solution for a given set of units and then add one more unit to my team, the algorithm will give me a new optimal solution. Then, given any set of units, I just start by finding the optimal solution given one unit and then I keep adding units one by one and finding a new optimal solution, given the old optimal solution as input.

Here's how to execute that task. First, I make a matrix with enemy units indexing the rows, and with my own units indexing the columns. Then I start with column zero, evaluating my zero unit against each enemy unit in turn. Next, I fill in the corresponding matrix cell with that score. I might evaluate it with Vincke's fitness function (Fitness (*S, m, n*) where *S* is the layout of the board, *n* are all of the enemy units, and *m* consists of my units, which for column zero is my single unit zero). The highest score in the column represents the optimal solution given that single zero unit. It is also trivially true that the score in each cell of the column M(*0,j*) represents the optimal solution given that single unit attacking the *j* enemy unit (because there is only one such configuration). For each subsequent column, I evaluate a new score for each cell in the matrix, where the new score for the cell M(*i,j*) is the maximum of Fitness (*S, m, n*) given my new unit *j* attacking the enemy's unit *i* in combination with the configuration described by the cells of the previous row in turn. So for M(0,1) I would get the maximum of Fitness (*S, m, n*) where each time my unit 1 attacks the enemy's unit 0, and my unit 0 attacks each of the enemy's units in turn. I keep a pointer in cell M(0,1) to the cell in column 0 that I used to obtain the maximum score. Once I have done this for column 1, the largest score in column 1 represents the optimal configuration given those 2 units. The score in each cell M(*i,j*) represents the optimal score given my unit *j* attacking the enemy's unit *i,* and with all of my units less than *j* on the board. I can repeat the process for each new column for as many units as I want. Each cell has a pointer to the cell in the previous column representing the rest of the configuration of units. When I get to the last column and have placed all my units, I can get the optimal configuration of units by following the back pointers from the cell with the best score in the last column all the way back to the first column.

The first column costs *n* evaluations of the fitness function. Each subsequent column costs *n*^2 evaluations. For an eight on eight battle, this is 456 evaluations to get the optimal solution to the problem. This is far fewer than the 16 million the article suggests. The algorithm does scale roughly as *n*^3, which can still be prohibitive for large numbers of units. To handle this, you can use the simple trick of thresholding. For each new cell M(*i, j*), only evaluate against the top *k* scoring configurations in the previous column. This immediately cuts the algorithm complexity down to *n*^2. You can prune it even further by not evaluating every cell in the matrix, but say only the cells likely to score highly. You might evaluate each cell in a given column against a single random configuration of the previous units, and chose

"This is a classic resource allocation problem that can be solved quickly and easily using a technique known as dynamic programming."
-Rafael Baptista

only the top *l* scoring cells to fill in. This reduces the problem down to a time complexity of *n*. Glorious linear time complexity! In a fight with 40 units on 40 units and *k* = 5 and *l* = 5 it would cost 1,000 evaluations of the function. The solution is not guaranteed to be optimal, but unless the inputs are particularly perverse it probably will find the optimal answer. There are further optimizations. Fitness(*S, m, n*) is probably expensive. Because each cell represents a partial configuration of units, in each cell I could also store a partial evaluation of Fitness(*S, m, n*) that I use to help me evaluate the next column faster. With a reasonable function, one could probably get it down to about 1,000 pairwise evaluations of units and a few table lookups. To go even faster you could decode as described above using a cheap approximation of the fitness function. You could chose the top *q* configurations from a table evaluated with the cheap function and then evaluate those *q* configurations using your more accurate and more expensive function. Hey, that gets us down to a constant number of evaluations of the expensive Fitness(*S, m, n*) function (and a linear number of partial evaluations of some cheaper function). So we might decide to evaluate Fitness(*S, m, n*)… oh, say ten times. This might make it possible to field very large numbers of units. Or you can use all the time you save for the real-time generation of fractal trees or some other useless diversion. Why does this work? If you think about it, a stochastic algorithm like Vinke's spends most of its time evaluating configurations that are way off. It also reevaluates very similar configurations over and over again. Dynamic programming zeroes in on the neighborhood of good solutions right away. Each time you add a unit, it considers only configurations that are similar to what it had determined was the optimal configuration with one unit fewer. Think about what a human would do when solving the problem manually. You would assign pieces to enemies one by one. Once in awhile when assigning a new piece you might reassign one you had assigned before because now you have that particular enemy covered by a better piece. The dynamic programming approach also evaluates the situation in an orderly and predictable way that allows you to re-use computation in a way that is hard to do with stochastic methods.

By day, I am the AI programmer for THQ's upcoming excellent 3D space shooting game, EXCESSION. Our AI is very fast.

Rafael Baptista
GameFX
via e-mail

VINCKE RESPONDS. *What the Viterbi algorithm essentially does is find a least-cost path, and you correctly point out that in order for it to do this optimally its complexity qualifies as* m *times* n2, *or if* m=n, *as* n3. *Using some heuristic such as selecting a fixed number of alternatives as you do, or if you have access to some knowledge, a variable number of alternatives, significantly increases the speed of the algorithm, and indeed in most cases finds an optimal to near optimal sequence. You could, however, miss the global optimum. But I agree that for this particular problem and for real-time applications it is definitely superior to genetic algorithms, and I should've mentioned that in my introduction where, as it stands, I might have given the impression that it didn't get any better than* n^m. *The point of the article, however, was to introduce the reader to genetic algorithms, and to show how they can be applied to a large class of problems where the search space is complex and where there are many local optimums. You could say that the first example, that of the orc problem, was a bit mischosen, given that there are faster ways of obtaining a good solution. Still, that, together with the TSP example, should have driven home the point that GAs can be very useful. The fact that GAs spend time evaluating bad configurations shouldn't be regarded as a weakness, however. It is actually their great strength because it's how they overcome the local minima problem. You can easily modify them not to exploit this by configuring them to use something like weakest chromosome replacement together with fit-fit selection. Using that, the 40-40 situation can be transformed into a win situation in as little as 700 iterations without using any heuristic. Together with a good heuristic, this number can even be decreased. Of course, the overhead is still higher than that of the Viterbi algorithm, which is why I'm hitting myself on the head for not mentioning it. One other advantage GAs have over traditional methods is that they present several solutions, which can be significantly different, at a given time. This can give you some measure of randomness, which can be great when you are dealing with something like map generation, for instance. All in all, GAs aren't the be all and end all, but they do have significant advantages, and because of the ease with which you can adapt them to new problems, they are definitely worth considering when you're working on a tight schedule.*

Our AI is also very fast. Maybe one day they can play with each other.

"Our AI is also very fast. Maybe one day they can play with each other."

Swen Vincke of Larian Studios to Rafael Baptista of GameFX

MacOS and Audio File Formats

I read Chuck Carr's Postmortem, "989 Sutdio's NBA SHOOTOUT 98," in the September 1998 issue of *Game Developer* magazine, and I noticed his mention of the problem of Macintoshes not being able to recognize .AIF audio files generated on an Intel PC. He mentioned that "Sound Forge and other PC audio programs don't embed in their audio files the contextual information that the Macintosh needs to identify these files." (p. 49). I have much experience with manually setting the file creator and type on HFS volumes. Carr didn't mention the cool utility program called File Kit used for exactly this purpose. It allows the user to set the file type and creator, thus solving problems such as this. Apple should have included something like this with their operating system in the first place.

Andrew Munkres
via e-mail

Back

BIT Blasts

News from the World of Game Development

New Products

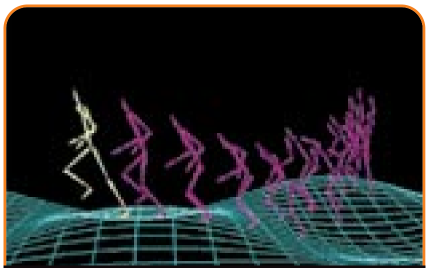
by Wesley Hall

Virtually Human

LEARNING MACHINES TECHNOLOGY GROUP (LMTG) recently released Nomad/Virtual Human SDK 1.2, a physical-simulation–based 3D animation system and database that will generate animation sequences for you.

If you've ever tried to incorporate physics into a game engine, you may have realized that modeling leg-locomotion is a challenge. LMTG has attempted to solve some of these difficulties with its SDK. Nomad features a library of 3D character animations based on the simulation of physics. The current release includes a quadruped class, a hexapod class, a bird class, and a humanoid bipedal class.

But the SDK is more than a simple library; it's an alternative to keyframed animation. You can specify weight, joint elasticity, and other physical attributes (including geometrical parameters) to customize the Nomad characters. Built-in sensors and actuators (virtual muscles), allow you to design feedback controllers for Nomad characters to achieve a number of adaptive behaviors. Each character has a built-in locomotion controller, and the simulation includes accurate foot-terrain collision detection. This allows characters to walk and run adaptively over uneven ground (which you're also free to design, along with custom obstacles and environment objects).


The Nomad/Virtual Human SDK has foot-terrain collision detection so characters can run over uneven ground.

The SDK contains other features such as real-time learning and adaptation, and the ability to access any necessary character geometry. The overall effect, LMTG claims, is a novel technology that "injects personality into the simulated characters." A professional edition of the SDK includes an AI engine for each character class so that new animation behaviors can be synthesized at run time. As a result, the characters appear to learn.

Nomad/Virtual Human SDK 1.2 is available for Windows NT, Windows 95/98, and DOS. A single developer's license includes technical support and a free version upgrade for $189. The professional edition of the SDK sells for $2,500.

■ Learning Machines Technology Group
Sunnyvale, Calif.
(408) 505-5398
http://www.lmtg.com

Easy 3D Audio

QSOUND LABS has just launched QCreator, a 3D audio authoring tool for consumers and professionals.

Using normal .WAV or .AIFF mono sound files, QCreator allows you to position sound within a three-dimensional space. When you open a mono sound file, QCreator automatically creates an Edit Window to display the contents of the file as a waveform graph. A special Pan Tool lets you decide where the sound will appear in space — and how the sound will move. The result is a customized sound file delivered in 3D.

QCreator will sells for $49.95, and is available for purchase from the QSound web site and RealNetworks' web site at http://www.real.com.

■ QSound Labs Inc.
Calgary, Alberta, Canada
(403) 291-2492
http://www.qsound.com

My First Tool Suite

PARADIGM ENTERTAINMENT has announced the release of VisKit. This is a low-budget 3D rendering tool and API in C++ for constructing 3D applications. The kit can be used for games and other applications including simulations, visualizations, interactive web applications, and real-time players.

Small and fast, Viskit's minimal runtime memory footprint and high-speed unrolled rendering loops help ensure improved real-time 3D application performance. Small footprint or no, VisKit still has more than 100 classes and provides all the tools and features necessary to develop high-performance 3D applications. Features include scene management, hierarchical state management, extensive vector and matrix support, multiple camera support, particle systems, texture and image import in most of the popular formats, automatic MIP-map generation, and more.

Viskit is layered on OpenGL and DirectSound, and is compatible with accelerated graphics. It sells for $99 with no additional royalties or licensing fees. The system requirements are a Pentium 166 or greater, Windows 95/98 or Windows NT 4.0, OpenGL, and Microsoft Visual C++ 5.0. An accelerated 3D graphics adapter is recommended.

■ Paradigm Entertainment Inc.
Dallas, Tex.
(972) 960-2301
http://www.viskit.com

Industry Watch

by Alex Dunne

PARIS-BASED PUBLISHER INFOGRAMES reported its results for the fiscal year ending June 30, and things look much rosier for the company this year. Infogrames posted revenue of $271 million, compared to $118 million last year, amounting to a $15.5 million profit. Last year, the company lost $6 million. The company attributed its success to several strong-selling titles, including V-RALLY, which sold over two million copies worldwide. Company president and CEO Bruno Bonnell said that the company's MISSION IMPOSSIBLE title, which was released in July 1998 for the N64, has reached the million-units–sold mark.

DIAMOND MULTIMEDIA SYSTEMS reported financial results for its third quarter, which ended September 30. The company's net revenues were up 34 percent to $123.2 million, from $92.0 million for the third quarter of 1997. The company incurred a net loss for the third quarter of $22.2 million, compared to a net loss of $2.5 million in the third quarter of last year. William Schroeder, president and CEO, said that the net loss was partly as a result of "price protection charges related to our Monster3D II product, whose supply, along with that of other Voodoo2-based graphics subsystems, exceeded demand during the third quarter."

AUREAL SEMICONDUCTOR, which builds the Vortex audio chip for sound cards (including Diamond's Monster Sound MX300) received a patent for a "Method and Apparatus for Efficient Presentation of High-Quality, Three-Dimensional Audio, Including Ambient Effects." It was the 19th patent for the company, and it has 39 additional audio patent applications in the works. The company's portfolio of interactive audio patents are related to all sorts of head-related transfer function (HRTF) technologies, including measurement HRTFs, rendering HRTFs, and implementation HRTFs. Creative Labs, however, hasn't been too thrilled with some of Aureal's marketing and advertising claims, apparently. CL filed a lawsuit against Aureal claiming "false advertising" and "unfair business practices." The Creative complaint centers primarily on a comparison chart prepared by Aureal highlighting feature differences between their technology and CL's SoundBlaster Live!

AMERICA ONLINE signed a multititle distribution agreement with online game developer, VR-1. In the deal, VR-1 will be providing AOL with three new pay-to-play games in 1999. The games included in the agreement include VR-1 CROSSROADS, NOMADS OF KLANTH, and THE S.A.R.A.C. PROJECT. Each of these games was built around the VR-1 Conductor technology platform, and each supports hundreds of players in a single arena. AOL has the option to license up to four additional VR-1 titles. VR-1 also developed the first two pay-to-play games for Microsoft's online gaming service (now called the MSN Gaming Zone) and has several international distribution agreements in place, including one with BT's WirePlay in the U.K.

HERE'S ANOTHER ENGINE for your growing list of "technologies for sale." 3DO announced plans to license the engine for REQUIEM: AVENGING ANGEL, which is slated for release in January 1999. The Requiem Engine, according to Trip Hawkins, "has a notable pedigree." OK, Trip. Actually, this probably isn't just hot air. The company's Cyclone Studios division, which is headed up by General Manager Helmut Kobler, also developed last year's UPRISING title and does indeed have good technical chops. We'll find out soon enough.

ACTIVISION AND VIACOM CONSUMER PRODUCTS (the licensing division of Paramount Pictures) announced a 10-year alliance that will allow Activision to develop and publish interactive entertainment titles based on the Star Trek franchise. Under the terms of the agreement, Activision has obtained the exclusive worldwide publishing rights for multiple platforms to all Star Trek properties, subject to the expiration of existing publishing agreements that Viacom Consumer Products currently maintains with respect to certain Star Trek properties. The deal also provides Activision with exclusive worldwide rights to any new television series or motion picture based on Star Trek. Activision intends to release its first Star Trek title based on Paramount Pictures' holiday '98 feature film, *Star Trek: Insurrection*. The announcement represents the first time that Viacom Consumer Products has entered into an exclusive umbrella deal with an independent entertainment software publisher.

SILICON GRAPHICS took a 10 percent stake in graphics chip maker Real 3D and said that the two companies would undertake cooperative marketing, technology, and business development. Under the terms of the agreement, Real 3D will be a preferred provider for future SGI workstation products. SGI will assist in the marketing of Real 3D products such as RealScan 3D, and the two companies will collaborate on future software technologies and initiatives. SGI's investment is worth between $20 million and $25 million to the Orlando, Fla., company, according to industry sources. Intel owns 20 percent of Real 3D, while Lockheed Martin, which founded Real 3D in 1996, owns the remainder.

Separately, SGI and Real 3D agreed to end litigation and entered into a royalty-free graphics patent cross-license. The agreement adds to licensing deals Real 3D already has with Lockheed Martin, Intel, and Sega. Looks to be an amenable way to end a lawsuit, don't you think? ■

UPCOMING EVENTS CALENDAR

GDC Roadtrip: Baltimore

BALTIMORE CONVENTION CENTER
Baltimore, Md.
December 10 & 11, 1998
$225
http://www.gdconf.com/1998/roadtrips

GDC Roadtrip: Boston

HYNES CONVENTION CENTER
Boston, Mass.
December 12 & 13, 1998
$225
http://www.gdconf.com/1998/roadtrips

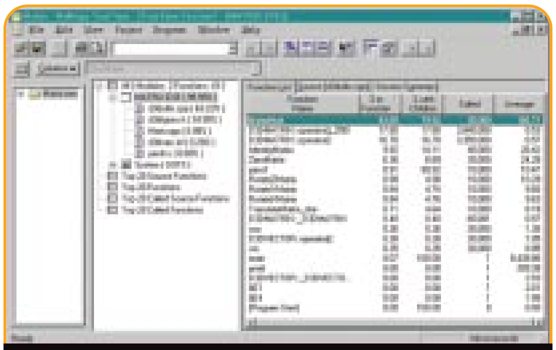


FIGURE 1. *TrueTime's initial interface.*

NuMega's TrueTime

by Ron Fosner

As Dan Teven mentioned last december, *Game Developer* magazine has reviewed a slew of profiling tools recently. This is no accident. We've made a very conscientious effort to cover these types of tools for a good reason. While everyone is busy jumping on the 3D hardware acceleration bandwagon, it's easy to forget the mantra of the game developer: quick, tight code. Just because everybody assumes that game players have 3D accelerators doesn't mean that we can code like we're building a word processor.

TrueTime integrates right into the Visual C++ development environment. This integration, combined with its excellent user interface, makes it easy to profile an application. The downside is that its suggested retail price is slightly higher than that of VTune, but you can get a free evaluation copy first from Compuware to determine whether it's worth the investment.

THE TRUETIME TRIALS. Installing TrueTime is easy, and once completed, the application adds a special toolbar to your Visual C++ IDE. With this toolbar in place and a project loaded, you have the option of instrumenting all or just part of your application. Profiling your application is as easy as selecting the "Rebuild with TrueTime" button. Your application recompiles with TrueTime instrumentation, links to TrueTime, then executes. Then, as your application runs, TrueTime records its timing information. When your application terminates, TrueTime brings up a window that displays various performance statistics for that profiling session.

I began testing TrueTime using my favorite example, a Direct3D Immediate Mode application that translates an object to some discrete 3D location. Along the way, I need to create a transformation matrix, which is generally composed of rotations about the x, y, and z axes, plus a translation. This code is part of an example that I use in my course on code optimization to illustrate how to use the matrix manipulation routines provided with Direct3D Immediate Mode. Unfortunately, this example creates a transformation matrix that is composed of a translation and three rotations in a mathematically inefficient manner (four matrix multiplies that are easily simplified), while also being inefficient with temporary storage. Straight out of the SDK, this code exhibits numerous inefficiencies. Because I already have an optimized version, I wanted to see how TrueTime would profile the original source.

I first created a test harness that simply called one particular code segment from the Direct3D application in a loop. With this working harness, it was simply a matter of pushing the TrueTime button in the IDE to recompile my application with TrueTime instrumentation. During this process, the application's source files get recompiled with invisible wrappers around all the function entry and exit points, which puts these into a TrueTime database. TrueTime can then measure the time spent in a function, as well as the time spent in each function that a particular function calls — right down to operating system calls.

When TrueTime finishes compiling your application, you run your instrumented application via another TrueTime button on the IDE, which launches the TrueTime monitor. The TrueTime monitor is a status window that shows the timing information for your application. TrueTime continues to monitor your program until it has terminated, at which point the initial TrueTime interface comes up (Figure 1).

The TrueTime interface is highly interactive, and allows you to quickly drill down into your program to find the areas that take the most time. The left-most column shown in Figure 1 is the Session Window and it simply displays information about the current TrueTime session. The interesting information is contained in the Filter Pane, located in middle column. From the highlighted bar in this column you can see that MATRIX.EXE, the test harness, takes up over 99 percent of the time. Just below that, you can see that "System" takes up the remainder. The item highlighted in the Filter Pane displays relevant data in the right pane, called the Session Data Pane. By selecting the executable, the Session Data Pane displays timing data for the entire program. The other top-level tabs in the Filter Pane allow you to filter the display by various function definitions.

In Figure 1, the Session Data Pane indicates that the function takes up the majority of time spent in the program. The "% in Function" column tells how much time was spent just in a particular function. The "% with Children" column shows how much time was spent in a function plus all of the other functions that were called while we were in this function. The last two columns describe how many times the function was called and the average execution time of the function. There are many other options that can be displayed in the Session Data Pane, including the first, maximum, and minimum execution times of the function, the average execution time including children, and so on.

The Session Data Pane illustrates TrueTime's excellent user interface. Double-clicking on the `MatrixMult` function in the Session Data Pane brings up the Function Details window shown in Figure 2. The two pie chart sections give details about both the functions that call `MatrixMult` and func-

Ron Fosner works on fast 3D rendering code at Data Visualization, is the author of OpenGL Programming for Windows 95 and Windows NT *from Addison-Wesley, and has taught course on code tuning at the Game Developer's Conference and the Win-Dev conferences.*

tions that **MatrixMult** calls itself. As you can see, only the **pass1** function calls **MatrixMult**. More interesting information can be gleaned by examining the child functions that **MatrixMult** calls. Additional details appear when the cursor hovers over a particular area. For instance, in Figure 2, my cursor is over the yellow slice of the pie chart and a pop-up informational window has appeared. The green slice indicates time spent in the function itself, the other slices indicate time spent in other functions called from **MatrixMult**. Of particular interest are the yellow and blue slices, which contain **D3DMatrix::Operator()**, and **D3DMatrix::Operator()_25f0**. The former is the compiler-generated name for the "( )" operator (the parentheses operator) for the **D3DMatrix** class from the DirectX SDK. The latter name is a bit obscure, but thankfully a button in the upper left corner of the window displays the source code of the currently selected function. Double-clicking on the **D3DMatrix::Operator()_25f0** line in the Function Details window brings up **D3DMatrix::Operator()_25f0**. If we click on the source button, the Session Data Pane will then display the source for **D3DMatrix::Operator()_25f0**. The left columns indicate the count and the execution time percentage for each line of the program. Unlike Intel's VTune, which disassembles source code and displays assembly instructions and performance information, TrueTime stops at the source code level.

With TrueTime, it's just as easy to profile a retail build as a debug build. If you do, that you'll find that the **operator()** measurements go away. However, if you're like me, you spend most of your time running the debug build, and it's easy to get a skewed view of your program's performance, because an often called utility class contains a high overhead in debug mode. Another nice feature of TrueTime is its ability to display the results of more than one timing session in multiple windows simultaneously.

Thanks to TrueTime's timing information, you can make modification to your code based on timing results, test new these new strategies, and verify their results. These are all features that should be present in any good profiler. But TrueTime make these tasks very easy thanks to its seamless integration with the development environment.

On the negative side, TrueTime *is* an instrumenting profiler. This means that every function call that you're profiling has to have a TrueTime wrapper around it before it's executed. This means your program will typically run two to three times slower than the non-instrumented version. Additionally, TrueTime by default won't measure any threads that are outside of your process. If you want a more "system-wide" profile, NuMega has something it calls Quantum Technology. Basically, it's simply a method of including or excluding time spent in other processes. However, this doesn't extend into system calls. This means that if you're thrashing virtual memory, you won't see it. But by far I feel the biggest negative is its price: $499 for the standalone version (up from $399 in September 1998, and now more expensive than VTune). With the current version, it's impossible to turn profiling on or off — it just runs while your application is running. I pointed out this fact to the folks at NuMega, and they informed me that they're building this feature into the next version.

Despite some of these problems, I feel that TrueTime is excellent for any programmer looking to find out how his or her application really spends its time. And, given the fact that TrueTime is also bundled with NuMega's DevPartner Studio (which includes utilities such as Bounds-Checker and SoftICE), it may be a better deal to purchase it within that suite of products if you don't already own them.

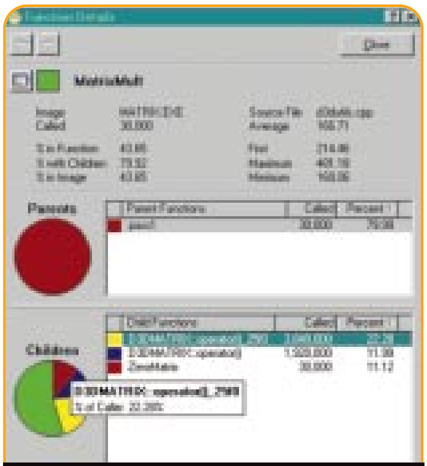


FIGURE 2. *TrueTime's Function Details window.*

So, how does TrueTime compare to VTune? Well, it's a little easier to use and the integration with the Visual C++ IDE is excellent. While VTune will force you to profile the entire system, TrueTime will profile just your application. Additionally, TrueTime just profiles source code, not the actual assembly-level statements. If you need to profile either the system or you do occasional assembly code optimizations, then you might want to stick with VTune. But if you just want to get up and running quickly, and you use Visual C++, TrueTime may be the product for you. ■

TrueTime 1.0 for Visual C++: ★★★★

Company:
Compuware (NuMega)
Nashua, N.H.
(603) 578-8400
http://www.numega.com

Price:
$499 (DevPartner Studio is $1,199).

System Requirements:
Windows 95/98, Windows NT 4. Requires Visual C++ 4.2 or higher, Visual J++ 6.0 or Visual Basic 5.0. Requires a Pentium processor, 32MB RAM, and approximately 17MB of disk space.

Pros:
1. TrueTime features an easy-to-use interface for viewing timing information.
2. It integrates well within Visual C++
3. TrueTime makes it easy to profile release builds

Cons:
1. TrueTime only works with Microsoft development tools.
2. It slows down the speed of your application while it's testing it.
3.It's slightly (about $70) more expensive than Intel's VTune. (These tools are getting a bit pricey.)
4. TrueTime doesn't drill down into assembly like VTune can.

by Jeff Lander

Mighty Morphing Mesh Machine

Most game people have played around with those programs that will morph between two graphic images. They allow you to take a picture of your brother and a picture of a freckle-faced baboon, play with the sliders, and enjoy hours of endless amusement.

Now, think about the fun you could have morphing between a 3D model of your brother and the monkey — that could make your week. Most 3D animation packages will allow you to do this, but what you might really be looking for is a real-time 3D demo featuring your brother in some sort of failed lab experiment that you could mercilessly blow away. Or something like that…

To engage in this advanced level of fun, you need a method for morphing between two 3D shapes. This technique is not only amusing, but also quite useful. 3D morphing can help create organic animation that would otherwise be difficult to develop.

Morphing and Real-Time 3D Animation

Another use for 3D morphing is smoothing between keyframes. In games such as QUAKE 2, you may read that the animation in-betweens are interpolated. I have discussed before how in QUAKE (and many 3D action games) each frame of animation is actually represented by an individual mesh. By sequencing through those meshes, the illusion of animation is created. However, the original object frames are created at a set frame rate, say 10 frames per second (FPS). This means that the smoothest those animations can play back is at that original 10 FPS. If, for example, the engine is actually displaying at 60 FPS, each frame of character animation is held for six frames. That's quite a wasted opportunity. Smoother animation could be achieved by morphing from one frame to the next over those six frames. That is exactly what the "feature hypers" (you know, the feature-happy marketing dudes) are talking about when they talk about interpolated in-betweens. But what exactly is being interpolated?

LERP = Morph

This is an important equation in the game programmer's arsenal. Many programmers who have been working in 3D for some time take this for granted as widely known information. But judging by the mail I get and the comments I see in the public forums, there are many individuals who aren't up-to-speed on how morphing works. The secret to object morphing is that the only thing that changes between the two models is the vertex positions in the object (a little white lie — I'll get to the truth later). When you morph between 3D objects, you are doing a "linear interpolation," or LERP, between the vertex positions in the two objects. For this technique to work, the two models you are morphing between must have identical vertex counts, and the vertices must correspond to each other. This means that vertex 1 on the first model should end up in the position of vertex 1 in the second model. The easiest way to make sure that the models are created correctly is to actually create the second model by moving the vertices in the first model into the shape you want. That way, the models will interpolate exactly the way you created them. By now you have the models and want to morph between them. The formula for a LERP between two values is

```
InBetween = Value1 + ((Value2 - Value1) *
  lerpValue);
Where lerpValue is a float between 0 and 1.
```

Now, this formula needs to be applied

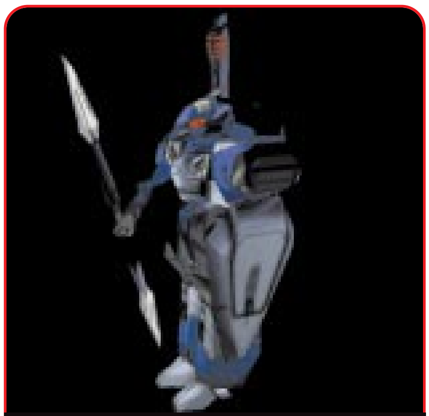


FIGURE 1A. *Your model at rest, for the first frame of animation.*

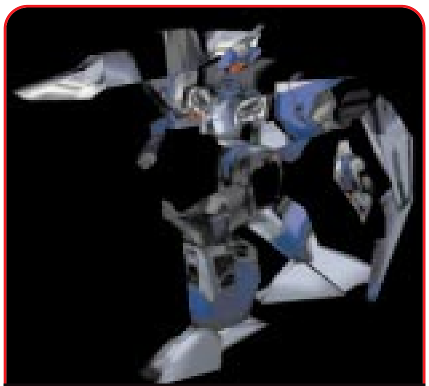


FIGURE 1B. *The same model, frame two. You'll get smoother animation by morphing between frames.*

Morphing between a lounging postion on the beach and a slave position at his desk, Jeff can be found at Darwin 3D working on real-time game technology. Email him at jeffl@darwin3d.com with suggestions for the next keyframe.

to every parameter that will vary during the morph. For a 3D vertex coordinate, those parameters would be the x, y, and z values for that point. This is where I explain the truth behind that little white lie. There may be more parameters than just the coordinates which you wish to interpolate for an individual vertex. If your game engine supports real-time lighting and your model data contains vertex-normal information, for example, you may want to interpolate the normal values as well. Also, if your model contains vertex color information, an interesting effect may be created by interpolating the color.

What about textures? If your 3D model contains texture coordinates, you may want those coordinates to change over time as well. However, there are many possible pitfalls associated with morphing textures. Small changes in UV coordinates can cause a major shift in the appearance of a model that may not be desired. Likewise, it may be more interesting to change the texture completely for the second position. This complicates things a bit. However, if the UV coordinates stay constant throughout the morph and only the texture changes, image processing techniques can help. By using a 2D dissolve to create a blended image between both textures, this will smoothly change along with the model. These blended textures could either be prebuilt as a texture animation or actually created on-the-fly, if possible. Multitexture hardware can provide a hardware-accelerated method for blending between two textures via methods such as the `D3DTOP_BLENDFACTORALPHA` and `D3DTOP_BLENDDIFFUSEALPHA` in DirectX 6 (See "Multitexturing in DirectX 6," *Game Developer*, September 1998). With multitexture hardware

LISTING 1. *The morph code.*

```
#define LERP(a,b,c)  (a + ((b - a) * c))
///////////////////////////////////////////////////////////////////////////////
// Procedure: morphModel
// Purpose:   Does the Morph for the Model
// Arguments: Pointer to main bone
///////////////////////////////////////////////////////////////////////////////
GLvoid COGLView::morphModel(t_Bone *curBone)
{
/// Local Variables ///////////////////////////////////////////////////////////
   int loop,pointloop;
   float *dest,*src1,*src2,ratio;
///////////////////////////////////////////////////////////////////////////////
   if (curBone->visualCnt > m_curVisual)
   {
      src1 = curBone->visuals[0].vertexData;      // FRAME 1
      src2 = curBone->visuals[1].vertexData;      // FRAME 2
      dest = curBone->visuals[2].vertexData;      // DESTINATION FOR MORPHED FRAME
      ratio = m_Slider->GetSetting();             // GET MORPH VALUE (0 - 1)
      // LOOP THROUGH THE VERTICES
      for (loop = 0; loop < curBone->visuals[0].triCnt * 3; loop++)
      {
         // GO THROUGH EACH ELEMENT IN THE VERTEX STRUCTURE
         for (pointloop = 0; pointloop < curBone->visuals[0].vSize; pointloop++)
         {
            // THE NEW POSITION IS A LERP BETWEEN THE TWO POINTS
            dest[(loop * curBone->visuals[0].vSize) + pointloop] =
               LERP(src1[(loop * curBone->visuals[0].vSize) + pointloop],
                  src2[(loop * curBone->visuals[0].vSize) + pointloop],ratio);
         }
      }
   }
}
// morphModel
```

TABLE 1. *A comparison of different formats.*

Format	Used By	Exported By	Imported By	Normals	Poly Color	Vertex Color	UV Coords	Animation	Hierarchies	AscII	Ease of Use (10 = Easy)
.3DS	3DS R4	1,2,8	1,2,8	X	X	X	X	X	X		6 (w/ KTX lib)
.3DS AscII	3DS R4	1	1	X	X	X	X	X	X	X	6
.DXF	Autocad	1,2,3,4,6,7,8	1,3,5,6,7,8	X	X					X	7
LWOB	Lightwave	5	5	X	X			X			5
.OBJ	Wavefront	2,3,6,7,8	2,3,6,7,8	X	X	X	X	X			10
Game Exchange	Nichimen	7	7	X	X	X	X	X	X	X	9
.HRC	Softimage	8	8	X	X	X	X	X	X		1
MAX	3DS MAX	2	2	X	X	X	X	X	X		0 (Max only)
Maya ASCII	Maya	6	6	X	X	X	X	X	X	X	8
VRML	VRML	2,7,8	2,7,8	X	X	X	X	X	X	X	6
.X	Direct X	2,7,8	2,7,8	X	X		X	X	X	X	7

Program Names:		
	1	.3DS R4
	2	.3DS MAX
	3	Alias
	4	Hash
	5	Lightwave
	6	Maya
	7	Nichimen
	8	Softimage

becoming more common, this could be an area to add value to your hardware-accelerated application.

This algorithm is actually very easy to get up and running. You can see the code for a 3D morph in Listing 1. One easy optimization to make would be to precalculate the deltas between each parameter to remove a subtraction operation. In the case of an animation system, predividing the deltas by the number of desired in-betweens would turn it into a pure addition operation.

That's all there is to 3D morphing. Given how easy a 3D morph actually is to implement, I wonder why we don't see it more in real-time 3D games. There are many uses for this technology. Beyond creating in-betweens for character animation, morphing is an excellent way to change the shape of characters. It can also be used to create facial animation and other special effects. Hopefully, we will start to see more in the next generation of real-time projects.

Getting Your Model Data

So, you're ready to start morphing every object you can get in your hands. That brings up an important question. How do you get your hands on model data? When it comes to game companies with staff artists and tool programmers, many rely on high-end animation packages with SDKs. These toolkits allow the programmers to write plug-ins that allow them to get to the data directly. This is not always possible. Some programmers do not have the money, time, or experience to get models this way. The other option is to use a public 3D file format. The ideal file format contains all the information that the application requires and is easy to use. The ideal format is also public, meaning that information is publicly available describing the format. Code samples are even more desirable.

So, which file format is the best one for you? Table 1 (see page 16) contains a list of several file formats along with some information about them. This is by no means comprehensive, but the list offers a glimpse of what's out there. Ease-of-use is an opinion gathered from my own experience and from discussing it with other programmers.

The key to finding a format to support in a custom tool is to pick a format that contains all the information that is critical to your application. If your game engine uses vertex coloring, make sure the format supports that feature. It's also helpful to pick a format that is supported by your or your artist's favorite art tool. There are commercial 3D file converters available, but they add cost and an additional step to the production process. Also, using an ASCII format makes checking your data and debugging the process much easier.

I have found that one of the easiest to use and most widely supported formats is the Wavefront .OBJ format. It contains support for UV coordinates as well as for vertex normals. The format is so easy that you really don't even need a spec to write a file loader. Listing 2 shows a simple cube in .OBJ format. Lines that begin with the pound sign **#** are comments and can be ignored. The initial `o cube1` defines the name of the object. That is followed by the `mtllib cube.mtl`. This line describes a file that contains material information about the object. But more on that later.

The next block of information actually describes the vertices. Each line starts with a `v` and is followed by the x, y, and z coordinate values. The comment at the end actually tells you the number of vertices. But, that doesn't seem to be a standard feature of this format, so you shouldn't count on it.

The `vn` block gives the x, y, z, values for the normals in the model and the `vt` block describes the texture coordinates. The texture coordinates can be either two coordinates (u and v) or three (u, v, and w). For most real-time applications, however, the `w` value could be ignored. A 3D model may have normals or texture coordinates, or both, or neither, but it obviously

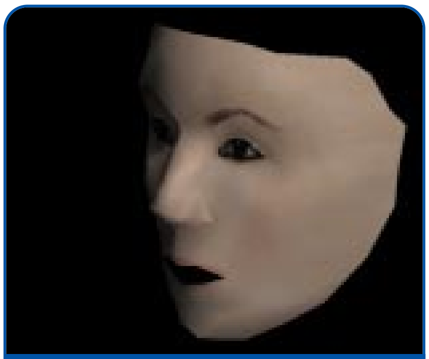


FIGURE 2A. *Your "brother" mapped onto a 3D model.*

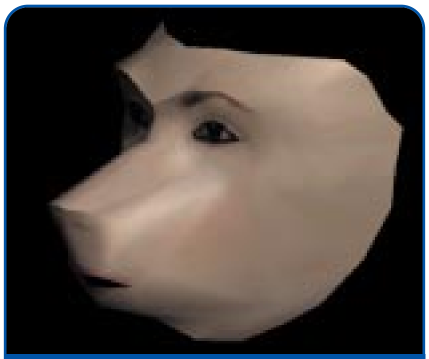


FIGURE 2B. *The original figure morphed into a baboon.*

LISTING 2. *An OBJ Cube.*

```
o cube1

mtllib cube.mtl

v -5.000000 -5.000000 -5.000000
v -5.000000 -5.000000 5.000000
v -5.000000 5.000000 -5.000000
v -5.000000 5.000000 5.000000
v 5.000000 -5.000000 -5.000000
v 5.000000 -5.000000 5.000000
v 5.000000 5.000000 -5.000000
v 5.000000 5.000000 5.000000
# 8 vertices

vn  -1.000000 0.000000 0.000000
vn  0.000000 0.000000 1.000000
vn  1.000000 0.000000 0.000000
vn  0.000000 0.000000 -1.000000
vn  0.000000 -1.000000 0.000000
vn  0.000000 1.000000 0.000000
# 6 normals

vt  0.000000 0.000000
vt  0.000000 1.000000
vt  1.000000 0.000000
vt  1.000000 1.000000
# 4 texture vertices

usemtl mat1_FACE
f 1/2/1 2/4/1 4/3/1
f 1/2/1 4/3/1 3/1/1
f 2/2/2 6/4/2 8/3/2
f 2/2/2 8/3/2 4/1/2
f 6/2/3 5/4/3 7/3/3
f 6/2/3 7/3/3 8/1/3
f 5/2/4 1/4/4 3/3/4
f 5/2/4 3/3/4 7/1/4
f 5/2/5 6/4/5 2/3/5
f 5/2/5 2/3/5 1/1/5
f 3/2/6 4/4/6 8/3/6
f 3/2/6 8/3/6 7/1/6
# 12 elements
```

must have the vertex values.

The final block in the file begins with the **usemtl mat1_FACE**. This says to the loader, from now on all faces defined should use the **mat1_FACE** material. This material is defined in the **cube.mtl** file. All lines that begin with an **f** describe a face in the model. Each face can be composed of multiple vertices. Each face is not required to have the same number of vertices. However, because it is more efficient for 3D hardware if all faces have the same number of vertices, I make sure this is the case. I could tessellate the face to triangles at run time, but this is really easy to do in the modeling program. Therefore, I just make it a requirement that all models are triangulated before exporting. A pop-up box in the loader can warn users that a face is not triangulated.

In the face statement each vertex is defined by three elements separated by forward slashes that describe the vertex, texture coordinate, and normal for that face. The values are indices into the list of elements already defined. It's important to notice that these indices are one-based instead of zero-based. In a file that only has vertex coordinates (and not normals or texture vertices), the vertex index will be followed by two slashes as in **f 1// 2// 3//**. Likewise, if there is a vertex and a normal, the format is **f 1//1 2//2 3//3** and so on. Each vertex in the line is separated by a space.

You can see a material file in Listing 3. The file can describe multiple materials. Each one begins with **newmtl** and the name of the material in this case **mat1_FACE.** The next lines **Ka**, **Kd**, and **Ks** respectively describe the ambient, diffuse, and specular color for the material. The **Ns** term describes the specular highlight. I have never had a need for the**Ni** and **illum** term, though they are there if you want them. Finally, the **map_Kd** describes the diffuse map applied to the object. In other words, this is the texture map that should be applied to the surface. I use this name as the name of the file loaded by the application. I just convert the image to a .TGA or .BMP file to make use of existing file loading code.

See there, I said it was an easy format. Actually there are other blocks that may be useful in a real-time simulation that are not in my sample file. The **g** command allows faces to be grouped together so the file can contain multiple objects even in a hierarchy. This is definitely handy when working with hierarchical characters.

Writing a .OBJ File Loader

The MFC **CString** class makes string manipulation much easier than in basic C. For custom tools, this means loading ASCII file formats is easier than ever. My strategy is to load a line of text from the file and then break it up into a string of words in a **CStringArray** structure. If you haven't used the **CString** class, it will bring back fond memories of BASIC string handling.

I don't want to go through the entire .OBJ loader here. You can just grab it off the web site (http://www.gdmag.com). However, my strategy for loading an .OBJ file is to pass through the file once, determining how many vertices, normals, and texture coordinates for which I need to allocate space. Then, all the actual coordinate values are simple to load in. The only tricky part comes in when I want to load in the face indices. You can see how I approached that in Listing 4.

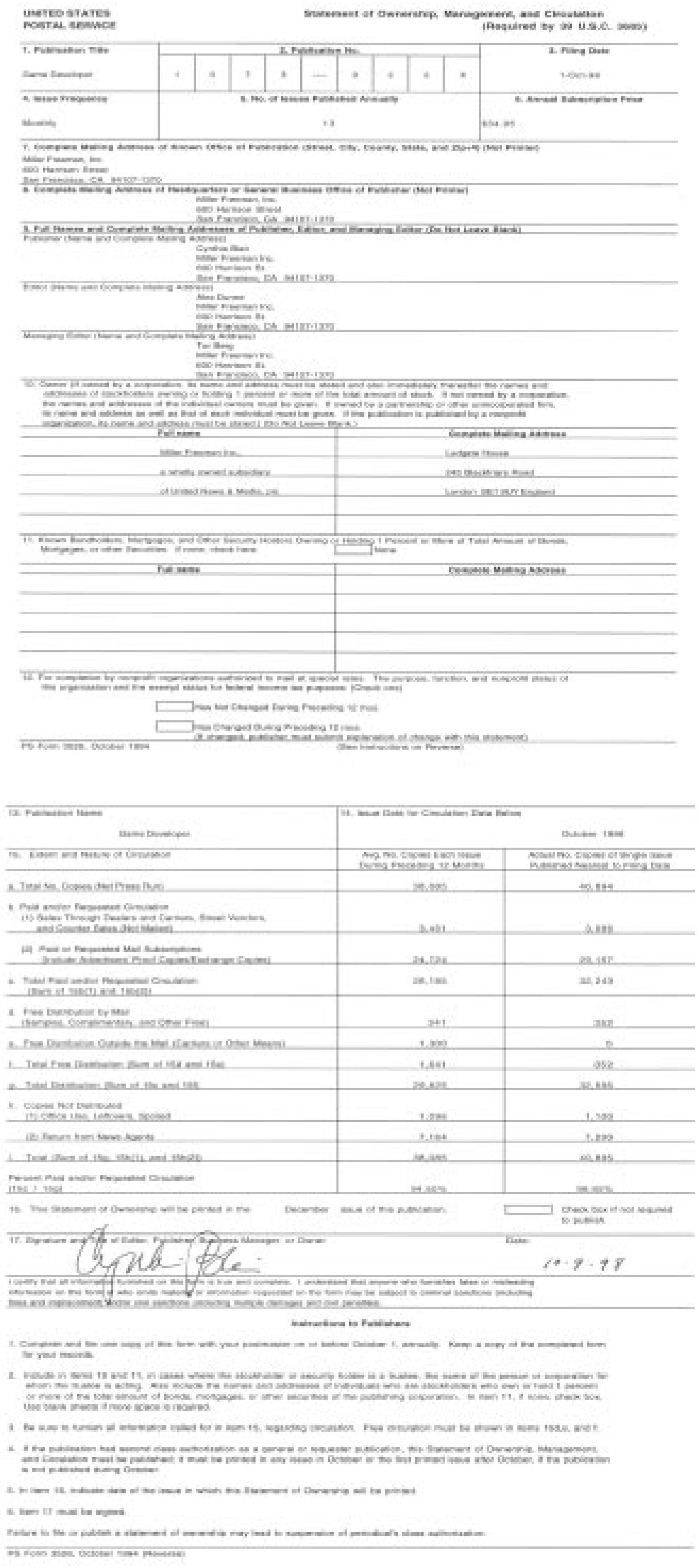
The point of this loader is not for me to show highly optimized, well formulated code samples for loading these files. My code here certainly is not fine-tuned in any sense of the word. The great thing about creating production tools is that, unlike almost all other

LISTING 3. *MTL file for the Cube.*

```
newmtl mat1_FACE
Ka 0.5000 0.5000 0.5000
Kd 0.7000 0.7000 0.7000
Ks 1.0000 1.0000 1.0000
Ns 50.0000
Ni 1.0000
illum 2
map_Kd FACE.pic
```

LISTING 4. *Handling a face line in an .OBJ.*

```
///////////////////////////////////////////////////////////////////////////////
// Procedure: HandleFace
// Purpose:   Handles the Face Line in an OBJ file.  Extracts index info to
//            a face Structure
// Arguments: Array of words from the face line, place to put the data
// Notes:     Not an Official OBJ loader as it doesn't handle more then
//            3 vertex polygons.  This only handles Triangles
///////////////////////////////////////////////////////////////////////////////
void HandleFace(CStringArray *words,t_faceIndex *face)
{
/// Local Variables ///////////////////////////////////////////////////////////
   int loop;
   CString temp;
   CString vStr,nStr,tStr;     // HOLD POINTERS TO ELEMENT POINTERS
   int nPos,tPos;
///////////////////////////////////////////////////////////////////////////////
   // LOOP THROUGH THE 3 WORDS OF THE FACELIST LINE, WORD 0 HAS 'f'
   for (loop = 1; loop < 4; loop++)
   {
      temp = words->GetAt(loop);          // GRAB THE NEXT WORD
      // FACE DATA IS IN THE FORMAT vertex/texture/normal
      tPos = temp.Find('/');              // FIND THE '/' SEPARATING VERTEX AND TEXTURE
      vStr = temp.Left(tPos);             // GET THE VERTEX NUMBER
      temp.SetAt(tPos,' ');               // CHANGE THE '/' TO A SPACE SO I CAN TRY AGAIN
      nPos = temp.Find('/');              // FIND THE '/' SEPARATING TEXTURE AND NORMAL
      tStr = temp.Mid(tPos + 1, nPos - tPos - 1);     // GET THE TEXTURE NUMBER
      nStr = temp.Right(temp.GetLength() - nPos - 1); // GET THE NORMAL NUMBER
      face->v[loop - 1] = atoi(vStr);        // STORE OFF THE INDEX FOR THE VERTEX
      face->t[loop - 1] = atoi(tStr);        // STORE OFF THE INDEX FOR THE TEXTURE
      face->n[loop - 1] = atoi(nStr);        // STORE OFF THE INDEX FOR THE NORMAL
   }
}
///// HandleFace //////////////////////////////////////////////////////////////
```



UNITED STATES
POSTAL SERVICE

Statement of Ownership, Management, and Circulation
(Required by 39 U.S.C. 3685)

1. Publication Title	2. Publication No.									3. Filing Date
Game Developer	1	0	7	3	—	9	2	2	X	1-Oct-98

4. Issue Frequency	5. No. of Issues Published Annually	6. Annual Subscription Price
Monthly	12	$34.95

7. Complete Mailing Address of Known Office of Publication (Street, City, County, State, and Zip+4) (Not Printer)
Miller Freeman, Inc.
600 Harrison Street
San Francisco, CA 94107-1370

8. Complete Mailing Address of Headquarters or General Business Office of Publisher (Not Printer)
Miller Freeman, Inc.
600 Harrison Street
San Francisco, CA 94107-1370

9. Full Names and Complete Mailing Addresses of Publisher, Editor, and Managing Editor (Do Not Leave Blank)
Publisher (Name and Complete Mailing Address)
Cynthia Blair
Miller Freeman Inc.
600 Harrison St.
San Francisco, CA 94107-1370
Editor (Name and Complete Mailing Address)
Alex Dunne
Miller Freeman Inc.
600 Harrison St.
San Francisco, CA 94107-1370
Managing Editor (Name and Complete Mailing Address)
Tor Berg
Miller Freeman Inc.
600 Harrison St.
San Francisco, CA 94107-1370

10. Owner (If owned by a corporation, its name and address must be stated and also immediately thereafter the names and addresses of stockholders owning or holding 1 percent or more of the total amount of stock. If not owned by a corporation, the names and addresses of the individual owners must be given. If owned by a partnership or other unincorporated firm, its name and address as well as that of each individual must be given. If the publication is published by a nonprofit organization, its name and address must be stated.) (Do Not Leave Blank.)

Full name	Complete Mailing Address
Miller Freeman Inc.,	Ludgate House
a wholly owned subsidiary	245 Blackfriars Road
of United News & Media, plc	London SE1 9UY England

11. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages, or other Securities. If none, check here ☐ None

Full name	Complete Mailing Address

12. For completion by nonprofit organizations authorized to mail at special rates. The purpose, function, and nonprofit status of this organization and the exempt status for federal income tax purposes: (Check one)

☐ Has Not Changed During Preceding 12 mos.
☐ Has Changed During Preceding 12 mos.
(If changed, publisher must submit explanation of change with this statement)

PS Form 3526, October 1994 (See Instructions on Reverse)

13. Publication Name	14. Issue Date for Circulation Data Below	
Game Developer		October 1998
15. Extent and Nature of Circulation	Avg. No. Copies Each Issue During Preceding 12 Months	Actual No. Copies of Single Issue Published Nearest to Filing Date
a. Total No. Copies (Net Press Run)	38,305	40,844
b. Paid and/or Requested Circulation (1) Sales Through Dealers and Carriers, Street Vendors, and Counter Sales (Not Mailed)	3,451	3,086
(2) Paid or Requested Mail Subscriptions (Include Advertisers' Proof Copies/Exchange Copies)	24,734	29,157
c. Total Paid and/or Requested Circulation (Sum of 15b(1) and 15b(2))	28,185	32,243
d. Free Distribution by Mail (Samples, Complimentary, and Other Free)	341	352
e. Free Distribution Outside the Mail (Carriers or Other Means)	1,300	0
f. Total Free Distribution (Sum of 15d and 15e)	1,641	352
g. Total Distribution (Sum of 15c and 15f)	29,826	32,595
h. Copies Not Distributed (1) Office Use, Leftovers, Spoiled	1,295	1,100
(2) Return from News Agents	7,184	7,200
i. Total (Sum of 15g, 15h(1), and 15h(2))	38,305	40,895
Percent Paid and/or Requested Circulation (15c / 15g)	94.50%	98.92%

16. This Statement of Ownership will be printed in the December issue of this publication. ☐ Check box if not required to publish.

17. Signature and Title of Editor, Publisher, Business Manager, or Owner [illegible] Date: 10-9-98

I certify that all information furnished on this form is true and complete. I understand that anyone who furnishes false or misleading information on this form or who omits material or information requested on the form may be subject to criminal sanctions (including fines and imprisonment) and/or civil sanctions (including multiple damages and civil penalties).

Instructions to Publishers

1. Complete and file one copy of this form with your postmaster on or before October 1, annually. Keep a copy of the completed form for your records.
2. Include in items 10 and 11, in cases where the stockholder or security holder is a trustee, the name of the person or corporation for whom the trustee is acting. Also include the names and addresses of individuals who are stockholders who own or hold 1 percent or more of the total amount of bonds, mortgages, or other securities of the publishing corporation. In item 11, if none, check box. Use blank sheets if more space is required.
3. Be sure to furnish all information called for in item 15, regarding circulation. Free circulation must be shown in items 15d, e, and f.
4. If the publication had second class authorization as a general or requester publication, this Statement of Ownership, Management, and Circulation must be published; it must be printed in any issue in October or the first printed issue after October, if the publication is not published during October.
5. In item 16, indicate date of the issue in which this Statement of Ownership will be printed.
6. Item 17 must be signed.

Failure to file or publish a statement of ownership may lead to suspension of periodical's class authorization.

PS Form 3526, October 1994 (Reverse)

coding in the game industry, the focus is not directly on the speed of the routine. It doesn't have to be very fast. In fact, when working on tools, it is often better to sacrifice speed for clarity. Often, a tool that you create now and will have a long life and pass through many hands after you. This is not usually true of the actual game code (no matter what your producer may want to think) as most core game routines are rewritten for each project. Tools tend to linger.

The point of showing these types of file loading routines is to demonstrate how easily it can be done. I talk to programmers who say they understand the algorithms but have no models with which to work. They are not comfortable writing a 3D Studio MAX plug-in to get models. I hope that this shows that very commonly used 3D file formats can be easily integrated into your own production tools. Once you build up a library of routines like this, you can easily get models to work within your game applications. If you create a loader for a commonly used format, there are models everywhere that you can use. For actual production applications, I tend to create custom binary formats because they are compact and exactly tuned to the application. But by loading a general format, you can easily make a file converter.

This month, I have provided an application that allows you to load two .OBJ files that have identical vertex arrangements. You then can use the slider to morph between the two. The program can handle objects with and without texture mapping. Grab it at the *Game Developer* web site at http://www.gdmag.com. Special Thanks to Bennie Terry for providing the model, and to Eddie Smith for providing the texture map for the LAG14 character from their real-time action title, ARIES PROJECT. ■

FOR FURTHER INFO

You can find a list of 3D file formats and their specs at:
http://www.cica.indiana.edu/graphics/3D.objects.html

Rule, Keith. *3D Graphics File Formats : A Programmer's Reference.* Addison-Wesley, 1996.

Covers the .OBJ, .3DS, and VRML file formats, among others.

So, You Want to Make a Console Game?

You're not alone. When you compare the $900 million in revenue generated by the PC gaming industry to the $25 billion brought in by the console market, it's not hard to see why. High volume sell through, a virtual absence of hardware compatibility issues, and a thriving rental industry are leading more and more PC developers to release console versions of their most popular titles.

This month, we're going to switch gears and examine some of the aspects of developing content for the two console powerhouses, the Sony PlayStation, and Nintendo's N64. With the help of two local developers, Crave Entertainment and Snowblind Studios, we'll look at the problem from the standpoint of "How They Did It" and try to pre-emptively address some of the technical dos and don'ts you're likely to come across. For those console veterans out there, consider this a refresher course and a discussion of what some of your peers in the industry are doing.

Sony's PlayStation

To start off, we'll look at the aging-but-still-popular Sony PlayStation. With more than 30 million units on the worldwide market (as of February 1998), the PlayStation user base is staggering. And with games such as FINAL FANTASY VII, GRAN TURISMO, and the Resident Evil franchise, the platform shows no sign of slowing down. To round out the PlayStation perspective, we've interviewed the SHADOW MADNESS team over at Crave Entertainment.

THE GAME. SHADOW MADNESS (due out in Spring 1999) is Crave's attempt to break into the lucrative RPG franchise business. The game play follows the format perfected by Squaresoft's FINAL FANTASY VII with real-time 3D (RT3D) characters exploring static backgrounds, the everpresent story-telling FMVs, and fully immersive RT3D combat arenas.

Production on the title started back in mid-1997, when, after an initial design phase, the art team was assembled and work began on the FMVs and the more than 800 interactive backgrounds for the game. The engineering team was assembled some months later, when the art path and toolset for actually getting data into a 3D engine was created.

THE SONY APPROVAL PROCESS. Every PSX game that you see on the shelf has Sony's stamp of approval on it. This means it has met the rigorous standards of content and quality that Sony uses to judge games released on it's platform. How does this work? Well, first of all, there are a limited number of slots given out each year to developers. Some of the major players, such as Eidos and Square for example, are allotted slots carte-blanche, based on past performance. But most developers must first go through an initial design and planning phase to get Sony to buy-off on their game concept. During the development lifecycle, Sony keeps tabs on the product's content and quality, and in most cases, has the authority at any time to reject the product based on it's failure to meet their company standards.

Sound strange? Well, imagine if, when the time came to get your alpha milestone approved for your PC title , that you had to submit a version to Intel or 3Dfx along with the one you send up to your publisher. In essence , that's what it comes down to.

DEVELOPMENT STRATEGY. Early on, the developers at Crave chose the PSX as the target platform. One of the many reasons for going this route was the PSX's ability to seamlessly display full-length prerendered FMVs, a tool the developers planned to exploit fully as part of the story-telling process in the RPG. With this exceptional functionality came a slew of technical challenges, which had to be addressed before the game went into full production.

The most challenging aspect of developing on the PSX was the limited amount of storage space (2MB of RAM) available to the artists — texture maps,

SHADOW MADNESS
Category: RPG
Format: PlayStation
Developer: Crave Entertainment
Publisher: Crave Entertainment
Release Date: Q1 1999

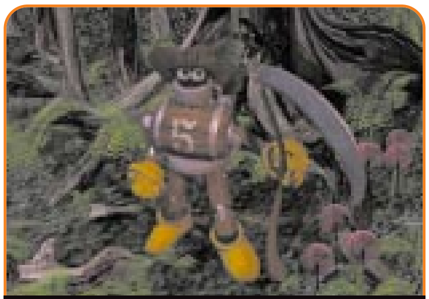


FIGURE 1. SHADOW MADNESS.

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

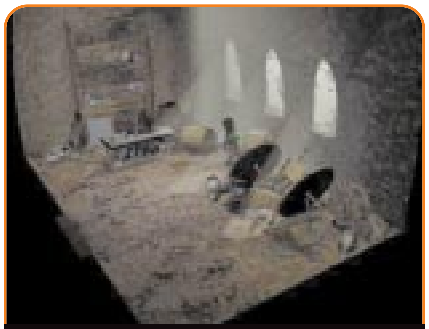


FIGURE 2. *Weeble mode.*

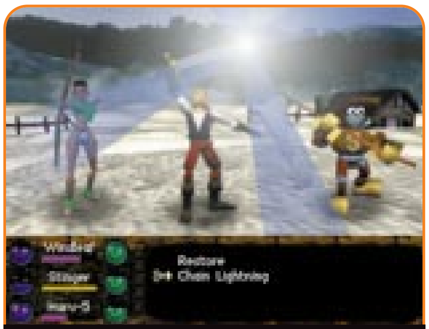


FIGURE 3. *Battle Mode*

color look-up tables, animations, geometry, and static backgrounds all have to fit in this window of memory. To create an immersive gaming experience then, the folks at Crave opted for two different modes of play.

Weeble mode (Figure 2), is used by the player when navigating the environment. This mode is characterized by low-resolution character models animated on a scrolling or static background. Most of the memory is taken up by high-resolution background images and overlays.

The hurdles in weeble mode dealt mainly with cinematic feel and camera control, because this is where most of the non-FMV storytelling takes place. Each static background, then, became equivalent to a stage on a film set. By taking advantage of overlays for creating parallax, the team was able to create seemingly 3D environments using only static backgrounds. To do this, the team incorporated knowledge gained from the film industry — camera control and field-of-view manipulation became key, with each artist responsible for creating the correct mood in his or her area.

In weeble mode, the characters take up a relativley small percentage of the viewing area. Consequently, they can be lower-resolution (around 100 to 140 polygons) and use fewer textures, thereby allowing more characters on screen. Additionally, their animations can be of lower complexity. Finally, the savings in texture space, polygon storage, and character animation translates directly into additional space for background images and overlays. Determining the right balance of textures, geometry, and animations early on proved crucial to completing the project in a timely manner.

Battle mode, as illustrated in Figure 3, is used for close encounters with NPCs or enemy characters. Battle mode uses fully interactive 3D environments with higher-resolution character models.

Battle mode is closer to what players have come to expect in RT3D. Here, the focus shifts from telling the story to dynamic real-time action. Fully immersive, interactive environments, with an active camera that can swivel and pan around the battle arena, characterize this phase of the game. In battle mode, the characters are higher-resolution, and the environments are composed mainly of textured 3D objects, with only cursory background images for environmental texture mapping. Because the characters take up much more screen space in battle mode, they need to have a proportional amount of complexity associated with them. Here, the characters have roughly twice the number of polygons (300 to 400 polygons each) and a much smoother set of animations. (Crave's engine took advantage of the new .HMD format for PSX, which allows real-time interpolation between keyframes, giving a much smoother feel with a lot fewer stored keyframes.)

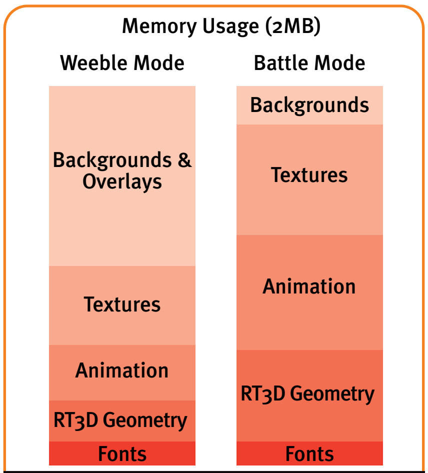


FIGURE 4. *What fits into 2MB ?*

TECHNICAL TRICKS AND CHALLENGES. Overall, the problem that the artists kept running into was how to efficiently use the 2 MB of RAM they had in their budget. Because the background images couldn't really be optimized, the bulk of tweaking time was spent on the 3D characters and geometry.

The characters in the game were expensive mainly because of the textures and animations associated with them. Consequently, this is where most of the effort was put to try to streamline the already expensive process.

VERTEX COLORS INSTEAD OF TEXTURES. Vertex colors were used in addition to, and in place of, textures for each character (for more on vertex coloring, see "Painting With Vertex Colors" *Game Developer,* November 1998). Colors were applied directly in 3D Studio MAX with the vertex coloring tool. Figure 3 shows an example of characters in the game that were painted almost entirely using vertex coloring.

.HMD AND KEYFRAME INTERPOLATION. When it came to determining a format in which to store and access the 3D geometry and animation information, the developers had three options: MIME, .TOD/.TMD, and the new .HMD format. Each had its respective advantages and disadvantages, but .HMD was the only unproven format because it was new and there was relatively little documentation on it. The one big advantage that .HMD had was the freedom it provided for the programmers to access the keyframe data. This allowed them to generate algorithms for real-time interpolation, so that an animation that

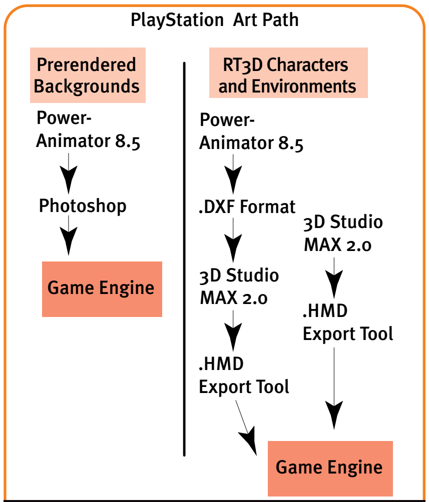


FIGURE 5. *The PlayStation art path.*

classically would have used 30 frames could be created with 7 keyframes. Having an engine that provides for real-time interpolation between keyframes means that each animation sequence can be made up of very few actual data-points, with the engine providing the smooth transition between keyframes and poses. Overall, this meant a much smaller allocation of expensive memory space for any given animation, and a larger total number of animations allowed for each character. According to Crave, the game could not have been finished without using the new .HMD format, which to date has not been used on any published title.

TOOLS. Once the technical hurdles were addressed, the team had to choose tools. Two main goals needed to be achieved: rapid generation of high-quality prerendered PowerAnimator content, and fully-textured and animated RT3D characters. For this task, the developers chose PowerAnimator 8.5 and 3D Studio MAX 2.0. The proven high-quality rendering capabilities of PowerAnimator were augmented by interfacing with Renderman.

Figure 5 shows the art path Crave used to create content for their title. Most RT3D modeling was done within 3D Studio MAX, with the bulk of cinematics and static backgrounds done with NURBS in PowerAnimator 8.5. The developers wrote a plug-in to export geometry, texture information, and animation from 3D Studio MAX to the .HMD format, consequently, all the RT3D data had to pass through MAX.

In addition to the off-the-shelf technology, there were several in-house tools developed, including one which allowed the artists to physically place each texture in memory. This allowed for the most efficient use of space when storing differently sized textures with variable bit depths, a process that is hard to automate effectively.

At the time of the interview, the product was well on it's way to completion. SHADOW MADNESS had just reached alpha, and was on track for a Q1 1999 release date.

TOP GEAR OVERDRIVE
Category: Racing Game
Format: N64
Developer: Snowblind Studios
Publisher: Kemco
Release Date: Q4 1998

The Nintendo 64

Although it arrived relatively late in the game, many developers think the polished look and feel of the N64's technology were worth the wait. Boasting a user base of over 7 million units (as of September 1998), the latest version of Nintendo's console line has a lucrative market ready to be exploited by the savvy developer. To examine the N64 platform development process, we'll look at a game that is just finishing production over at Snowblind Studios, TOP GEAR OVERDRIVE.

THE GAME. TOP GEAR OVERDRIVE (TGOD) is Snowblind's first title, and is based on the Top Gear franchise created by Kemco of Japan. The self-proclaimed goal at Snowblind was to create a racing game with a more "arcade-like" feel. The game boasts over 30 tracks, myriad driving conditions, and over 20 cars to choose from, including the new VW Beetle. The fact that the game is near completion and has only been in production for about nine months is a testament to the efficiency and technical know-how of the six-man development team working at Snowblind.

DEVELOPMENT STRATEGY. With some experience on the platform prior to starting work on TGOD, Snowblind wasn't going into the situation well, er, with blinders on (sorry, couldn't resist that one). As with the PSX, the sheltered N64 market promised to be extremely lucrative to the developers. The relative dearth of racing games on the N64 was an added bonus. The N64's rendering features included several that were not available on PSX, such as perspective correction, tri-linear filtering, and Z-buffering, to name a few. And the lack of a need for any cinematics meant there was no real need to go with a CD-based system.

Obviously, getting the artists off to an early start on a development cycle this short was crucial. In order to familiarize the art team with the limitations of the system, their strategy, says Raoul Said, a programmer at Snowblind, was

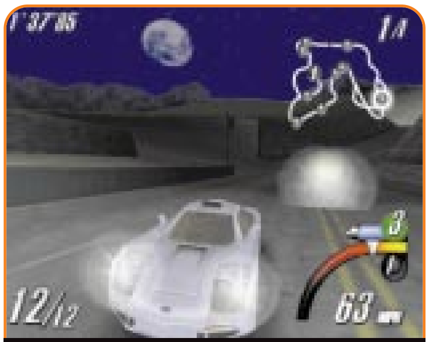


FIGURE 6. TGOD for the N64.

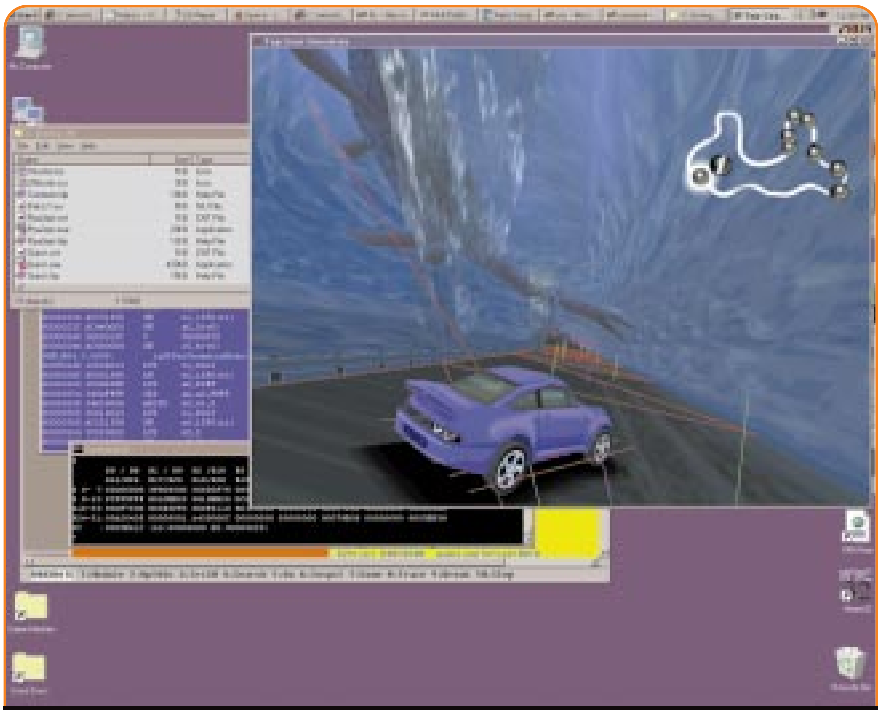


FIGURE 7. PC-compatible version of TOP GEAR OVERDRIVE.

to "… have one programmer jump ahead on the game framework and car physics code (on OpenGL/PC) while another programmer brought our N64-specific code up to speed, while the third programmer helped out on tools. The game ran on N64 dev stations from very early on." Figure 7 shows the PC compatible version of the game.

This gave the art team access to the nuances of N64 development prior to entering full production, which allowed them to streamline their art path to what was necessary and efficient.

TECHNOLOGY TRICKS AND CHALLENGES. As with the PSX, the development challenges were somewhat offset by myriad technical features aimed at making production easier. Additionally, the rendering and texturing features found on the N64 give the games a softer, more polished look.

The two biggest potential pitfalls faced by the art team were the total on-screen polygon count (total rendered polygons limited to between 1,500 to 2,500 polygons), and the limitation on memory storage (4MB of general purpose ram, of which 1.1MB were set aside for tracks, textures, and cars).

The challenge of managing scene complexity is not unique to consoles, but is instead faced by almost all developers working with RT3D content. The solution was found through the tried-and-true method of iterative testing. Tracks were generated using the custom lofting tool the team developed (discussed hereafter), then tested out with terrain and geometry added. Adjustments were made over the course of testing to get the approximate polygon count as close to the target as possible. This, of course, grew easier with practice, until the artists got to the point that they could create a track from start to finish in under three months (for one artist working on one track).

The second challenge, that of total memory storage, was tackled by setting a limit on the total number of polygons in a track (around 10K polygons, including physics barriers), thereby setting a limit on the actually size of the geometry in memory. Surprisingly, those innocent little polygons turned out to be gluttons for space. In addition to the raw geometry, additional data tied into radiosity information (for soft shadows) and visibility set information had to fit into the same block of memory. Figure 8 maps out the memory usage.

Finally, the team needed to take advantage of N64's ultra-fast texture cache. Aside from the 4MB of general purpose RAM, the N64 comes equipped with a super-fast texture cache, capable of extremely high transfer speeds. The downside is that the pipeline is relatively small. To use the cache to full capacity, texture maps had to be small; equivalent to 32×32×16-bit, or an equivalent size on disk (textures could have large dimension, with a correspondingly lower pixel depth, 64×64×4-bit, for example). So, large areas of complex terrain required detailed texture work, and the large background images had to be cut up into smaller individual pieces in order to meet the size requirements. Figure 9 shows an example of an environment map used for one of the sky domes. This texture map, a tiling sky piece, had to be cut up into 36 smaller textures and mapped onto a 170 polygon dome.

THE INTENSITY ALPHA TEXTURE. IA4 for short, the intensity alpha texture is a feature found only on the N64 Platform. Used extensively by the artists at Snowblind, the IA4 texture provides a space-efficient way to create texture maps. Here's how it works. You create a texture using a grayscale palette, and then choose two bounding colors to use as your color wash. The bounding colors act like a gradient tool, in that they match up to the texture map correspondingly to the lightest and darkest areas of the texture. For example, let's say I created a grayscale texture map with shades ranging from black to white, and then picked my bounding colors to be blue and red. The result would be a colorized texture map, with the black areas receiving a blue tint, the white areas receiving a red tint, and the gray areas receiving a purple tint. You basically create a texture that takes up the space of a grayscale texture, yet is fully colored.

TEXTURE MIRRORING. The artists at

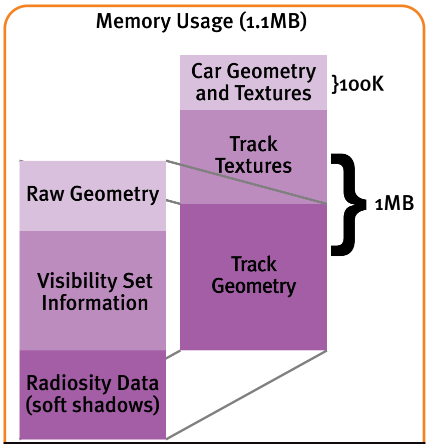

Memory Usage (1.1MB)
Car Geometry and Textures
}100K
Track Textures
Raw Geometry
1MB
Track Geometry
Visibility Set Information
Radiosity Data (soft shadows)

FIGURE 8. *Memory Usage.*

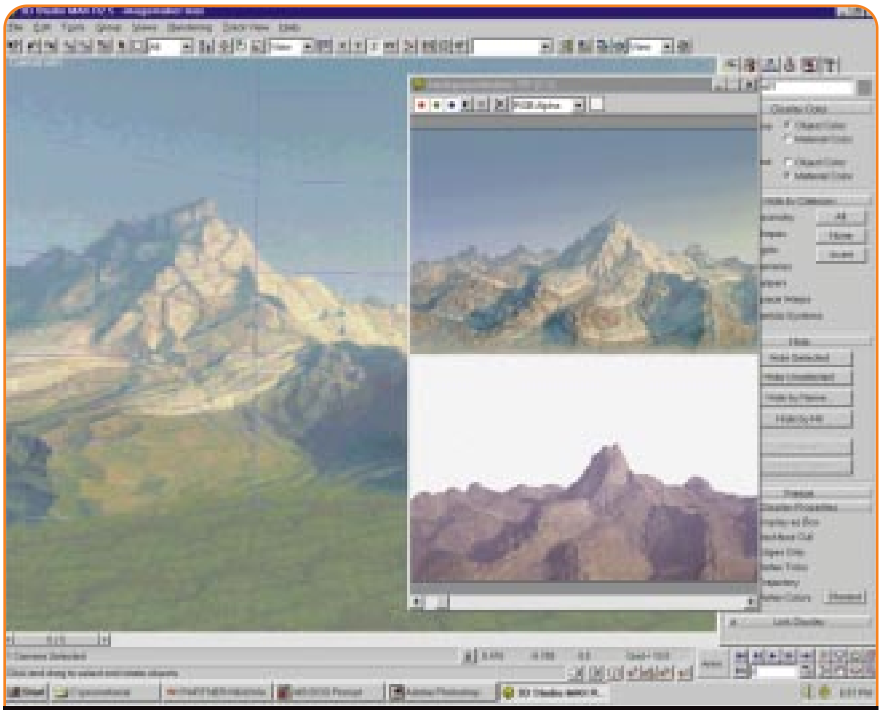


FIGURE 9. *An environment map used for one of TOP GEAR OVERDRIVE's sky domes.*

Snowblind also used the mirrored texture, another feature unique to the N64. This feature allows the artist to flip a texture along one or both axes of a quad polygon, a method that would otherwise require additional tesselation to produce the requisite polygon edges at the mirror boundaries.

THE TOOLS. In its most basic form, the task for the art team boiled down to one thing: create texture-mapped polygons. To accomplish this, the group settled on 3D Studio MAX 2.0 as its single off-the-shelf technology. The open plug-in architecture allowed for rapid toolset generation, and within only a few weeks of going into full production, the engineering team at Snowblind had come up with a suite of plug-ins to augment the already powerful modeling tools in 3D Studio MAX. The primary tool was the track lofting plug-in, which allowed the artists to create and test initial track layouts at the rate of one per day. This rapid turnaround was crucial to tweaking the playability of the tracks, and the entire process of getting a track into the engine was accomplished with only a few button clicks. Other plug-ins included lofting and sculpting tools, a texturing tool for dealing with huge numbers of texture maps on a single mesh (a must for large, multi-material geometries).

The folks over at Snowblind may well have a hit on their hands. The game had that clean, polished feeling characteristic of the N64, and it was fun to play. And considering this six-man crew will have created the game from start to finish in just under ten months, their achievement becomes even greater. This group will be one to watch in the future.

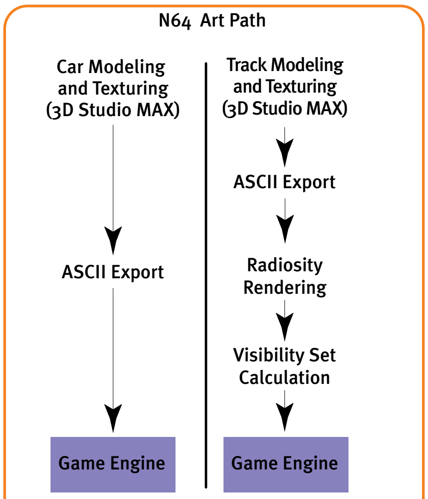

N64 Art Path
Car Modeling and Texturing (3D Studio MAX)
ASCII Export
Game Engine
Track Modeling and Texturing (3D Studio MAX)
ASCII Export
Radiosity Rendering
Visibility Set Calculation
Game Engine

FIGURE 10. *N64 art path.*

Wrap-Up

The hazards and limitations associated with console development pale in comparison to the benefits. The stable technology base, though limiting at first glance, inevitably allows developers to focus on ways to get the job done, instead of trying to figure out exactly what their target platform can do. This is a subtle yet critical distinction, and is leading more and more developers down the path of console development. ■

Special Thanks To:
Brian Sostrom, Ryan Geithman, Raoul Said, Ezra Dreisbach, Brian Johnson, Jason Wiggin, Louise Smith, Annabella Serra, Kieth Wymetalek, Joseph Laurino, and Anna Farr.

Trends in the Graphics Board Market

Game developers have been targeted extensively over the past four years by chip makers trying to convince them that their feature set, or particular approach to 3D hardware, is ideally suited to their applications. Ironically, chip makers rarely ever get to connect with the end user.

This is the job of the board maker. Only in the case of ATI and Matrox, where the chip maker and the board maker are one and the same, is there a direct connection between the technology at the silicon level and its marketing to the consumer. Companies such as Diamond Multimedia, Creative Labs, and STB Systems are chip agnostic, and in this month's column we're going to explore primarily the chip-agnostic graphics board industry to determine how these companies will fare in the coming year. The 3D graphics hardware business is getting merciless. The board vendor is on the front line, and is going to feel the pain more acutely than anyone else in the industry. The consumer, on the other hand, is going to be getting even more 3D power for even less money.

Love and Hate Relationships

The relationship between chip- and board-makers is one that they both resent. Why it even exists is our first port of call, followed by why both parties wish that it didn't have to exist. Before I go any further, I should probably put a proviso in here about 3Dfx, the most visible 3D graphics company in the game industry. 3Dfx is unique in having been successful at building its brand and influencing the end user of its products directly through its marketing efforts. Thus, 3Dfx has achieved a unique position in the industry, but I see the company's success in this regard as an exception rather than the norm. Companies such as Nvidia, S3, and 3Dlabs do market their brand, but their influence with end users doesn't rise to same level. As a result, these chip companies, including 3Dfx, put their energies into supporting the board vendor brands of Diamond, Creative Labs, or STB and others. The reason for this deference to board vendors is distribution and support.

Board vendors control the distribution channels. This gives them the connection to the customer, and invariably, they have to create a support structure to serve that customer. By contrast, chip vendors' customers are their board and system OEMs. The value of board vendors is their experience supporting end users, and their realization that there is more to delivering a graphics product than producing a working piece of silicon. That something more is software drivers. Chip vendors have enough on their plate dealing with the every rising complexity of chip design and manufacture. They may not be as savvy as board vendors when it comes to writing software drivers. From the time a finished chip is ready to go, to the time it is ready to be shipped to a customer may take weeks or months, depending on the quality of the drivers. This lag in time between when a chip vendor thinks a product is ready to go, and when a board vendor thinks it is ready to go is one reason why the two entities don't always see eye to eye.

In addition, chip-agnostic board vendors are usually vying with each other to differentiate products based on the chipset using software features and driver performance. The exception to the rule was 3Dfx, which reduced the board vendors' value-add by delivering a finished board and driver product that didn't leave much room for differentiation. If you read the round-ups and reviews of 3Dfx boards, you may have found that they were quite amusing — reviewers desperately tried to justify their preferred choices, but there was hardly any difference from one Voodoo2 board to the next. The Voodoo2 reduced board vendors to distributors and support staff for what was essentially a purely 3Dfx product.

So, the question arises, how much value-add do board vendors actually provide? Taking a cue from Voodoo2, it would seem not to be very much, but Voodoo2 was a 3D-only product targeted at a very specific hardcore gamer demographic. Go into the realm of 2D, and games aren't the only problem. Every piece of software that runs on a PC could be a nightmare support problem. Maybe chip makers, who all make 2D/3D chips, could deliver a finished board product themselves, but they realize that it might involve supporting OS/2 or Linux, or someone's ten-year-old DOS application. So, although the board vendor is really a tier of distribution for the chip vendor, it's also a buffer against consumer support.

Yet, forces beyond the control of chip and board vendors threaten this delicate graphics ecosystem. The biggest impact in 1998 on the board business was to be the entry of Intel into the market. Intel has managed to successfully devalue the graphics chip business to such an extent that the effects are going to be felt in the industry for some time to come. John Latta of the market research firm, 4th Wave (http://www.fourthwave.com), was the first person to publicly decry the pricing policies of Intel with regard to its Intel740 graphics chip ($7 a chip).

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

Brave man, Mr. Latta. A devalued graphics chip market drives prices down on all components and board products associated with it. As we go into 1999, we can expect much more aggressive pricing on board products, some very shaky board vendor financials, but healthy volumes for the performance leaders based on the pricing. It's good news for the consumer. It's good news for Intel, which just wants to sell more CPUs. But at what cost to the graphics board industry?

The Market

The cost for the industry, on the surface of it, doesn't look bad. "The 1998 add-in board market is expected to grow 65 percent over 1997, and then will slow down to an average growth of 18 percent per year up through the end of 2001," says Bob MacQuillan, senior research editor at Jon Peddie Associates (http://www.jpa.com). "The overall add-in board market will enjoy good growth. However, the entry/value and mid-range categories in the mainstream segment will be virtually eliminated as the market moves toward sub-$1,000 PCs that utilize embedded designs of graphics controllers and core logic."

Mr. MacQuillan's comments point out the real effect of Intel's entry into the market in 1998, and its plans to put 3D graphics in core logic in 1999, and come out with a lower-cost Intel740 follow-up, too. Competition in the entry-level and mid-range is only for the brave with deep pockets, or a strong constitution.

The expansion of the market for higher-performance 3D parts is the bright spot for the graphics board vendors. I think it might turn out that 1999 brings in enough volume in the performance sector to offset the entry-level and mid-range business woes. It may not, however, bring with it a comparable increase in revenues and profits. The battle between Nvidia, S3, Matrox, ATI, 3Dlabs, and 3Dfx for performance and brand leadership is just as likely to drive prices down as anything Intel could, or would do. Without exception, in 1999 the chip companies are going to move 0.25 micron chip designs, which means cheaper and faster designs. Further, the multitexture engine is here to stay, so the interest in new 3D technology will still be there, although market drivers remain stagnant. So all this competition is strictly for games — no other software needs the power. Having everyone chasing the game enthusiast may be good for game developers, but it's not necessarily good for an industry that may be crowding too many competitors into too small a space.

Figure 1 shows how important these high-performance parts are to the market. Jon Peddie Associates defines the category of performance as typically characterized as having either hardware floating point triangle set-up (such as ATI Rage Pro, Nvidia Riva 128, and so on) and/or high fill-rates (such as 3Dfx Voodoo Graphics, NEC PowerVR, and others). The Tiburon, Calif., firm defines mainstream as controllers containing 2D plus minimal 3D capabilities ("free-D"). Examples include the S3 Virge series and the Trident 3DImage 975.

Performance parts are the only value parts of the board vendor's product line. It used to be that a board vendor could look forward to selling a mature product into mid-range and entry-level markets, but that opportunity is dwindling. Board vendors make their money from creating awareness among consumers of cutting-edge products. They make their money from selling premium products. It seems as if a sterling year awaits the board business in 1999, but again, it's a question of how board vendors will make money if the graphics market is being devalued by the presence of Intel, and by the strength of the sub-$1,000 PC. It's all up to the players.

The Players

In Table 1, I have designated Tier 1–3 board vendors. The classification is partly based on channels, partly on exposure to brand name PC OEMs, and

FIGURE 1. *3D controllers by segment: 1997 to 2000. (Source: Jon Peddie Associates)*

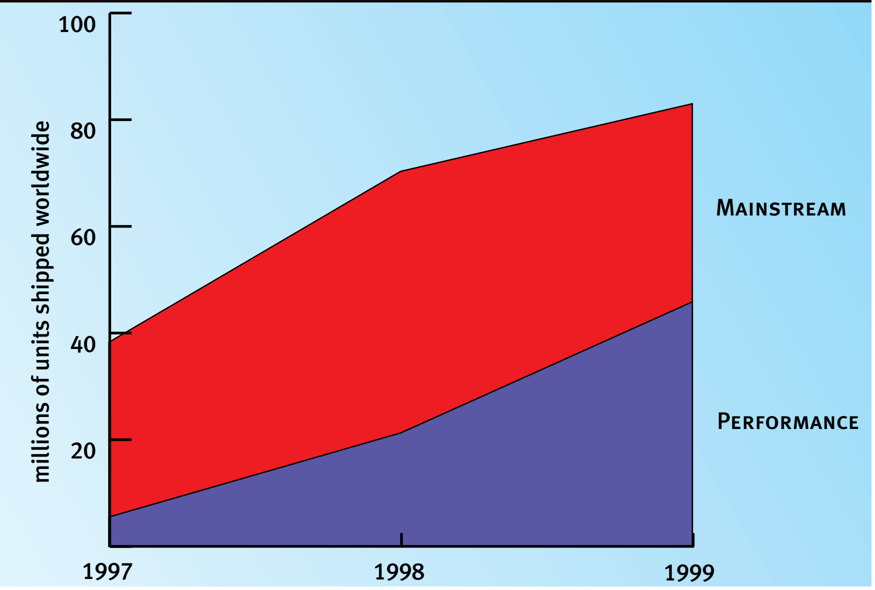
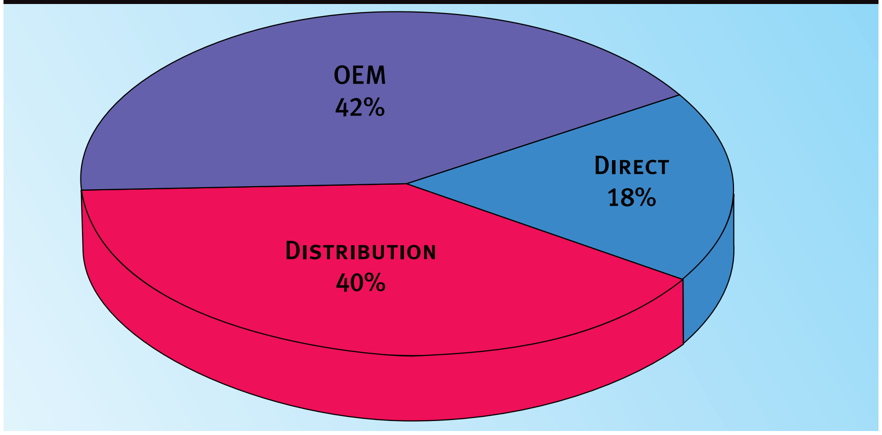


TABLE 1. *A list of the most recognized board names by category.*

Tier 1 Vendors	Tier 2 Vendors	Tier 3 Vendors
ATI Technologies	California Graphics	ASUStek
Creative Labs	Canopus	Leadtek
Diamond Multimedia	E4	Jaton
ELSA/Hercules	iXMicro	Expert Color
Intel	Number Nine	Techworks
Matrox	Radius	
STB Systems	VideoLogic	
	Wicked3D	

FIGURE 2. *Branded 3D add-in board market by channel. (Source: Jon Peddie Associates)*



partly on volumes. It's something that hasn't been done for graphics board companies, but is done for PC OEMs. It is not a definitive list of the Tier 3 vendors, but it does give you the graphics board landscape as it is today. I have given Intel a Tier 1 position, rather reluctantly, but they do have a strong brand. Also, I am being generous to ELSA/Hercules because of ELSA's strengths in Germany. In fact, I believe that in reality there should only be five Tier 1 companies: ATI, Creative, Diamond, Matrox, and STB. That's five companies serving ten Tier 1 PC OEMs. These Tier 1 vendors are under the greatest pressure in the coming year. They face squeezed margins, competing product lines from their chip suppliers (Nvidia's TNT versus the 3Dfx Banshee is one good example), and rising costs as they struggle to differentiate their brands. ATI seems to be the strongest company, having a healthy OEM presence, and some sound design wins for its Rage chipset. Creative is by far the best retail company in the Tier 1 arena. However, when it comes to branding graphics boards for a games audience, Diamond has done a very good job in North America. The weakest brand in the Tier 1 group is STB, which tends to be much more focused on supporting the Tier 1 PC OEMs using its manufacturing skills.

The Tier 2 vendors are, in some ways, more interesting than the Tier 1 board vendors. They have to work harder at differentiating their products. Canopus innovates in the hardware design by adding features and enhancements that the other board vendors may overlook. Wicked3D, the Metabyte board company, has pioneered stereo graphics by delivering drivers that support the widest ranges of stereo-enabled games. E4 has extensive support for DVD, and so forth. It's tough for the Tier 2 vendors because they are at the mercy of allocation by the chip vendors, more eager to get their three months of shelf space fame with Diamond and Creative, or to support STB supplies to Compaq, Dell, and Gateway. So, Tier 2 board vendors may innovate and target a game-playing audience better than the Tier 1 player, but they have to battle for channel recognition, and they don't get the best support from the chip vendors who look to their volume customers.

At the tail end are the Tier 3 vendors, which happen to be the biggest volume takers too. They take mature products, they make low-priced solutions, and they can ship hundreds of thousands of units a month, in excess of the Tier 1 brand names. As a game developer, you may never have to support a Jaton board, for example, but within the board industry, Jaton and its like play an important role delivering boards to all those resellers and PC-makers who don't want to pay a premium for a brand name, but do want to get their hands on good graphics boards, or the products of a recognized chip vendor. The non-branded board vendors supply in excess of the branded board vendors by a factor of 2:1. It used to be that Tier 3 vendors were seen as adding little value, and in bringing product prices down. They were the volume supplier of mature products. However, as competition among chip vendors heats up in the performance sector, the Tier 3 vendors will get to compete head to head with Tier 1 and Tier 2 players by getting product before it matures. Unfortunately, the Tier 3 vendors rarely have the resources, or the commitment, to raise awareness of their brands, so although they may end up shipping more product than the brand companies, no one outside the industry gets to hear much about it.

One mustn't forget that the PC graphics business is bigger than the sum of these vendors — over 80 million units in 1999 alone. There's room for a lot of players, but whether there is value for the players to stay in the business is debatable. For example, I have relegated Number Nine to a Tier 2 position because, despite having some excellent technology, the company has failed to deliver a Tier 1 position in any of its markets. Number Nine doesn't sell the cheapest boards, and it certainly doesn't make the worst products. Yet, the company finds it hard to mine its niche of the high-end business desktop, and it cannot generate the money it needs to move its development in a direction that would make it competitive. VideoLogic may also seem contentious for a Tier 2 position, but the company has never done anything of any significance in the board market, and the fact that NEC breathed life into the company through PowerVR is not influencing their board business. With technology, both these companies are good acquisitions: Number Nine is very close to Silicon Graphics and, of course, VideoLogic is very close to NEC and Sega.

With such premier names showing signs of strain in the board business, it makes you wonder which is the next brand name to feel the pinch. The answer may lie in the dynamics of the channel.

The Graphics Channel

If you look at the distribution pie chart supplied by Jon Peddie Associates (Figure 2), the distribution and OEM channels take an even bite of the sales pie, and interestingly enough, direct sales are showing some strength as well, specifically as a result of the Web-savvy game enthusiasts. With distribution and OEM channels practically eating up whatever profit margin potential there is, the only real profitable outlets for board vendors are direct sales. Yet, the brand name board companies are very cautious not to be seen in competition with their distribution channels. There's a little pull and push there.

It seems that the direct channel may afford an opportunity for the Tier 2 vendors to establish themselves. Tier 3 vendors are primarily OEM and distribution bound, with little in the way of retail. Therefore, the channel may have already decided how the board business shapes up.

Looking at the channels in more detail, we can see some patterns emerging. Diamond is very strong in the North American retail market, but is under increasing pressure from Creative. ATI and Matrox fare better in Europe and international markets than they do in North America, but in general, have strengths in the PC OEM and reseller channels. ATI's retail presence is the stronger of the two. STB has yet to make any breakthroughs in retail, and is 80 percent PC OEM business. It faces pressure from ATI, Matrox, and Diamond, but it always has.

Jaton, Leadtek, and Expert Color sell through distributors serving resellers and systems integrators. You may find the odd low-cost controller from the Tier 3 vendor in a retail store, but it won't be sitting on the shelf at CompUSA or Electronic Boutique. They could break out with product, but none of the Tier 3 vendors are inclined to package their products for the consumer.

So, the status quo is pretty much set, but there is hope in the Tier 2 section. Companies such as Canopus, Wicked3D, California Graphics, and E4 want to be consumer friendly. They have the hunger, and they have to differentiate their products much more than any Tier 1 or Tier 3 vendor would have to do. They are relatively free to make new channels, or to go into areas that the big companies are precluded from, such as having a more aggressive direct sales strategy. However, Tier 2 vendors need volumes to get better chip pricing and support from the semiconductor companies, and that's where they face the biggest challenge. To get volumes, Tier 2 vendors need OEMs. To get OEMs, they need strong retail, or branding, or manufacturing capability. It all costs money. To get money in a price-squeezed market, they need volume.

However, I believe that board companies have to change. They need to become leaner, and they need to change the way they do business to take into account the new business dynamics. This may be the best chance for a Tier 2 or Tier 3 vendor to break into the Tier 1 category, but it doesn't look like the big five are going anywhere for now, profits or no profits. In the middle of all of this, the consumer is going to get a smorgasbord of very good 3D products, in different flavors of chips, and from a diverse group of board vendors. Cheap, too. A 16MB, Banshee board from Creative Labs retails for about $99, having gone down from $149 in the space of six weeks, or less. Brutal business, but someone's got to do it. ■

BRINGING ENGINEERING DISCIPLINE TO GAME DEVELOPMENT

BY GORDON WALTON

As the manager of Kesmai Studios, I oversee the internal development division of Kesmai Corporation, the oldest and largest maker of massively multiplayer games. The games we develop (such as AIR WARRIOR, MULTIPLAYER BATTLETECH, ALIENS ONLINE and LEGENDS OF KESMAI) support hundreds to thousands of simultaneous players via the major online service providers (America Online, Compuserve and others) and over the Internet (at http://www.gamestorm.com). Our games are played much longer than most stand-alone boxed games. For example, two of our current games are over ten years old, and several were released more than five years ago. All of these games have been updated many times as the underlying computer technology has evolved.

Gordon Walton has been authoring games and managing game development since 1977. He is currently senior vice president of Kesmai Corporation and general manager of Kesmai Studios, the leading developer and distributor of multiplayer online games. He has been a speaker at every CGDC since its inception, is a founder and current officer of the IGDN and speaks at other industry gatherings. He can be reached at gordon@kesmai.com.

A year ago, we decided to change radically our methods of developing games. We were dissatisfied with our maintenance costs, the effort required to achieve our desired level of quality, and our inability to predict production schedules. This article is a description of what we did to achieve this change, although you should view what you're about to read as an "after-action report" — our development processes are continuously evolving to meet our objectives.

What Is This "Engineering Discipline" Thing?

Are you sick of not being able to predict when the development of your game will be complete? Are you tired of creating bug patches for a game that shipped months ago? Are you having trouble adding a feature that the new vice president of marketing demanded halfway through development? If any of these scenarios sound familiar, let me offer you a way of dealing with them.

Engineering discipline, as I define it, is determining and applying processes to the development of game software with the goal of improving the quality, maintainability, extensibility, and schedule visibility of the software. Being able to deliver these particular elements consistently in game software is unprecedented, in my experience.

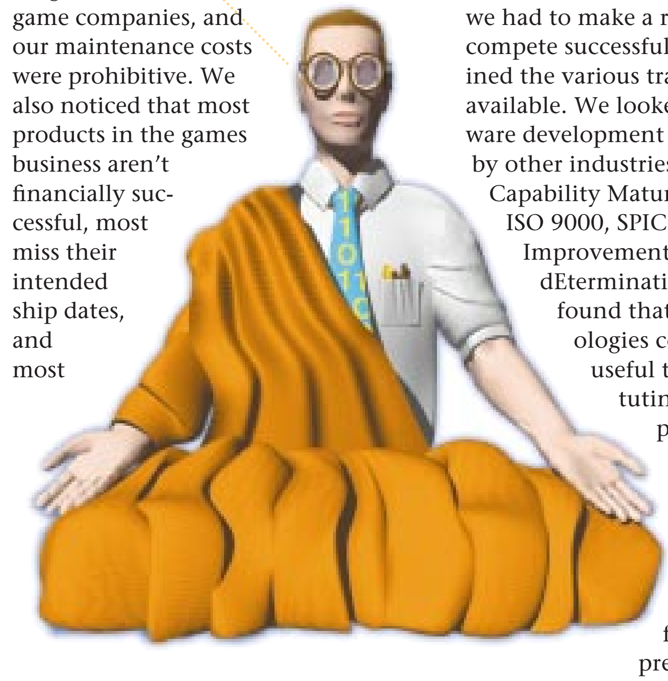
What Is the Reality of Current Game Software Development?

While both the team and management share a desire for the elements of quality, maintainability, extensibility, and schedule visibility in their software products, the primary mission of most game developers is to ship a fun game by Christmas. The desirable ship dates for most products are dictated by the realities of the retail market. With the majority of all software sales occurring between the months of September and January, the pressure to ship before Christmas is incredible. The final ship date for Christmas is about two weeks before Thanksgiving, given the normal retail distribution delays and processes. Some products succeed when shipping outside of this narrow window, but typically this success is limited to about ten highly promoted and anticipated titles. Odds are that you are not working on one of these titles.

Almost everyone with experience in game development has had to work on someone's legacy code. Most experienced programmers have had to fix an incompatibility or bug for a product developed by someone else, or have had to start a sequel from a previous product code base. In general, game software isn't written to be maintainable. Rather, the primary goal is getting the code "finished enough" to ship it as a commercial product. As such, many game developers believe that any documentation and coding standards come at the expense of getting the game done sooner. This lack of documentation and standards worked for most game developers for a time, as teams used to be relatively small and time to market was the most important issue. But times have changed.

Why Consider An Engineering Discipline?

The main reason that we at Kesmai considered a new way to develop software was that the methods commonly used in the game development business weren't working well. Our products' life cycles were longer than those of most game companies, and our maintenance costs were prohibitive. We also noticed that most products in the games business aren't financially successful, most miss their intended ship dates, and most have significant bugs and require a patch (or three). Within a viciously competitive environment, as an industry we must find a way to lower some of the risks involved with building game software.

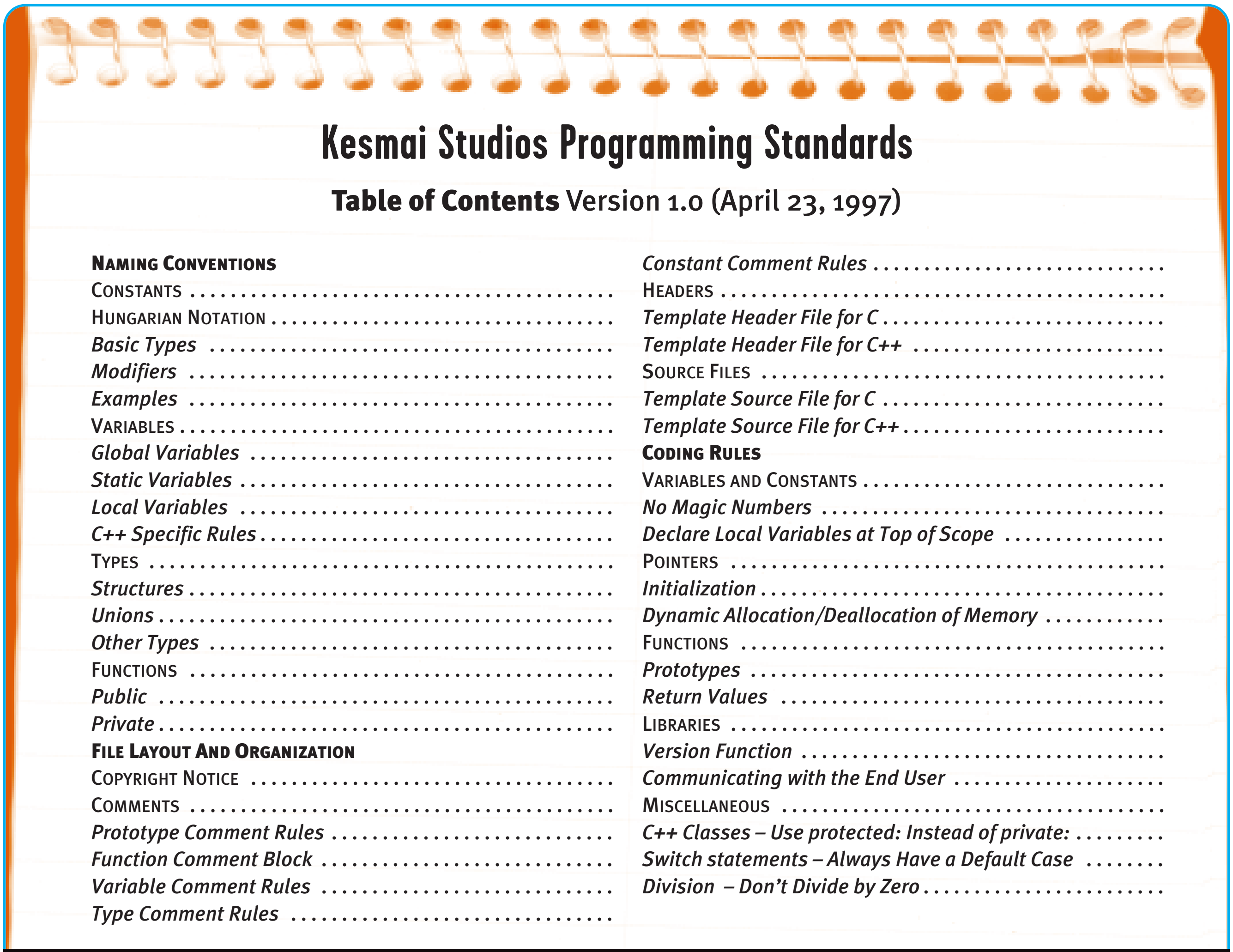


Not all risks are controllable while developing game software. You can never be sure if consumers will like what you offer (though there are methodologies for reducing this risk, too). You never know when Microsoft might change the operating system components. Sometimes, a competitive product is unexpectedly introduced, and you must react to it. Management often will dictate large numbers of unexpected changes.

Given this chaotic environment, it's important to acknowledge and manage the risks that are under your control. For risks that you don't control, but are within your sphere of influence, you must make agreements with all parties so that no surprises occur without renegotiation. This gives you a manageable baseline with which to start, so that you can react with more clarity and confidence to the truly uncontrollable risks.

How We Approached an Engineering Discipline

Because our company is in a business where products must be maintained for several years, we felt we had to make a radical change to compete successfully. We first examined the various training opportunities available. We looked at several software development methodologies used by other industries, including the SEI Capability Maturity Model (CMM), ISO 9000, SPICE (Software Process Improvement Capability and dEtermination), and others. We found that all these methodologies contained elements useful to our goal of instituting engineering discipline. They also contained elements that could stifle creativity and ignored the importance of having fun software in favor of schedule predictability.



Kesmai Studios Programming Standards

Table of Contents Version 1.0 (April 23, 1997)

NAMING CONVENTIONS
CONSTANTS
HUNGARIAN NOTATION
Basic Types
Modifiers
Examples
VARIABLES
Global Variables
Static Variables
Local Variables
C++ Specific Rules
TYPES
Structures
Unions
Other Types
FUNCTIONS
Public
Private
FILE LAYOUT AND ORGANIZATION
COPYRIGHT NOTICE
COMMENTS
Prototype Comment Rules
Function Comment Block
Variable Comment Rules
Type Comment Rules
Constant Comment Rules
HEADERS
Template Header File for C
Template Header File for C++
SOURCE FILES
Template Source File for C
Template Source File for C++
CODING RULES
VARIABLES AND CONSTANTS
No Magic Numbers
Declare Local Variables at Top of Scope
POINTERS
Initialization
Dynamic Allocation/Deallocation of Memory
FUNCTIONS
Prototypes
Return Values
LIBRARIES
Version Function
Communicating with the End User
MISCELLANEOUS
C++ Classes – Use protected: Instead of private:
Switch statements – Always Have a Default Case
Division – Don't Divide by Zero

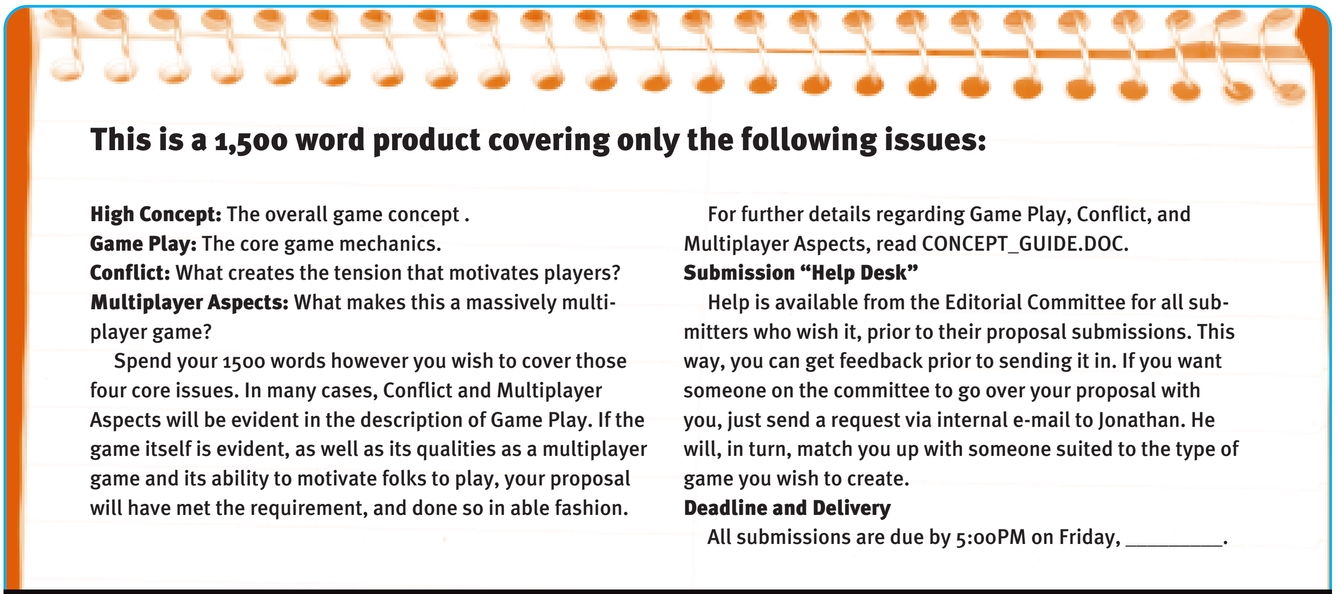
FIGURE 1. *Kesmai's programming standards table of contents.*

Because no current methodology that we investigated suited our needs completely, we decided to invest in training our personnel in the SEI's CMM methodology, with the proviso that we didn't want to follow their developmental model exactly. We sent 80 percent of our development and QA personnel to this training. We also investigated several project-management training programs. After one course given by the Learning Tree Company, two of our producers came back very excited and charged up. They really liked the particular instructor, so we contracted that instructor to give a one week class on project management to our entire development staff.

At this point, we had most of our personnel "speaking the same language" and interested in improving our software development methodologies. This was a key step. This kind of change in mindset cannot be mandated from management; it must be a joint effort between developers and management to work. We started our process of instituting an engineering discipline with three initiatives: programming standards, the peer review process, and the software process working group.

PROGRAMMING STANDARDS. We quickly devised a good set of programming standards that the entire development staff could buy into, although there was some debate as to how these standards would be instituted. These standards were created in reaction to the difficulty we experienced in maintaining our legacy code. While these new standards didn't help us maintain our old code, they made sure we wouldn't perpetuate poor coding practices in our new games. Thus, our payoff will come when we hire new people to maintain games in the future. The products that our developers will maintain in the future will have a defined coding style and methods. The table of contents for our programming standards, which illustrates the elements that we believed were important to standardize, are shown in Figure 1.

PEER REVIEW PROCESS. The definition of the term "peer review" is encapsulated in this quote from the SEI Capability Maturity Model: "The purpose of peer reviews is to remove defects from the software work products early and efficiently. An important corollary effect is to develop a better understanding of the software work products and of defects that might be prevented." Peer reviews, in this context, have no impact on personnel evaluation at all; they're simply a method of process improvement — never a methodology for placing blame. This peer review



This is a 1,500 word product covering only the following issues:

High Concept: The overall game concept .
Game Play: The core game mechanics.
Conflict: What creates the tension that motivates players?
Multiplayer Aspects: What makes this a massively multiplayer game?

Spend your 1500 words however you wish to cover those four core issues. In many cases, Conflict and Multiplayer Aspects will be evident in the description of Game Play. If the game itself is evident, as well as its qualities as a multiplayer game and its ability to motivate folks to play, your proposal will have met the requirement, and done so in able fashion.

For further details regarding Game Play, Conflict, and Multiplayer Aspects, read CONCEPT_GUIDE.DOC.

Submission "Help Desk"

Help is available from the Editorial Committee for all submitters who wish it, prior to their proposal submissions. This way, you can get feedback prior to sending it in. If you want someone on the committee to go over your proposal with you, just send a request via internal e-mail to Jonathan. He will, in turn, match you up with someone suited to the type of game you wish to create.

Deadline and Delivery

All submissions are due by 5:00PM on Friday, _________.

FIGURE 2. *Concept document guideline.*

process only took a few weeks to hammer out, and is now embraced by our developers. Reviews go on almost every week in our studio. The primary value of peer reviews is in finding problems and potential problems long before they would normally be found in the testing stage. A less obvious but equally powerful benefit is that all participants in the peer review process learn to avoid both common and subtle errors in their own work. We encountered some resistance to peer reviews, because there was concern that management would use these reviews to rate our developers. Once it was clear that this was not the case though, our developers embraced the process.

Our first peer reviews found significant problems that had defied other debugging methods, and every peer review we've conducted so far has found and corrected problems. As our people gain more experience with peer reviews, they're uncovering problems earlier in the development process — bugs might otherwise have taken weeks of debugging had they still existed when the game was in beta testing.

When used effectively, peer reviews are a powerful tool to improve both the quality and the effectiveness of your development staff. The relative cost of finding and correcting errors early in the development of a title is incredibly low compared to trying to correct bugs when you're about to ship. If you don't already conduct peer reviews, I recom-

The Requirements Baseline

The Requirements Baseline (RB) is a comprehensive list of specific requirements that apply to the software project. It draws from documents, correspondence, and interviews from all relevant sources to ensure a balanced depiction of project requirements. Essentially, it is a solicitation, then formalization, of the requirements of each department as they relate to your game. Its purpose is to document all known requirements in a clear and unambiguous manner, sufficient to be used as the basis for the game design process. The RB is the first document in the project life cycle that is owned and developed by the project team.

Why do we need it?

The RB is an agreement between the project team, management, and other departments over what constitutes the requirements that bear upon the task. By documenting the agreement, all requirements become visible and tangible. By mandating approval of the agreement by all parties, each becomes responsible for the accuracy and completeness of their stated requirements. Finally, by putting the agreement under revision control, changes can be identified and schedule impacts accounted for.

What's in it?

To accomplish the purposes described above, the Requirements Baseline contains the following elements:

Overview: A one- or two-page description of the game, sufficient to give the reader a sense of its game play and scope. This would be very similar in format to the Initial Concept Document, except for the omission of any features removed through the Initial Concept approval process.

Key Requirements Summary: A one-page condensation of the most important requirements drawn from the total known set. This is intended to provide a high-level, balanced perspective on the most important goals to be achieved by the project, uncluttered by the detail necessary in the departmental breakdowns.

Requirements Breakdown: A department-by-department itemization of all known and accepted requirements. This should contain documented, institutionalized requirements as well as project-specific requirements, as identified and submitted by the department's representative. Sources include, but are not limited to, published documents, e-mails, meeting minutes, and phone calls.

Change Log: Once the RB has been placed under revision control, any changes to the document should be noted in the change log. Each entry should contain (at a minimum) a description, date, and reason for the change.

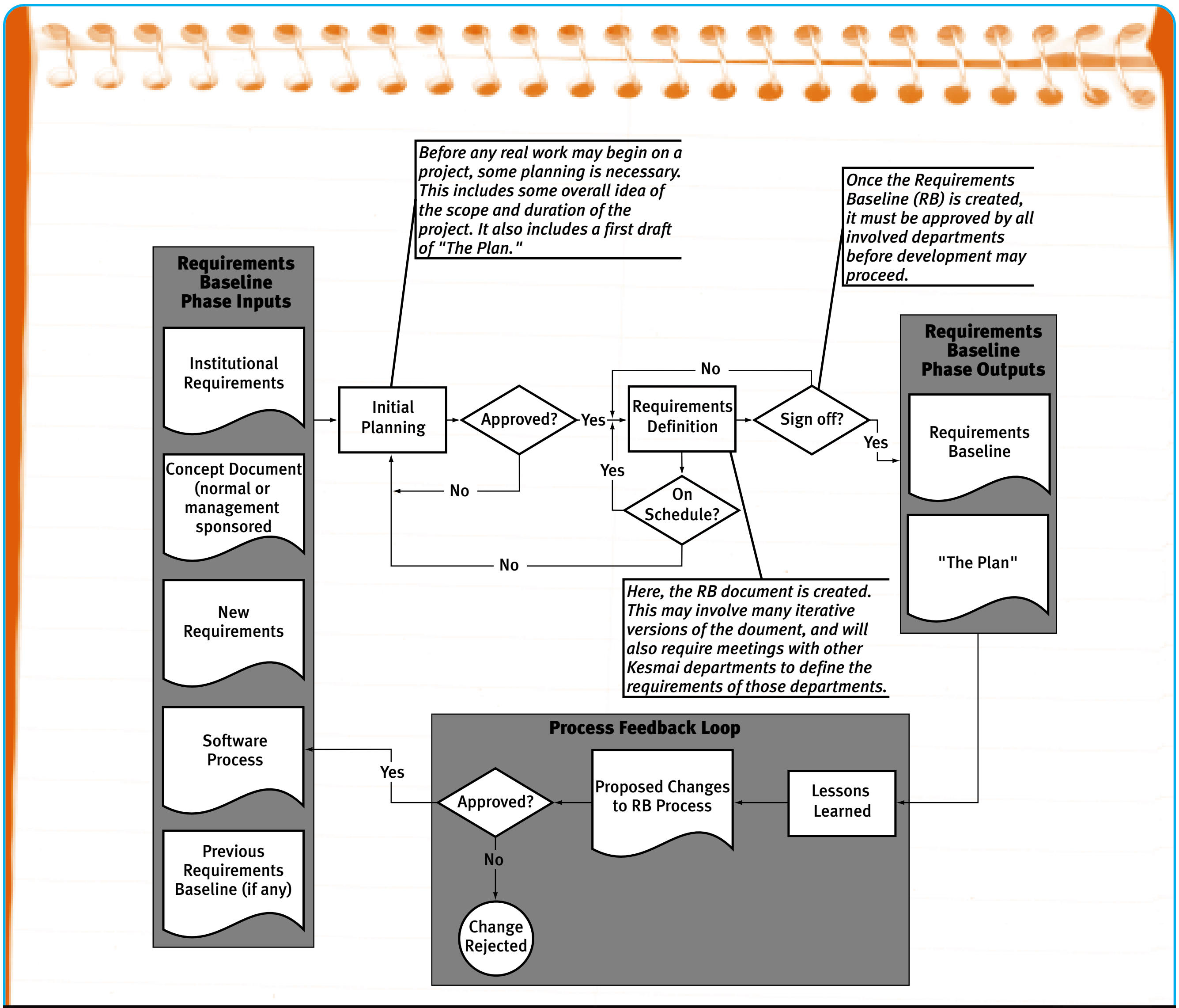

Before any real work may begin on a project, some planning is necessary. This includes some overall idea of the scope and duration of the project. It also includes a first draft of "The Plan."
Once the Requirements Baseline (RB) is created, it must be approved by all involved departments before development may proceed.
Requirements Baseline Phase Inputs
Institutional Requirements
Concept Document (normal or management sponsored
New Requirements
Software Process
Previous Requirements Baseline (if any)
Initial Planning
Approved?
Yes
No
Requirements Definition
No
Sign off?
Yes
Yes
On Schedule?
No
Requirements Baseline Phase Outputs
Requirements Baseline
"The Plan"
Here, the RB document is created. This may involve many iterative versions of the dooument, and will also require meetings with other Kesmai departments to define the requirements of those departments.
Process Feedback Loop
Yes
Approved?
Proposed Changes to RB Process
Lessons Learned
No
Change Rejected

FIGURE 3. *Requirements Baseline initiation and revision process.*

mend starting today.

Software Process Working Group. The Software Process Working Group was responsible for creating and delivering a number of items, which are consolidated into a document called the Kesmai Studios Software Process Guide. The Kesmai Studios life cycle has five phases:

1. Concept
2. Requirements Definition
3. Design
4. Implementation
5. Maintenance

This list is just one of many ways to define a product life cycle. Your company may have different phases and terminology. There is no one true way, as each company must define its life cycle to match its business and market requirements. We picked some new projects to test these processes as we created them. I will use a fictional product called Bazooka Bunny to illustrate some of these phases.

Concept. We created a document to describe what made up a valid game concept for our product life cycle. It's used to select what products are built in our studio. Because our development focus is massively multiplayer games, this quality is heavily emphasized within the concept document guideline, shown in Figure 2.

For the purposes of our development life cycle and internal selection processes, we wanted our concepts to be very compact; other companies might want much more volume and detail. A committee made up of members of the studio approves concepts for further development.

Requirements definition. The requirements definition phase is embodied by two documents, the Requirements Baseline and The Plan (see the sidebar, "The Requirements Baseline"). Our Requirements Baseline initiation and revision process is shown in Figure 3. Notice that there is a defined process for creating a Requirements Baseline, as well as for modifying the process of creating the Requirements Baseline (in the event that we decide that the process isn't optimal). All processes used must have feedback mechanisms so that they can continuously improve

Bazooka Bunny—Sample Requirements

Baseline Revision 1.4, September 20, 1997

Prepared by: John Robinson, Will Robinson, Doctor Smith

Overview

Bazooka Bunny is multiplayer action game in which poor, defenseless, and slow-witted computer programmers (the players), armed only with a handful of light, armor-piercing rocket launchers (and a few miniguns) venture forth into the cruel, harsh meadow to do battle with fiendishly clever rabbits. Terror ensues when the players' repeated saturation fire into the underbrush fails to produce the anticipated scurrying about, and fratricide levels begin to reach... (snipped for brevity) ...and in a furious frenzy smash the albino queen into red and white pulpy bits with their shovels, thus ending the hideous threat.

Summary of Key Requirements

- Supports up to 3,000 players competing in separate arenas (meadows) of up to 100 players each.
- Plays on a Windows 95 machine with DirectX 5.0 or greater.
- Plays on a Pentium 200MHz or better with 3D accelerator required.
- Requires 100MB hard drive space and at least 32MB of RAM.
- Plays with keyboard only, mouse/keyboard, and joystick/keyboard.
- Uses an isometric, third-person view.
- Uses real 3D, not prerendered sprites, voxels, or raycasting.
- Uses Visual C/C++ for FE development.

Requirements Breakdown

Project

- Supports up to 3,000 players competing in separate arenas (meadows) of up to 100 players each.
- Uses an isometric, third-person view.
- Uses real 3D, not prerendered sprites, voxels, or raycasting.
- Plays with keyboard only, mouse/keyboard, and joystick/keyboard.
- Each character has 24 distinct animations, 4 unique to each class.
- Each meadow is approximately 10 kilometers square.

Kesmai Studios

- Plays on a Windows 95 machine with DirectX 5.0 or greater.
- Plays on a Pentium 200MHz or better with 3D accelerator required.
- Requires 100MB hard drive space and at least 32MB of RAM.
- Uses Creator to produce objects and terrain.
- Uses Visual C/C++ for FE development.
- Uses a revision control system.
- Host development environment is HP/UX on HP hardware.

Creative Services

- Need to negotiate a staged production schedule for art, sound, and editorial, taking into account the software schedule, the interdependency of each piece, and the quantity to be produced.
- Need to be informed of the various technologies that will be used within the project to ensure that CS personnel are adequately trained and necessary tools are available by the work start date.

Marketing

- Need sample art and sounds for web pages three months prior to scheduled release.
- Need character class descriptions and updated feature list three months prior to scheduled release.
- Need .AVI of game play one month prior to scheduled release.

Product Support

- A list of error messages that might be conveyed to the user in the course of game operation and what actions to take to alleviate the problem. Need to have a way to enter the game when it's full.
- Need to have a way to find out player info using the player's in-game handle as a key.
- Need to be able to eject a player who is violating TOS agreement while playing.

Operations/Tools

- Need to have a list of all game processes, with sample configuration files, an explanation of what they do, and any special requirements they may have.
- Need to have detailed installation instructions for the host processes.
- Need a coordination meeting to determine the ARIES implications and feature schedule.

Product Acceptance

- Need a software test plan at least two weeks prior to the scheduled start of acceptance testing.
- Need to have FE install at least two days before the scheduled start of acceptance testing for every release.
- Need to have a beta test host delivered at least three days prior to the scheduled start of acceptance testing for every release.

Executive Committee

- The product must run on AOL and GameStorm.
- The product must be available by Q1, 1999.

Third-Party Support

- The game should avoid liberal use of the word "rabbit," because one of our other games already supports a familiar theme of blasting rabbits with bazookas. Referring to the critters as "bunnies" or "fuzzies" should deflect from any similarity.

Community Support

- The game must emit a player scoreboard in a negotiated format for display on the Web.
- The game must support a control query returning player information in the game context for display on the Web.
- The game must support a "news ticker" giving descriptions of what's happening in the game to be displayed in real-time on the Web.

Help Desk

- A list of error messages that might be conveyed to the user in the course of game operation and what actions to take to alleviate the problem.

Publishing

- The project must deliver installation packages in the format required by the GameStorm client.

Change Log

11/3/97: Added rabbit to bunny word replacement throughout the design documentation at the request of third-party support.

11/25/97: Changed number of animations from 12 to 24 after some hashing out of the initial paper design with the project team.

FIGURE 4. *Sample Requirements Baseline*

based on the input from the teams that use the process. A simplified example of a Requirements Baseline for our fictional BAZOOKA BUNNY product is shown in Figure 4.

You may have noticed the large number of departments that are referenced in the Requirements Baseline. Involving so many people at this step is initially time consuming, but it gets all the key departments involved and gives them the ability to give their input about the product. It also secures their agreement that these are their requirements for the product. Any future changes result in the creation of a new agreement, and all the parties must re-approve the changes. While this applies friction to the process of making changes to the project, it does discourage people from suggesting frivolous changes. Additionally, important changes are allowed to move forward with a consensus from the key parties involved. Once these requirements are formalized for the first product, most of these requirements are reused for follow-on products.

The Plan document, alluded to in Figure 3, outlines the general path that development will follow and describes additional elements beyond the key requirements listed in the Requirements Baseline. It also outlines all project deliverables, such as a terrain editor, mission editor, manuals, and so on. The Plan is a living document that changes as the product moves from the requirements phase through the design and implementation phases. The Plan demonstrates to management that the team knows what they are doing and how they will go about getting it done. Specifically, The Plan includes the following items :

- The vision and goal of the project (approximately one or two paragraphs).
- The list of reference materials.
- Project organization (management, responsibilities, work products, process model).
- Risk management.
- Project plan (deliverables, major milestones, staffing, and organization).

Every project develops a large number of work products during the course of the project. Some of these work products are deliverables that are given to people outside of the project team, such as the software itself, user manual(s), and test plans. Other work products are mostly for internal use, such as analysis and design documents, source code, and build procedures. The Plan document outlines all of the work products that will be created and gives a rough timetable and order of delivery. The intended target audience for each deliverable is also defined.

In order to prevent changes from completely derailing a project during development, an effective change control plan must be established early in the project cycle. This process is written into The Plan so that change control procedures are established from the very beginning and are used on every baselined work product.

We define high-level project milestones in The Plan and leave the detailed scheduling information in a separate document. Milestone target dates are initially estimated based on the project scope and budget. As the development progresses and the schedule becomes more accurate, these milestone dates are changed within The Plan. As such, we always keep The Plan up to date.

DESIGN. The design of the product flows from the requirements definition. The design is made up of a series of stages that are contained in four main documents under our definition of this phase (Figure 5). These documents are the Software Design Specification, the Software Project Plan, the External Resource Specification, and an initial Software Testing Plan. The Software Design Specification, in turn, is composed of three subphases: creative design, technical design, and resource analysis.

The creative design subphase is made up of a series of four documents, titled Overview, Thematic Content, Interface Storyboards and Prototypes, and Game Mechanics. The combination of these documents describes how the game will look, feel, and play. In addition, the initial External Resource Specification is created during the creative design phase, which lists all the art, sounds, and other externally created elements necessary for the product.

The technical design subphase is made up of a series of three documents: Requirements Analysis, System Architecture, and Module Specification.

Finally, the resource analysis subphase is where The Schedule, the final External Resource Specification, and the initial Software Test Plan are all created. The Software Project Plan is simply our schedule for the game's design and implementation, and it's a fairly common document to most projects. The Schedule is based upon the detail provided by the combination of the Software Design Specification and the External Resource Specification. The Software Testing Plan, which is initially created in this phase, is a roadmap for our product testing group to use to validate the functionality of the product.

IMPLEMENTATION. Once the design phase is complete and approved, the project moves into the implementation phase. The implementation phase is divided into the construction subphase and the release-and-refine subphase. Most of our larger products use a staged delivery process, so we construct multiple stages and then move the final stage into a release-and-refine subphase. The Software Testing Plan, which began in the Design phase, is completed during the construction subphase. This is a complete testing plan for the product describing the methodology and elements of a proper test of the product.

The majority of the construction subphase is spent doing the traditional programming, integrating, and testing of the product's modules as it moves towards completion. In each stage of the construction subphase, the Software Testing Plan is being incrementally completed.

Once a project has reached the alpha release milestone, it moves into the release-and-refine subphase (Figure 6). In this subphase, the focus is not on developing new features, but on testing, fixing bugs, and completing the product for the production release to players. During the alpha release stage, it's inevitable that features will be changed or added, but the focus in the release-and-refine subphase is on stability and fine tuning. This is the last development phase of a project, and it consists of four key stages:

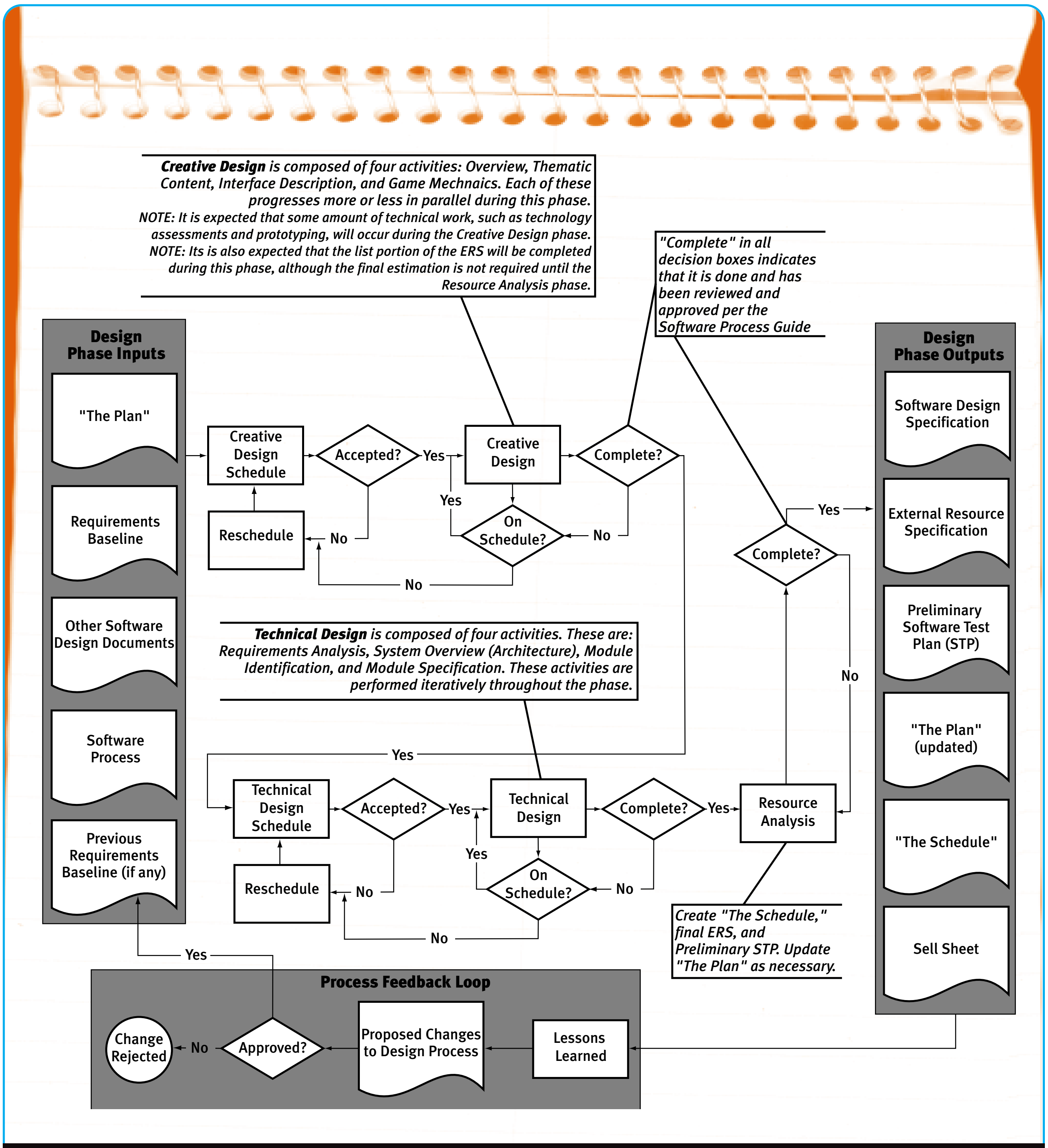

Creative Design is composed of four activities: Overview, Thematic Content, Interface Description, and Game Mechnaics. Each of these progresses more or less in parallel during this phase.
NOTE: It is expected that some amount of technical work, such as technology assessments and prototyping, will occur during the Creative Design phase.
NOTE: Its is also expected that the list portion of the ERS will be completed during this phase, although the final estimation is not required until the Resource Analysis phase.
"Complete" in all decision boxes indicates that it is done and has been reviewed and approved per the Software Process Guide
Design Phase Inputs
"The Plan"
Requirements Baseline
Other Software Design Documents
Software Process
Previous Requirements Baseline (if any)
Creative Design Schedule
Accepted?
Yes
Creative Design
Complete?
Reschedule
No
Yes
On Schedule?
No
No
Design Phase Outputs
Software Design Specification
External Resource Specification
Preliminary Software Test Plan (STP)
"The Plan" (updated)
"The Schedule"
Sell Sheet
Yes
Complete?
No
Technical Design is composed of four activities. These are: Requirements Analysis, System Overview (Architecture), Module Identification, and Module Specification. These activities are performed iteratively throughout the phase.
Yes
Technical Design Schedule
Accepted?
Yes
Technical Design
Complete?
Yes
Resource Analysis
Reschedule
No
Yes
On Schedule?
No
No
Create "The Schedule," final ERS, and Preliminary STP. Update "The Plan" as necessary.
Yes
Process Feedback Loop
Change Rejected
No
Approved?
Proposed Changes to Design Process
Lessons Learned

FIGURE 5. *Design process flow.*

1. Alpha Release
2. Closed Beta Release
3. Open Beta Release
4. Production Release

Maintenance. Once a game achieves good stability and it's at the gold-master stage, it moves out of development and into maintenance and enhancement. The project has been completed, and although work will likely continue throughout the life of our massively multiplayer games, it will be at a maintenance level. Most likely, maintenance will be handled by different personnel than the original development team.

Problems Faced at Kesmai

Lack of management buy-in was not a problem we had, but it might be one that you will encounter. Without the commitment of upper management to instituting engineering discipline through process improvement,

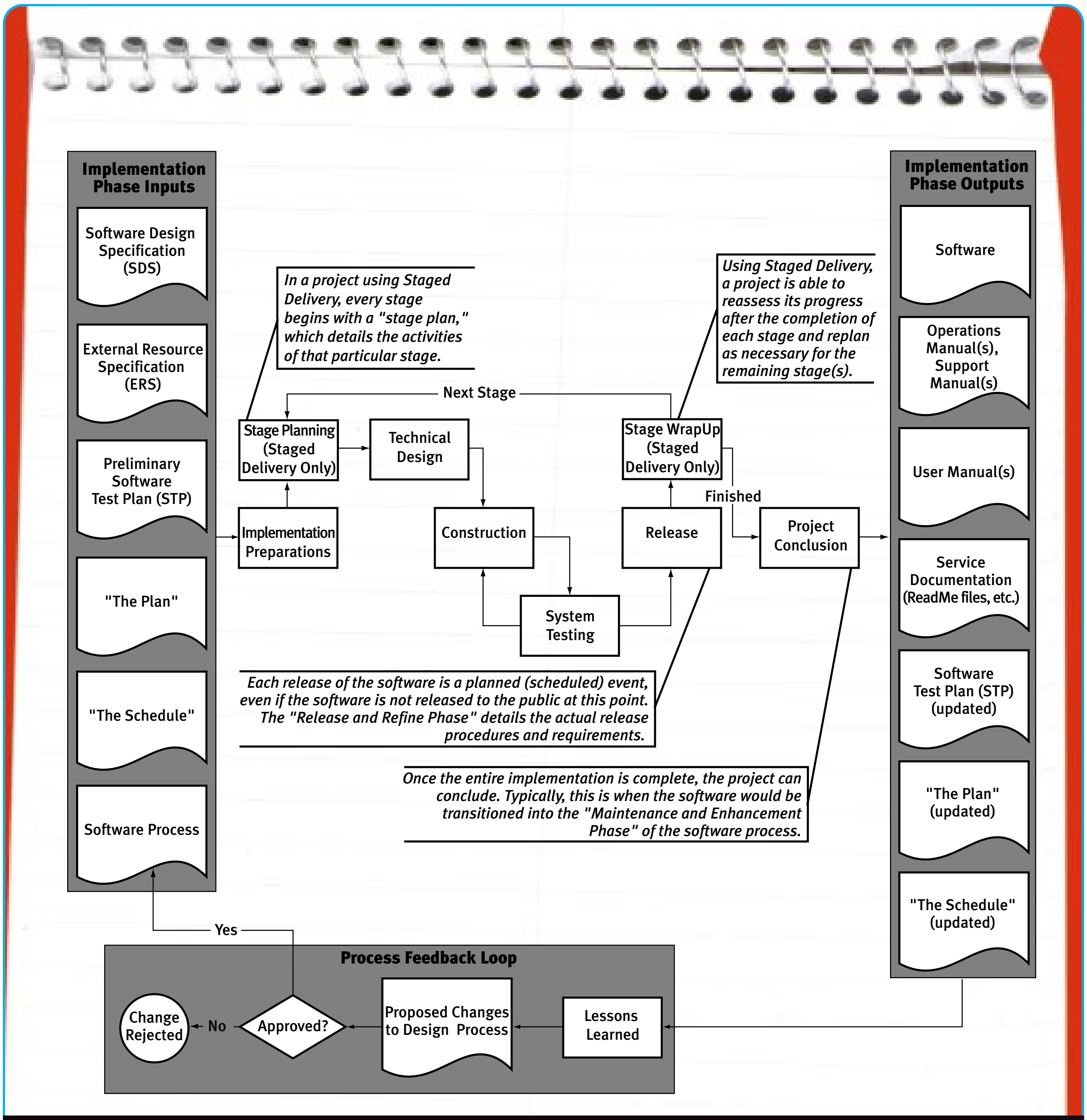

Implementation Phase Inputs
Software Design Specification (SDS)
External Resource Specification (ERS)
Preliminary Software Test Plan (STP)
"The Plan"
"The Schedule"
Software Process
In a project using Staged Delivery, every stage begins with a "stage plan," which details the activities of that particular stage.
Next Stage
Using Staged Delivery, a project is able to reassess its progress after the completion of each stage and replan as necessary for the remaining stage(s).
Stage Planning (Staged Delivery Only)
Technical Design
Stage WrapUp (Staged Delivery Only)
Finished
Implementation Preparations
Construction
Release
Project Conclusion
System Testing
Each release of the software is a planned (scheduled) event, even if the software is not released to the public at this point. The "Release and Refine Phase" details the actual release procedures and requirements.
Once the entire implementation is complete, the project can conclude. Typically, this is when the software would be transitioned into the "Maintenance and Enhancement Phase" of the software process.
Implementation Phase Outputs
Software
Operations Manual(s), Support Manual(s)
User Manual(s)
Service Documentation (ReadMe files, etc.)
Software Test Plan (STP) (updated)
"The Plan" (updated)
"The Schedule" (updated)
Yes
Process Feedback Loop
Change Rejected
No
Approved?
Proposed Changes to Design Process
Lessons Learned

FIGURE 6. *Implementation flow.*

it's almost impossible to accomplish any real changes. The process of making significant changes to production methods takes the commitment of everyone involved, and if management changes directions in mid-stream, all benefits can be lost. Our management and our developers devoted the time and effort that it required.

Our first significant problem was coming up with a vocabulary for everyone to use. Common terms meant different things to different people. This problem was primarily a training problem, and sending the majority of our people to SEI CMM training, combined with in-house project management training, largely overcame this problem. However, due to the complexity of the processes involved with developing games, and the number of departments involved outside of development who don't share our special vocabulary, we still find the vocabulary of disciplined engineering problematic.

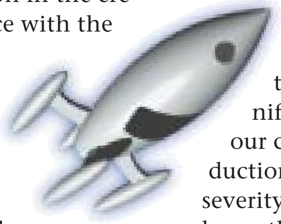
The next large problem that we encountered was the team's impatience to start working (programming) on a new project. Planning a new game down to the final details before writing a line of code (other than necessary "proof of concept" prototyping) is very counterintuitive to most game programmers. Discipline came into play here, as both management and the development staff kept focused on the new processes even when tempted to follow old patterns of development.

Getting all other departments — such as our publishing, product testing, marketing, and operations departments — to cooperate with our new processes was also a challenge. We overcame many of these problems by including these departments in the creation of our processes. A grassroots, organization-wide commitment to process improvement really made a difference for us in this area, but there are always holdouts who prefer the traditional methods despite potential benefits. You need to have the key people in the departments you work with educated and on-board with your planned improvements.

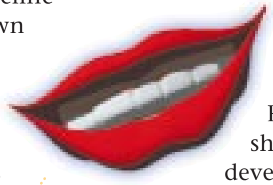
The sheer amount of paperwork and process involved is daunting, particularly for groups used to a seat-of-the-pants development process. I'm sure many people reading this article wonder how any "real" work gets done with all the paperwork involved. We discovered that all this paperwork inevitably gets done, either by a predefined plan or as an emergency later in the project. In our experience, most slippage in schedules is driven by elements that weren't explicitly defined at the beginning of a project. Our process improvement effort simply acknowledged all the known requirements at the beginning of a project rather than in the midst of development when it adds to both schedule and cost.

Benefits Accrued to Date

The biggest benefit that we've realized so far has been improved morale. Our development staff is excited to be involved in a process that is progressive and has the opportunity to take our group to a high level of effectiveness. There has been unanimous participation in the creation of and compliance with the new processes. The process changes are showing results, and developers feel more in control of the project. Mid-stream changes in projects still occur, but because projects are now less chaotic, dealing with these changes is much more manageable.



Getting the project baseline requirements written down and agreed to by all departments was a major benefit. When requirements change (as they often do), it puts you in a position to document the change and inform all parties of the effect it will have on the schedule. It allows cost and schedule trade-offs to be done rationally and reduces needless feature creep. Thinking about and documenting these requirements has headed off innumerable future problems.



Having a full design, a technical design, prototyping technology, and solid estimates of programming tasks at our disposal allows us to plan the creation of the art, sound, and editorial resources better. This has helped reduce the false starts and rework of these elements, in comparison to previous projects.

Management has a much higher confidence in the schedules of the products, and a much better idea of where the product is relative to the projected completion date. The actual schedule exists only after the technical design is completed and approved. We only use broad ranges of time prior to the completion of this phase, which requires a high degree of trust between management and development teams.

Future Expectations

We expect that as our personnel gain more familiarity with the processes, we will see further improvements in productivity. We also believe that we'll reap benefits in our maintenance efforts once our products move to that stage. Our standards will dramatically reduce the time that it takes to train new programmers to maintain legacy products. The time that it takes to find bugs and make additions to these legacy games will be significantly reduced. As we move our current set of products into production, we expect the quantity and severity of found bugs to be much lower than was previously the case. The combination of completely planning the product prior to implementation, peer reviews of key software modules, and test plans created by the team should continue to raise the quality of our software. Finally, our time-to-market should decrease as our software development process improves and our teams become accustomed to the new procedures.

It has been exciting to move towards building game software under our new model. Our new procedures allow our developers to be more effective in a normal workday, and although the occasional crunch still occurs, we no longer face grueling, continuous 16-hour days during a project death march. We have already seen a difference in our shop, with people rising to new levels of contribution and productivity simply because the environment allows it. The standard sweatshop atmosphere of game development only allowed the "coding cowboys" to be heroes. Now, we have teams entirely staffed with heroes, doing great work without having to resort to heroics. Being involved with fostering this move to engineering discipline is the most significant effort that I've been involved with in my 20 years of creating games. I believe it has the potential to take much of the chaos and uncertainty out of development and let us focus on creating great game play in our future titles. ■

FOR FURTHER INFO

- Bennatan, E.M. *On Time, Within Budget: Software Project Management Practices and Techniques*, 2nd ed. (John Wiley & Sons, 1995.)
- Caputo, Kim. *CMM Implementation Guide: Choreographing Software Process Improvement*. (Addison Wesley, 1998.)
- McConnell, Steve. *Rapid Development: Taming Wild Software Schedules*. (Microsoft Press, 1996.)
- Yourdon, Edward. *Death March: The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects*. (PTR Prentice Hall, 1997).
- The Software Engineering Institute's web site (http://www.sei.cmu.edu)

Back

Deer Hunter Th ee-Mo h De elo me C cle

by James Boer

Earlier this year, a small hunting simulation called DEER HUNTER shocked the industry by rocketing to the top of the sales charts for three months. Presently, it's still selling well and shows every indication of continued good sales.

While most hardcore game players stick up their noses at such value-priced games, the truth is that there's still a huge untapped market of computer users who do little with their PCs but write letters, send e-mail, and play solitaire. Death rays and world conquest hold very little interest for them, and they're still waiting for a game they'll enjoy playing.

Although I'm not inclined to debate the merits of games such as DEER HUNTER, let me at least point out a few facts regarding the game that might help to tip the balance of opinion in its favor.

- DEER HUNTER was the first game to attempt seriously to simulate the strategies involved in deer hunting. If you're snidely thinking that there is no strategy to swigging down a beer and storming through the woods after a buck while blasting away with your shotgun, then you have no real knowledge of hunting on which to base a valid opinion. Before I sound too sanctimonious, I should point out that before I worked on DEER HUNTER, I had little knowledge of hunting myself, and shared many of the same prejudices.
- DEER HUNTER is a value-priced game. It costs only $20 and includes $20 worth of value. Many people tend to compare this product against games costing two to three times as much, with budgets often 10 times or more what we had to work with. DEER HUNTER's total development cost was around $75,000. Remember to compare apples to apples.
- DEER HUNTER was targeted specifically at non-enthusiast game players. This meant that both controls and game play were kept as simple as possible. This wasn't only expedient; it was a requirement determined at the outset of the project.
- DEER HUNTER was developed from the ground up by three programmers (one was a college intern) and a part-time artist in less than three months.

It's this last point that I really want to focus on, because it is one of the more interesting aspects of the game, the

James Boer is a programmer, game designer, and musician. During the development of DEER HUNTER, ROCKY MOUNTAIN TROPHY HUNTER, and PRO BASS FISHING, he acted as designer, programmer, art coordinator, sound designer, and voice-talent. He currently resides in Seattle, Wash., and is working at WizBang Software Productions. He can be reached at jbsys@csi.com.

remarkable sales notwithstanding. How the game sold after it shipped was a shock to everyone, including us, but the development cycle was something that was controllable.

At this point you may be saying to yourself, "So what if a small game such as this was written quickly? The product I'm working on is much more complex, has many more features, yada, yada."

This could very well be true. However, many of the lessons learned in a compact development cycle can be applied to any sort of game development project. I've had the opportunity to analyze and fine-tune our development process through five games in one year. Even if a project is expected to require a year and a half to complete, the same strategies can be applied to individual milestones within the project, or even to project components over a much longer period. The techniques don't necessarily have to be used to speed up development, either. They can just as well be used to get more from your existing timelines.

In this article, I outline some of the salient aspects of our development cycle. Some of the points are undoubtedly common sense, but all too often common sense gets ignored in the face of tradition or other external pressures. Other points, however, fly in the face of conventional design approaches. While it's true that not every project could or should attempt to utilize some of these principles, the fact remains that, using these techniques, Sunstorm rapidly produced hit after hit in quick succession.

Lesson 1: Don't Over-Design

It's better to keep the game plan simple and understandable. You don't necessarily have to code the entire game on paper. I've found that the most important aspect of the initial design lies in clearly delegating responsibilities for the project components. Instead of delving into the inner workings of how individual components are going to work, focus on how the major components are going to work together. Again, these don't necessarily have to be completely set in stone. Rather, let the programmers who are dealing with those components work with each other to figure out how their components should communicate. Because they are the only ones the problem affects, it makes sense that they should facilitate the solution, both in design and in code. As long as their solution is solid and well documented, there's no reason to involve other members of the team in the decision process. Planning by committee in cases such as this is both unnecessary and inefficient.

There's one caveat to this technique. One instance in which it does not pay to skimp on design is interface design. Over the course of three projects, I've learned that time spent carefully mocking up all the game screens helps tremendously as a reference for both the programmers and the artists during all phases of development.

Note that I'm not talking about a complete graphical mock-up. I'm referring to a functional mock-up. I use CorelDraw to roughly position all elements on the various screens. I print each screen on a separate page. I label all buttons and explain their functions. I make no attempt to create an aesthetically pleasing screen. The artist's job is to arrange and beautify the various controls except where noted on the document.

A solid interface design not only helps to solidify the flow of the game, it also frees those responsible for its implementation to begin coding and illustrating while the finer points of game play continue to be debated. As the game evolves, this document is updated to reflect the latest changes.

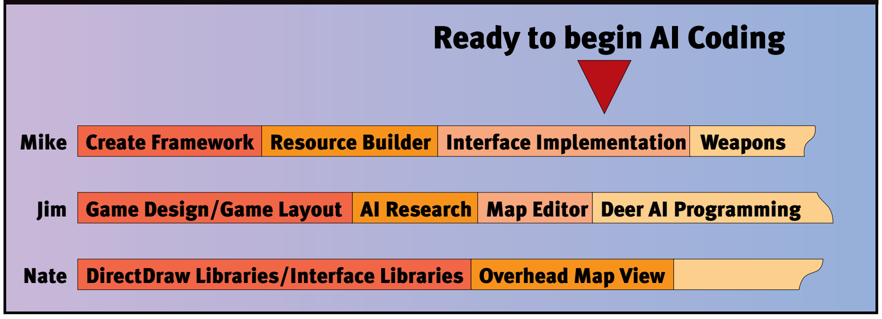
Lesson 2: Don't Be Afraid to Design Dynamically and in Parallel

The original idea for DEER HUNTER was actually simpler than what ended up on store shelves. I joined Sunstorm in May of 1997 with a rough idea of what I'd be working on, but no real knowledge of hunting. My first assignment was to design a deer hunting game. You might imagine how thrilled I was at the prospect of designing a game based on a "sport" that consists mostly of sitting and walking around in the woods for hours on end, waiting for a passive animal to show up so you can shoot it.

The initial game idea wasn't much more than many of the other hunting titles seen previously — that is, blasting a deer when it happened to meander across your path. As development progressed and I became more knowledgeable of the intricacies of hunting and stalking deer, I realized that we could, with a slight change of focus, turn the game from an arcade gallery into a simulation with the addition of an overhead map view and a dynamic scene generator. The biggest reason for DEER HUNTER's success was that we were the first company to make a serious attempt at creating a realistic simulation. Our dynamic development process allowed us to shift focus enough in mid-development to capture this market.

Although some projects have failed due to the lack of a strong design, we found that taking a flexible approach to game design worked in our case, and it might be worth considering on your next project. I'm not advocating that you turn your 3D shooter into a real-time strategy game halfway through development. Rather, try prioritizing design elements

FIGURE 1. *Deer Hunter's task schedule.*



by their expected completion date and finishing designing these elements first.

For instance, if the user interface is rather straightforward but the game play still needs some work, let a programmer and an artist start moving forward on this aspect of the project while others hammer out the details of game play. This is an example of designing in parallel with the project. Because the components have a logical separation from other aspects of the design, this process makes perfect sense and avoids down time.

Likewise, in-house development tools can often be started long before other portions of a game are completed. It may even be a requirement. (For more on this point, see *Game Developer*'s October 1998 Postmortem of Monolith's SHOGO to see what can happen if custom tools aren't given enough attention early in the development process.) If design issues are still up in the air, make sure the tool programmer makes allowances for additional functionality that might be added. The time wasted in having to change a file format is more than made up for by the fact that the programmer didn't have to wait another month for the design specifications to be finished.

Dynamic design — altering the design during development — might also bear consideration. Again, use some common sense — I'm not talking about changing the entire course of a project. We found that it doesn't pay to try to anticipate every problem that may come up during the course of development. Instead, we set goals to attain and expected both the artists and programmers to be creative enough to reach those goals on their own. Many times during the course of a project, we thought of a better way to achieve the goal that we'd set out to reach. One of the more interesting effects in the game is the zoom lens on the rifle scope and binoculars. Although the design of this effect was originally intended to be a simple pixel doubling, I realized that we could make the effect much more convincing if we could preserve the sprites' highest level of detail by using a technique we hadn't thought of earlier. By focusing on the goal instead of the process or the details, those of us working on the project could enhance the game significantly while still maintaining the original schedule.

Lesson 3: Smart Scheduling

I brought up the issue of scheduling when discussing parallel design, but it warrants more attention. Intelligent scheduling means planning to complete components so that you minimize your reliance upon incomplete code; this was one aspect of our development process that allowed us to get DEER HUNTER out the door fairly rapidly.

For instance, Figure 1 shows the task schedule from early in the DEER HUNTER project. Mike Root, our project leader and lead programmer, was responsible for creating and maintaining the general framework of the game, as well as other items such as creating the weapons and building the sound libraries. He also built a resource editor that helped us manage all the game artwork. Nate Terpstra, our trusty intern, worked on the DirectDraw graphics libraries and user interface controls. My responsibilities included game design, artificial intelligence and deer behavior, map creation, and scene generation.

You can see in Figure 1 how both my game design and AI research were scheduled for a time when it would have been difficult for me to do much else in the way of high-level coding. Much of the game design was still in flux when Mike and Nate were developing the DirectDraw libraries, resource builder, and basic program framework.

None of this concerned us, however, as we knew what was required of those components regardless of game design. This freed me to continue tweaking the design and continue research even while other team members were busy coding. Note that when I was finally ready to begin coding the deer AI, the elements for allowing me to visualize the deer on screen were nearly in place.

Lesson 4: Code Around Scheduling Delays

In some instances, scheduling cannot be fudged enough to minimize dependencies and still give enough time for everyone to complete their tasks. Or, there may not be enough alternate work with which to fill up that person's schedule. In this case, the best thing to do is simply to code around the problem. This was somewhat of a problem in DEER HUNTER's development.

I was responsible for the deer's AI and behavior. A problem arose when the map view took slightly longer than anticipated to get functional (with as tight a schedule as we had, even a week could make a huge difference). Without a graphical view into the deer's world, it was difficult for me to maintain my schedule.

In future projects, I would learn from this mistake and create a separate application exclusively for creating and testing animal movement and reactions. The next generation of hunting games were expected to have more advanced behavior, so I needed far more time to program the AI in our later projects. I used a third-party graphics library, Fastgraph for Windows, to create this test bed, because it had native support for a flexible floating-point coordinate system. The animals all used floating-point vectors for movement in a realistically scaled world, so this worked well for me.

Your particular choice of libraries and tools doesn't really matter, as long as you maintain a clear separation between the test bed and the game code that you're writing. This distinction also has another positive side effect: it forces programmers to maintain a clean and intuitive programming interface to the code modules that they're writing, because the code must be moved from the test bed to the main project. At some point, the actual project may become incompatible with the test bed due to increasing complexities in the game itself. Don't try to maintain the two separate projects unless you're absolutely forced to do so. Simply move the code to the main project and continue working there. Keep in mind that the test bed is simply a means to an end, and once its usefulness has ceased, it should be discarded. Don't make it too pretty by wasting time on irrelevant features. Quick and dirty is the key.

Lesson 5: Create Meaningful Milestones

It's too easy for those who can't see under the hood of a project to want visual milestones. Oftentimes, however, when a game looks only half done, the work is actually 90 percent complete, or vice versa. It's important for project managers to understand this. The easiest solution is to involve the designers, artists, and programmers in the task of creating milestones. Because they're the ones doing the work, they should have a more instinctive feel for how to balance out the workload among equally separated project milestones. This also helps to clarify every individual's responsibilities right from the beginning.

I've seen examples of both approaches. When management alone tries to dictate the milestones, the results are often a huge, unbalanced mix of goals, which fall mostly into the "too easy" or "nearly impossible to achieve" categories. When someone who appreciates the technical aspects of the design is involved, the balance between milestones tends to be much more realistic.

You'll notice that all of these suggestions relate to design or other similar aspects of the project. Truth be known, that was really the only remarkable aspect of DEER HUNTER's development cycle. The other parts were pretty plain: solid object-oriented design principles (or solid structured programming techniques for you C programmers), careful programming, and hard work.

The question of code reusability and robustness might come up, and it is indeed a valid point. Minimizing design time doesn't necessarily mean sacrificing these elements. I would offer proof of this by mentioning that the same basic code set was used to create two derivative products, a big-game hunting simulation and a duck/goose hunting game. The success of DEER HUNTER was completely unexpected, yet we were able to use the existing code as an engine with which to create these other games, both of which made substantial improvements to the original game. DEER HUNTER has now even been ported to the Macintosh. There's no question that the code was well-designed and quite robust. Individual programmers sticking to time-honored object-oriented programming techniques achieved the robustness of the code, not a fancy design document.

Remember that design time is essentially overhead. The faster you can get your team up and working, the more time they'll have to do what's really important: produce the game. ■

Back

A I od c io o So d Fil e i g

by Jonathan Blow

We're going to talk about the basics of sound filtering — playing with the various frequencies that compose a sound effect — in real time. In the process, we'll take a small step into the world of digital signal processing (DSP). DSP is a topic that intimidates a lot of people. University courses in DSP come with thick textbooks, and many programmers have heard of this thing called the "Fourier Transform" that consumes terrifying amounts of CPU time. Recently, I was shocked to learn just how simple the concepts underlying DSP are, and how sophisticated effects are easily done using small amounts of CPU time.

Motivation

A goal of most (if not all) game developers is to use interactive, realistic sounds in their titles. Many game programmers are familiar with the concept of "spatialization," which is the practice of manipulating sounds to fool listeners into thinking those sounds are coming from somewhere other than the speakers. Vendors of spatialization products would like you to believe that these products will make your game stunningly realistic; the fact is that, though currently available spatialization is a welcome feature, it's only one of many steps that you must take to achieve realistic sound.

Sound reverberates in closed spaces. In wide-open spaces, sound loses energy as it travels; high frequencies fall away faster than low frequencies, changing the character of the sound. In fact, this change is affected by the weather conditions between the source and the listener. When a sound bounces off a sheer rock face, different frequencies are deflected in different ways. When you're strutting down a bilinearly-filtered hallway cradling your BFG-31337, and you see a nice scripted sequence of a monster disemboweling a scientist on the other side of a thick polymer window, the cracking-bone effects should sound as though they're vibrating their way through a solid barrier — very different than the same event unfolding beside you in the open air.

Some game-oriented 3D sound systems, such as QSound or Aureal's A3D, contain some features for distance attenuation, reverberation, and the like; but those features usually aren't very sophisticated, nor do they give the developer enough control. Because I've never heard a sound API that let me say, "This cannon blast is happening on the other side of an echoey ravine, and it's passing through a damp fog bank on the way here, and the weather is very windy, and by the way I'm listening to the sound underwater," I believe that more game developers should learn to do this stuff themselves. (Some of the very latest games do use sophisticated filtering effects. UNREAL uses a filter to create a reverberation effect inside caves, and HALF-LIFE uses a filter to simulate sonic resonance when the player is crawling through air ducts.)

In this article, we'll cover the basic ideas behind achieving environmental effects through the manipulation of

And when we were all fallen to the earth, I heard a voice speaking unto me, and saying in the Hebrew tongue, jon@bolt-action.com, why persecutest thou me? It is hard for thee to kick against the pricks.

frequencies within a sound. This tutorial will be easier if we first accept a couple of ideas about sine waves, without proof. These ideas aren't too hard to swallow; ample references to explain them are given at the end of the article.

Sine Wave Review

Because we're going to talk about sine waves, let's review some of their properties. Sine waves are little curved beasties made by the equation $f(t) = \sin(t)$. The argument t is given in radians; sine waves repeat themselves every 2π radians. We say that this 2π radians is the wave's period of repetition, and we may think of period in the temporal sense: it is the amount of time the wave takes to repeat itself. The sine function can produce values from –1 to 1, so we say that the wave has an amplitude of 1 (Figure 1A).

To amplify a wave, we multiply it by a constant A, so the equation becomes $f(t) = A \sin(t)$. If A is 2, then every point on the wave becomes twice as far away from the 0 line, and now the wave ranges from –2 to 2 (Figure 1B). To shift a sine wave to the side by an amount x, we add x to the time parameter: $f(t) = A \sin(t + x)$. We can change the frequency of the sine wave by accelerating the flow of time as the sine function sees it; to do this we multiply t by a constant: $f(t) = A \sin(kt + x)$ (Figure 1C). When we add sine waves together, the positive and negative values can cancel each other out, just like any other function (Figure 1D).

The first remarkable fact that we'll accept on faith is this: whenever we add two sine waves of the same frequency, the result is a sine wave of that frequency. This is easy to see if the waves are not shifted: $A \sin(kt) + B \sin(kt) = (A + B) \sin(kt)$, a wave with amplitude $(A + B)$. However, the sum is still a sine wave even when the source waves are shifted: $A \sin(kt + \alpha) + B \sin(kt + \beta) = C \sin(kt + \gamma)$ for some C and γ.

Sine waves of differing frequencies, however, do not sum to produce a simple sine wave. $A \sin(k_1 t) + B \sin(k_2 t) = \text{Something\_More\_Complex}(t)$ in general. But this is a good thing, because sine waves themselves don't make for very interesting sounds. It is by throwing them together and making a mess that we develop true character.

FIGURE 1. *A. Sine wave y=sin(t). B. A sine wave with amplitude 2: y=2sin(t). C. A sine wave with a period of half of wave A.: y=sin(2t). D. When the two purple waves combine, destructive interference produces the green wave.*

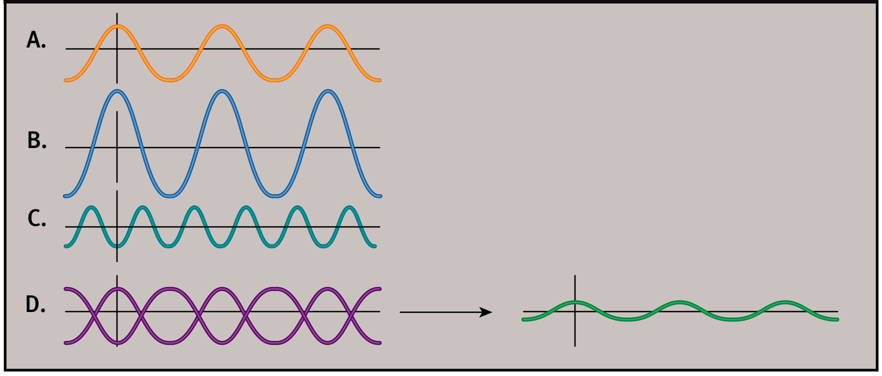
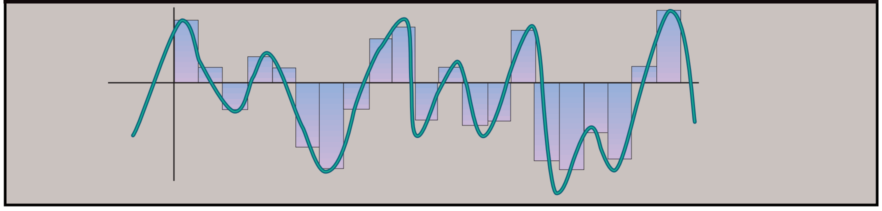


FIGURE 2. *We represent a continuous sound wave with an array of samples.*



Sound Wave Review

These days, we usually represent sounds as arrays of signed 16-bit integers. Each number in the array represents a sample of a sound wave at an instant in time (Figure 2). The sound card converts our samples back into a continuous waveform so that they can be played. We will call this waveform — which is essentially just a function that varies over time — a signal.

The second remarkable fact we'll take on faith is that every signal (over a finite interval) can be represented as a sum of sine waves of differing frequencies. This brilliant realization is called the Fourier Theorem, named after Jean Baptiste Joseph Fourier, who came up with the idea early in the nineteenth century. (There is nothing too magical about sine waves that gives them this capability; the study of wavelets is all about how to decompose signals into varying wave shapes.)

What This Means

Now we are armed with two important facts: all sounds are compositions of sine waves, and two sine waves of the same frequency, added together, produce a new wave of the same frequency. These facts give us the ability to look at sound in an enlightening new way.

Suppose we want to mix two sounds together by adding them; the sounds are represented by functions $f(t)$ and $g(t)$, and we want to add them to produce a new sound $h(t)$: $h(t) = f(t) + g(t)$. The old way of thinking about this is that, for each t, we evaluate $f(t)$ and $g(t)$, add those together, and the result is the value of h at t. According to the DSP way of thinking, we take all the sine waves that make up f, and all those that make up g, then we add them all together to get h.

What happens if we add a sound $f(t)$ to itself? $f(t)$ consists of a bunch of sine waves, $A_n \sin(k_n t + \alpha_n)$. Working in the DSP way, we add all these sine waves to duplicates of themselves, then put them back together to get the result. $f(t) + f(t)$ equals, for each n, $A_n \sin(k_n t + \alpha_n) + A_n \sin(k_n t + \alpha_n) = 2 A_n \sin(k_n t + \alpha_n)$. Every sine wave has twice the amplitude as before. When we put the waves together, it should come as no surprise that the result has twice the magnitude of the original $f(t)$ at every point. $f(t) + f(t) = 2f(t)$, after all. We've made the sound louder. The DSP way of thinking about sounds hasn't bought us any-

FIGURE 3. *A. A sine wave, added to a shifted version of itself, where the shift is small relative to the period of the wave. The wave mostly reinforces itself. B. A sine wave with a shorter period, shifted by the same amount. It ends up destructively interfering with itself, producing zero.*

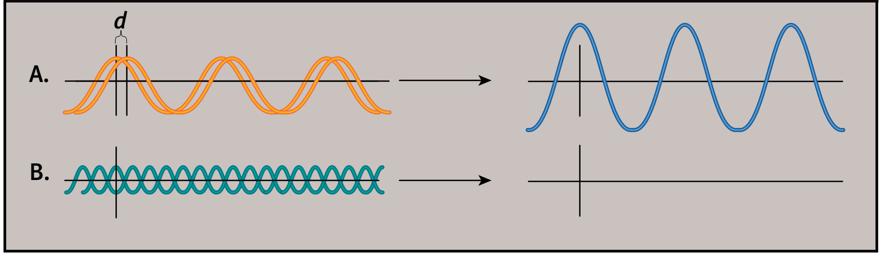
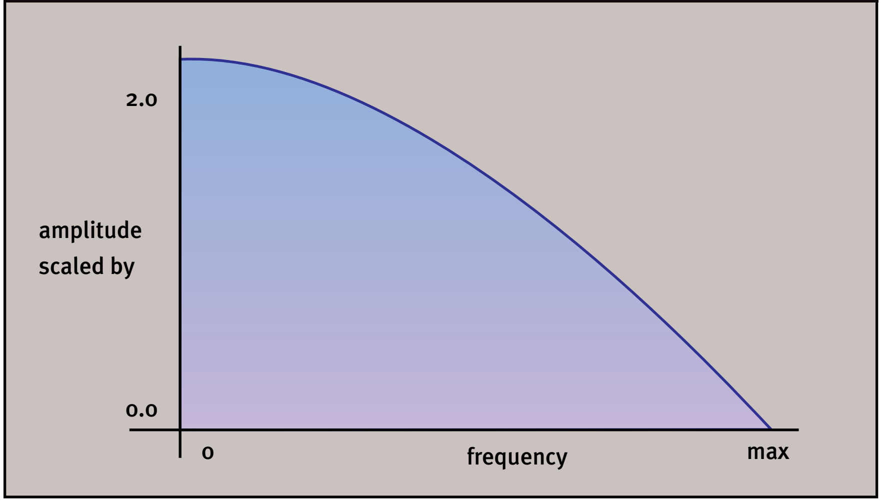


FIGURE 4. *A graph of the frequency response of the low-pass filter in Listing 1. Frequencies are along the x axis; the y axis shows the scale by which each frequency's amplitude will be multiplied.*



thing extra yet — but that was just a way of getting used to this way of thinking and convincing ourselves that it produces consistent results.

Now, here's a revealing thought experiment: what happens when we add the sound f to itself, but before adding, we shift one version of f by a small amount d? That is, if $h(t) = f(t) + f(t + d)$, what does $h(t)$ look like?

Let's look at the individual sine waves in Figure 3. If d is very small compared to the period of a sine wave $A\sin(kt)$, then the wave isn't displaced much with respect to itself. So when we add $A\sin(kt) + A\sin(kt + d)$, the result is very close to $2A\sin(kt)$, as in our last example (Figure 3A).

But what happens when d is longer than the small displacement we just discussed; say, half a period long? We see the result in Figure 3B: $A\sin(kt)$ and $A\sin(kt + d)$ cancel each other out, producing zero.

These cases represent two extremes. The lower the frequency of the wave (the longer its period), the better it will survive the summation process, approaching the ideal of doubling its original value. The higher the frequency (the shorter the period, down to $2d$), the better it will be eliminated by the offset.

It turns out that if d is the width of one sample, the highest frequency contained in the input sound has period $2d$ (so says the Nyquist Theorem, which tells us how much signal information can be stored in a series of samples); this is precisely the frequency that is completely eliminated by the filtering operation. A graph of frequency attenuation is given in Figure 4.

If we take an array of integer sound samples, shift it by one sample, and add it to itself, we are performing a low-pass filter of the sound. If we're representing a sound with 16-bit integers, we want to add the samples using a larger number representation (say, 32-bit integers), then divide them by 2 to ensure that they don't overflow when we pack them back into 16 bits (an issue familiar to anyone who's ever mixed sound). Listing 1 shows code for the low-pass filter that we've just described.

With a simple reversal, we can use this same method to preserve high frequencies and eliminate low frequencies, a process called high-pass filtering. Rather than adding $f(t)$ to itself, what if we subtract it from itself? That is, $h(t) = f(t) - f(t + d)$. This equation is the same as $h(t) = f(t) + (-f(t + d))$; we are negating one of the functions before we add. This has the effect of flipping one of the sine waves about the axis $f(t) = 0$. Now, the low frequencies, because they are hardly shifted at all, negate to cancel themselves out; the high frequencies negate to reinforce themselves (Figure 5).

You can imagine that, by specifying a d that is wider than one sample, one could mute frequencies in the midrange. For example, if d is three samples wide, we cancel all waves with periods of six samples. But a d of this

LISTING 1. *A subroutine that applies a simple low-pass filter to some samples.*

```
void lowpass_filter(short *samples, int nsamples) {
        static long last_sample = 0;

        int i;
        for (i = 0; i < nsamples; i++) {
                long current_sample = samples[i];
                long result = (current_sample + last_sample) / 2;
        last_sample = current_sample;
        samples[i] = result;
        }
}
```

value will also filter waves with a period of two samples (because those waves are shifted by 1.5 times their period, which is the same as shifting them by .5 times their period). This behavior would seem to place some heavy restrictions on the frequencies we'd be able to filter. It turns out, though, that by doing some geometry in the complex plane, we can obtain good control of our frequency response curve, including frequencies that cannot be represented by integer multiples of *d*. See the References section for more information about this. This type of filter is called a feedforward filter because, in producing output, it only uses past values of its input.

Resonance and Feedback

Anyone who's been to an early-90s grunge concert knows what feedback is. This goateed guy holds his guitar too close to the loudspeakers. The guitar pickups (essentially little microphones) catch the loud noise coming out of the speaker and transmit that noise back to the speaker, which tries to make it even louder. Soon, the audio system overloads and we're left with an angry screech.

We can apply this same idea to our audio filter, but in a more restrained and generally soothing way. What happens if we generate our sound $h(t)$ by combining $f(t)$ with previous values of the output, $h(t)$? For example, $h(t) = f(t) + h(t - d)$? For simplicity's sake, we'll measure t in samples, and d will be 1 sample: $h(t) = f(t) + h(t - 1)$.

We can evaluate the first few terms directly: $h(0) = f(0) + h(-1)$. We'll assume that all negative values of h are zero, so $h(0) = f(0)$. That's simple enough. Also, $h(1) = f(1) + h(0) = f(1) + f(0)$. So at the second sample, we're adding f to an offset version of itself, just as before. Now, $h(2) = f(2) + h(1) = f(2) + f(1) + f(0)$. As we repeat this process, we see that the pattern continues forever. We're adding together versions of f that are shifted by successive amounts.

What does this process do? Figure 6 shows the result for waves whose periods are much longer than two samples. As we saw before, adding the wave to a shifted version of itself will cause the wave nearly to double in amplitude. But now, to output the next sample, our feedback loop shifts that result wave again and adds it back to the original. Now, the result is even greater in amplitude — though because it's shifted further, it doesn't reinforce itself quite so strongly. After enough repetitions, the wave will become out of phase with the original version of itself, and the feedback will serve to reduce its amplitude rather than increase it.

We have to be careful with this kind of situation. Figure 6B shows what happens when we feed back a wave with a delay time that is a multiple of its period. The amplitude of the wave climbs toward infinity, which leads to total chaos. For this reason, we must multiply feedback signals by a constant of

FIGURE 5. *A. We perform the kinds of summations that took place in Figure 4, but first we negate the orange waves. The dashed light blue curves represent the position of the waves before negation. Now, the low frequencies, as in A, are killed; the high frequencies, as in B, are doubled.*

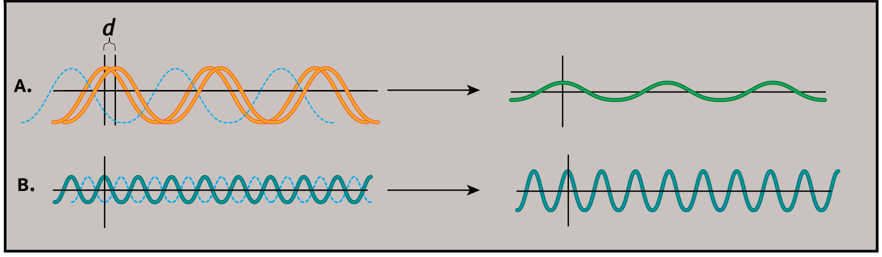


FIGURE 6. *A. The effect of feedback on a wave where* d *is misaligned with the wave's period. As we add it to shifted versions of itself, the result will wax and wane in amplitude, never growing beyond a certain point. B. If* d *is aligned with the wave's period, the wave will continue to grow with each round of feedback.*

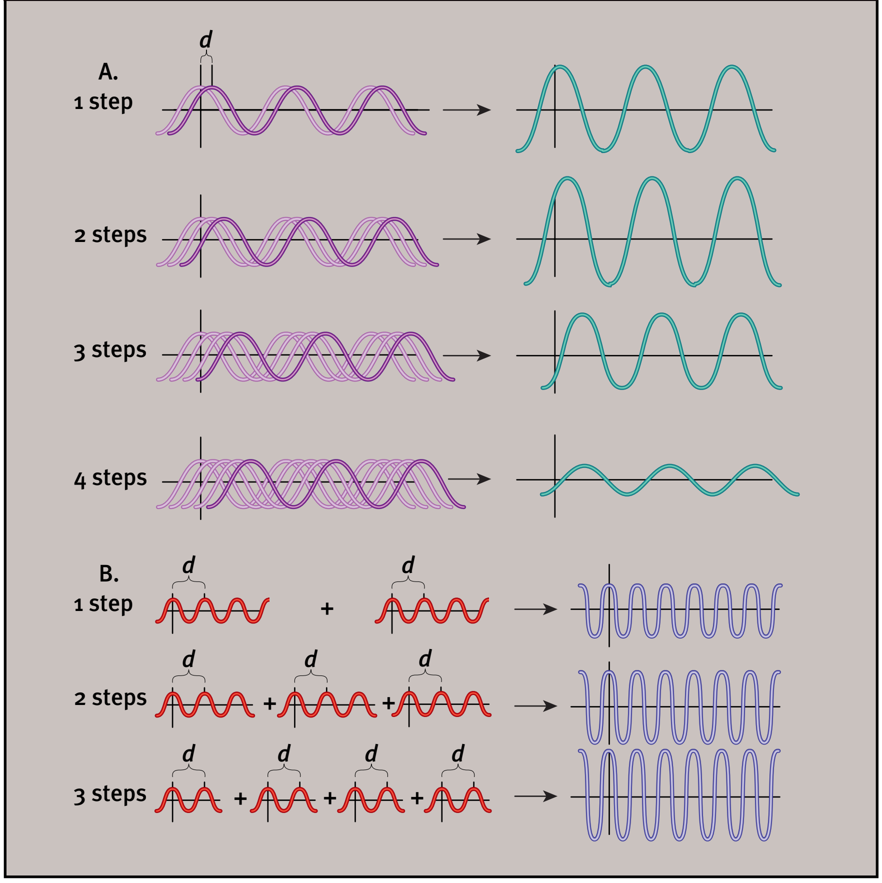
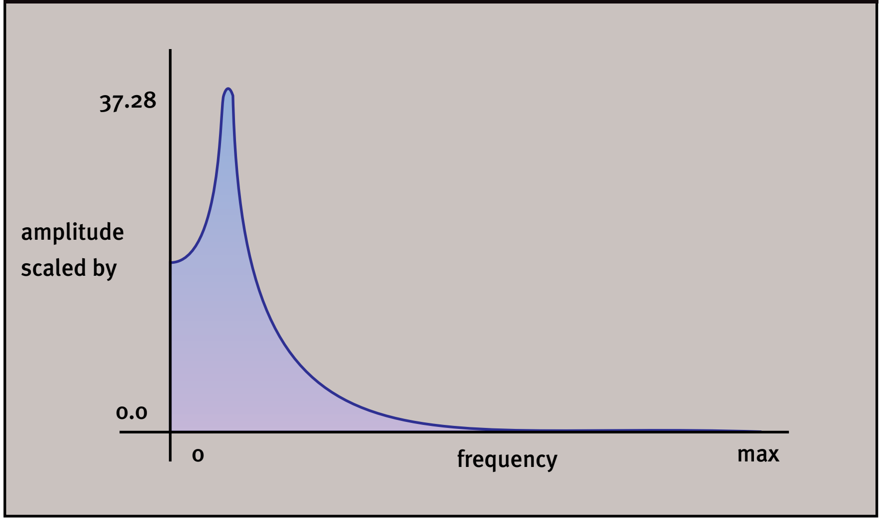


FIGURE 7. *The frequency response from Listing 2.*



magnitude less than 1, to damp them. Thus, $h(t) = f(t) + k\,h(t-1)$, $-1 < k < 1$. Then the feedback filter is stable.

Listing 2 shows code for a simple feedback filter. The frequency response of this filter is shown in Figure 7. Note the relative sharpness of the peak — feedback filters allow us to alter frequency response more dramatically than the feedforward filters we talked about earlier. Note also that it amplifies some frequencies by large amounts. To keep our number representation from overflowing, we compute the maximum amplification when we build the filter; later, as we output samples from the filter, we divide them by that factor to normalize the sound, thus preventing overflow.

As Figure 7 shows, feedback allows us to pick out certain frequencies and amplify them much more than others. It is said that the filter resonates at those frequencies. Interestingly, this is the same effect that occurs with sounds made in enclosed spaces. They resonate according to the shape and dimensions of the cavity, whether it's a yawning cave or a Stradivarius violin. Many physics books describe the reflection of waves between two surfaces, and in such descriptions, one can easily see the mathematical equivalence.

Implementation

In our feedback filter, we ended up multiplying samples by floating-point values to damp them. For most feedforward filters, we want to use floating-point math as well. Generally, we'll convert our 16-bit source samples to floating point and operate on the floating-point samples. For cases in which we're algorithmically generating a sound, instead of reading that sound from a file, our original samples will probably be in floating point.

Interestingly enough, new instruction sets such as AMD's 3DNow and Intel's new Katmai instructions contain stuff that's great for processing sound in floating point. Unfortunately, almost every sound API (3D or not) wants samples fed to it as integers. Therefore, we must convert our floats back to integers, consuming a little extra CPU time and reducing accuracy. These extra calculations are especially irritating because the sound library usually will convert the samples right back to floating point (for example, to scale the volume of the sounds so that they can be mixed into a single channel). Let me offer this plea to the makers of sound APIs: give us an interface that takes samples in floating point!

The example code that accompanies this article will demonstrate the effects discussed here, along with some others, such as pitch shifting and reverberation. (Reverberation works just like the filtering examples we've discussed already; the main difference is that the filter delay *d* is much longer, to the point where the delay is audible.) The code is available on the *Game Developer* web site.

Now You Try It

We've seen some low-level ways to manipulate the frequency content of sound. We haven't said much about how to put sounds together, though. If an explosion happens inside a sealed vault, you probably want to low-pass filter that sound for listeners on the outside; but exactly what frequency response to use, and how to make a filter to give you that response, could be the subject of an entire book.

We've tried to present sound filtering in an introductory way that is different from the approach taken by most textbooks. That way, upon further reading, you'll be exposed to ideas from a different direction, which can provide fresh insight. We certainly have not been rigorous here. In fact, we've tried to use as few mathematical ideas as possible; some of the References present filters in an elegant way that requires more background. In any case, further reading is required to accomplish a sophisticated understanding of filters. ■

REFERENCES

• Everybody in the world should read *A Digital Signal Processing Primer, with Applications to Digital Audio and Computer Music* (Addison-Wesley, 1996) by Ken Steiglitz. Starting with some simple mathematical exposition, it charges straight through DSP theory in the most accessible way I've seen to date.

• After that, read section 14.10 of *Computer Graphics: Principles and Practice* (Addison-Wesley, 1990) by Foley, van Dam, et al. This section of the book talks about signal processing with respect to graphics, and provides some good visualizations of what it means to filter a complex signal.

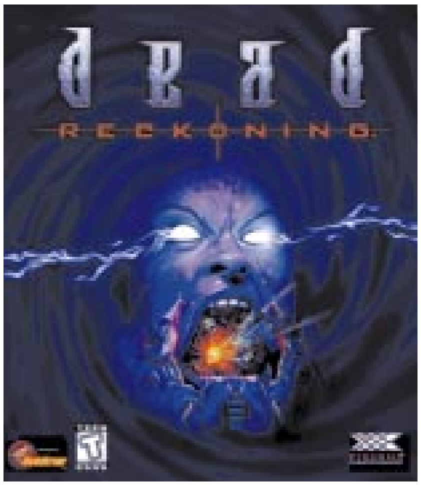
• *The Science of Sound* (Addison-Wesley, 1990) by Thomas D. Rossing is a great book that discusses all aspects of sound emission and reflection, with a focus on musical instruments and the human voice.

• *Principles of Digital Audio, 3rd ed.*, (McGraw-Hill, 1995) by Ken C. Pohlmann, contains some good chapters about the Nyquist Theorem, signal-to-noise ratios, and signal dithering (a topic that will be of interest to those who wish to generate sounds algorithmically).

Gold ee DEAD RECKONING

by Luke Ahearn

DEAD RECKONING is Goldtree's latest title and is a new chapter in first-person vehicle-based games. As a player in the, game you've been abducted by the Master Race to engage in gladiatorial combat against the best warriors of every know alien race in the universe. If you loose, Earth and all of mankind is destroyed. Before you begin, you choose your wingmen from among many characters — each with his or her own personality; personality traits include loyalty, obedience, aggressiveness, and more. And you choose your ship. Each has varying rates of firepower, shield strength, and speed. Most important



Luke Ahearn has been Lead Designer and Producer at Goldtree for five titles, including DEAD RECKONING. He is currently developing Goldtree's next title. Before games, Luke wrote novels, worked in the film industry, and did every dreadfully boring, menial job you can think of while going through college. He can be reached at luke@goldtree.com.

are the special weapons; chameleon powers, mines, traitor bombs, holographic decoys, and the Death Blossom are a few.

Combat takes place in huge cylindrical arenas, each varied in environment. There are arctic wastelands, the tombs of Egyptian kings, and asteroid belts. You can fight in a lake of lava or the innards of a giant creature. Goldtree wanted to take players off of the floor in QUAKE and out of the confining tunnels of DESCENT. You are in arenas where you have room to maneuver, but the firefights are quick and intense. The game demands more than simple twitch reflexes. You must select the right ships for you and your wingmen, give your wingmen orders, and monitor your resources. You must tag pylons to increase your weapon's strength, hover over the energy field to recharge your shields, and above all, kill your enemy.

As the game's designer and producer, I will focus this Postmortem on DEAD RECKONING's design. In some ways, DEAD RECKONING is a remake of our previous title CYLINDRIX; in other ways, the two games are worlds apart. CYLINDRIX was greatly admired by many game players and received great exposure online for its game play. But it lacked certain mass-market features such as textures and solid net play, and we were plagued by many problems while trying to publish CYLINDRIX.

After we finished CYLINDRIX, Goldtree was ready to produce the A+ title that we knew we were capable of producing. We were discussing what that title would be when Piranha Interactive expressed interest in a sequel to CYLINDRIX. Eventually, Goldtree and Piranha decided that we shouldn't make a sequel to a game that the world-at-large hadn't heard of, so we made DEAD RECKONING a game of its own. We felt this decision was legitimate, as the differences between the two games are vast in most areas.

The completion date for DEAD RECKONING was pushed off for almost a year, but the big delay was primarily due to the fact that Piranha believed that DEAD RECKONING would be an A+ title and didn't want to squander its resources on a half-baked launch. The company saw opportunities to improve vastly the game's value in the marketplace. Piranha seemed immune to the popular attitude that a publisher can dump anything into a box and then rely on sales and marketing to make the game successful. Developers see things from the opposite point of view. A great game will sell by word of mouth as the demo works its way rapidly across the Internet.

Piranha's long-term thinking and support allowed Goldtree to redevelop some critical areas of DEAD RECKONING, including support for online gaming services, Microsoft's force-feedback joystick, and the ability to support user-created levels. Supporting custom levels included a great deal of reworking of the game code and the level editor that we used during development. For the level editor, we hired Rosetta Game Development.

The level editor was never intended to be used by the end user, and as a result was undocumented and pretty hard to use. While I was writing the user manual for the level editor, Rosetta started improving the editor itself. With very little time to work on the editor, they couldn't make it perfect, but were able to fix some bugs, make usability improvements, and add significant functionality.

The level editor has an object-oriented design based on "entity" classes for each kind of game object, and it shares these classes with the game itself. The editor is a placement tool only, so new object shapes must be created in an application such as 3D Studio MAX and then imported into the level. The editor allows users to manipulate all of the gameplay properties of an entity, such as the recharge rates of an energy field, the explosion file that's played when a certain entity is destroyed, and the start points for players.

The usability improvements to the editor included single-level undo; keyboard shortcuts for commonly used operations; arrow-key nudges for fine motion; browse buttons everywhere a filename must be specified; and consistent dialog naming, layout, and keyboard operation.

DEAD RECKONING's development team — Goldtree: (from left to right, top row) Luke Ahearn, Designer/Producer; Anthony Thibault, Lead Programmer; Nicholas Marks, Art Director; (bottom row) John McCawley, AI Programmer; Michael Freimanis, Menu Programmer; Josh Eustis, Musical Director and Engineer; (not shown) Ted Baldwin, cut scene animation; TJ Bordelon, consulting programmer; Dan Farrell, Menu Prototype. RA Studios: (from left to right, back row) Richard Laquale, Brian Kennedy, Daniel Venditelli, (front row) Stephen Capasso, Matthew Zanni. Rosetta: (not shown) Scott K. Warren.

DEAD RECKONING

Goldtree Game Developers
Metairie, La.
(504) 837-0080
http://www.goldtree.com

Team Size: Eight in-house. RA Studios completed the game and demo, adding support for many features and fixing bugs. Rosetta updated the level editor for users in the contest.

Release Date: September 1998

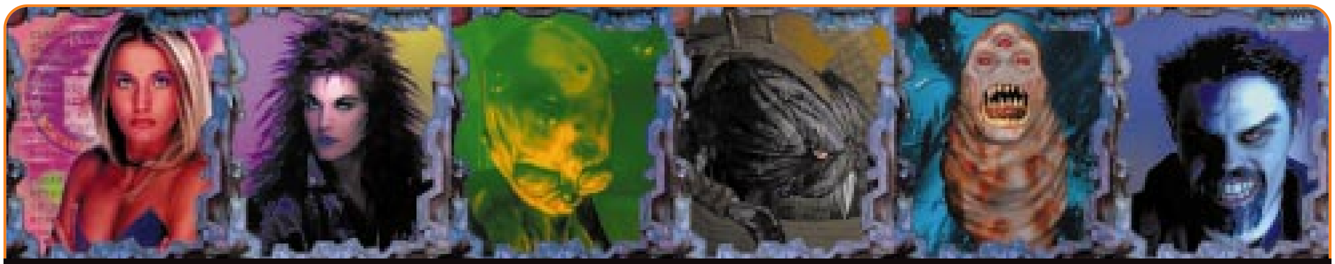
Title Budget: $350,000 (ballpark)

Time in development: 24 months

Intended game platform: Windows 95/98 with DirectX 5 (will run solid under DirectX 6). We are in the process of recompiling for DirectX 6 and expect a significant improvement in performance.

Critical hardware: A network of eight Pentium 200s that allowed the rapid compiling and testing of the game. Audio setup with several large drives, CardD and digital daughter card, DAT recorder, ROM burner, and a ton of really cool musical stuff the musician brought in.

Critical software: Direct X 5.2a, Photoshop 4, trueSpace 2, 3D Studio MAX, Sound Forge, Cakewalk, Visual C++.


In its review of DEAD RECKONING, *the Adrenalin Vault compared the game's cast of characters to the bar scene in* Star Wars.

Some of the primary new features that Rosetta implemented were the Turret Property Sheet, the Find Object dialog, the Level Summary dialog, and the ability to automatically import assets. Turrets are the most complicated entity in the game, yet our original editor provided no support for altering their properties, which were specified by an external text file. We had Rosetta add a fancy tabbed dialog so all properties could be seen, edited, and checked for validity within the editor. The Find Object dialog displays a scrolling list of all the entities in the level. Users can filter the list by entity type and can select any listed entity for editing. The most interesting feature of this dialog is the Look At Object button, which repositions the camera to bring the selected entity into view. Heuristics are used to pick a suitable camera pose and distance from the object, with the camera coming in closer for smaller objects. Unfortunately, we didn't have time for Rosetta to make these heuristics foolproof, so it's possible that the looked-at object will be obscured by something in the camera's way. In practice, I've found this to be rare. The Level Summary dialog not only lists statistics about the objects in the level, but also offers a Verify button, which checks whether every asset mentioned by the level has actually been imported into the level.

While Piranha was investing the additional time and resources in the marketing of DEAD RECKONING, bringing on Epicentric to help, they also cosponsored a level-design contest. This contest was the reason we rebuilt the level editor. Piranha also retained some serious marketing talent internally to help push DEAD RECKONING.

And, of course, Goldtree took this opportunity to optimize the code and fix many bugs. For this effort, we brought in RA Studios. By the time RA Studios received the code for DEAD RECKONING, the game was almost complete. Once again, time was short; the game needed to be ready to go gold within two months, so the programmers at RA needed to get acquainted with the code very quickly. We tasked them, primarily, with getting a better frame rate out of the engine, which was running at about 9-15 FPS on a Pentium 200 MMX. Better, in this case, meant 200 percent better, so trying to find the culprit for this performance problem was paramount. Because the engine was already complete, as was most of the game around it, it proved difficult for them to fix the speed issue without breaking the game. However, in the end, with tweaks to the collision routines and some rendering routines, RA achieved a comfortable 30 FPS.

The original storyboard and the rendered version of the final victory cut scene.

Design, Design, Design

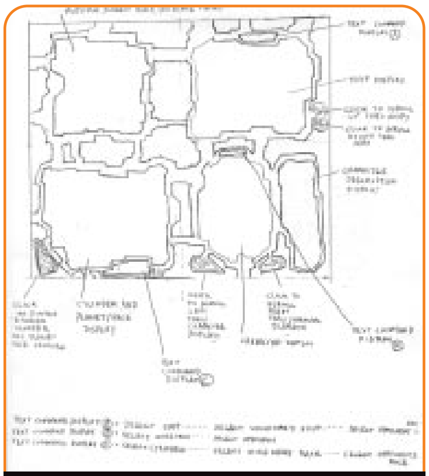
Looking back over the last two years, I can say that for all of the challenges that besieged us during the development of DEAD RECKONING, good up-front design is what guaranteed the delivery of a high-quality finished product. I believe that in a game company, design issues are the most important aspects of a project, even above business issues. Although written 15 years ago, *The Art of Computer Game Design* by Chris Crawford still holds some great advice and observation about game design. He says, "There is no easy way to produce good computer

A rogue's gallery of characters from DEAD RECKONING.

games. You must start with a good game designer, an individual with artistic flair and a feel for people." I believe the best approach to designing a game consists of working in the following three areas.

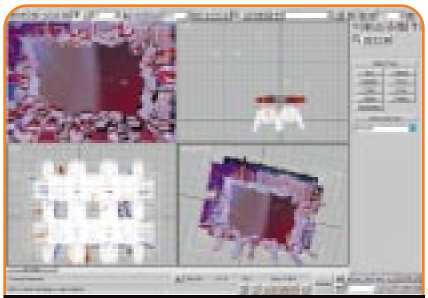
Obviously your primary area of absorption should be in gaming, especially within your genre (first-person, strategy, and so on). This effort should include not only playing games, but reading all the magazines, surfing the Net and browsing through news groups. From a design standpoint, I find that it's easy to lose touch with what a good game is by simply taking your eye off of the moving target (the game market as a whole) for even a brief period of time.

I also find it very helpful to read books, watch movies, and pursue other creative fields. There's much to learn from how a movie is scripted and storyboarded that applies to game development. Understanding how a novel or short story is written and constructed (and what actually makes a good story) helps a great deal when detailing your game on paper. I find that most of what I do is reading and writing. Writing skills are used in business reports, game treatments, product submissions, anonymous flames… the list is endless.


A concept sketch of our original menu idea. After mocking it up we determined that it was too unwieldy and inflexible. The time was not wasted.

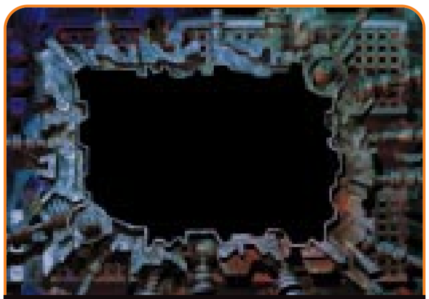
Finally, I turn to traditional project management a great deal for the tools that are critical in developing any serious undertaking. You simply must take the time to develop the work breakdown structure, master schedule, project budget, and other necessary documents in order to maintain control of your development. As W. Edwards Deming points out, "Fire prevention is far better than fire fighting." And there is no better way to instill the publisher with the confidence that you understand the job that must be done and can do it.

The initial stages of open discussions on DEAD RECKONING went well. We had time to think long and hard about what we liked and didn't like about CYLINDRIX and what we would do differently in the design and development of our next title. Like most developers, we wanted to improve on the concept and technology we understood best, and not start from ground zero again. We decided to stick with the approach that we developed for CYLINDRIX: a hardcore game that required that players be seriously good at gaming, well rounded, and not just fast. We felt that enough people had become so well acclimated to 3D games such as QUAKE or DESCENT, that they were ready for something harder. No cheat codes or camping, but not just toe-to-toe combat either. Slower players on the controls should still be able to defeat fragmasters by being more strategic and managing their wingmen, weapons, and overall battle plans better.


We built the menu as a 3D model in 3D Studio MAX.

Originally, we had no intention of allowing difficulty levels to exist other than nightmare, and we even made the enemy AI better than your computer teammates later in the game. After some initial play-testing by the publisher and reviewers, we realized that this intense gauntlet actually made some people really tense; we made some concessions (although they were few). We allowed an automatic save game option that starts you at the beginning of the level where you died. We also toned down the AI a little. Games are supposed to be fun.

On the other hand, as game enthusiasts, our proficiency in games was deceiving to us and caused a bit of design drift. After countless hours of QUAKE, DUKE NUKEM, DESCENT, WARCRAFT, and all the other really great games out there, we got *really* good. We felt as though we had a handle on the design issues early on that could prevent last minute work,


The background for the main menu.

rework, and confusion. But we also got so good at gaming that we lost touch with our audience. So we had to tone the game down a bit.

In addition to all the game play issues, we tackled several important technical decisions early on in the design process. We choose to support DirectX and Windows 95 because it looked as though the industry was moving that way, and we liked the idea of not having to write for specific hardware. One of our frustrations during the development of CYLINDRIX was that a great deal of our programming time was spent writing for hardware and not for the game. We took one look at 3D hardware performance and knew we had to support it.

Another benefit to having your game designed early and in terms a publisher will understand is that it gives you a firm footing when approaching and dealing with a publisher. While we positioned DEAD RECKONING as a game enthusiasts' game, we also had that "first-person, 3D, textured polygon, hardware-accelerated, multiplayer, super dooper thingy" buzzword stuff going for us. Initially, an unnamed individual at Piranha was pushing us to make the game "like DESCENT" (he doesn't work there anymore), which would have doomed DEAD RECKONING to "me too" status. I was prepared with a well thought out game design that addressed our publisher's desire for uniqueness and familiarity. Piranha listened, bought into it, and backed us all the way.

The "different but same" problem (as first brought to the American consciousness by the Karate Kid) is similar to that which faces writers. Aristotle defined the major plot situations that exist in drama and as far as I know, no one has added to them. But think of all the unique and great literature (not to mention films, radio, computer games) that has sprung up since the classical Greek period. Our genres are defined for the near future, and it is our job to be creative, unique, resourceful.

Design work is not all talking and sketching. We actually made prototypes and designed as much as possible using various software tools. Even with limited time and budgets, we used flow charts, storyboards, and fleshed out the look and feel of the menus and game flow as much as possible. We made a prototype of the entire menu system in Director and saved a great deal of work in the end, having been able to try out several versions of the menu first. We also became aware of how much illustration, fiction, and graphics had to done before we began working in other areas. DirectX allowed us to work in retained mode, which isn't good enough for a real-time 3D game, but did enable rapid prototyping.

What Went Right

1. PLAYING GAMES AS MUCH AS POSSIBLE AND BEING SERIOUSLY TAPPED INTO THE GAME MARKET. Of course, this fact is obvious to developers and need not be mentioned to a true game enthusiast. But for all you suits out there, you have to play games to make them. You have to go on the newsgroups, read the magazines, and surf the Net incessantly. I've actually heard that some companies (game companies) have instituted a "no games" policy. What is the world coming to?

Without a deep and intimate knowledge of games and the marketplace, we cannot design good games. And, in actuality, we aren't designing games for today, but for the marketplace two years in the future. We all know how rapidly this market changes. In other areas, even in other areas of application development, one doesn't need to be as tapped into the market and core audience as we do. We have to anticipate the desires of a group of people (game players) that will invest heavily in upgrades and new technology when it arrives and change the game development landscape dramatically in a short time.

In defense of the suits, you do have to make games to sell them. During DEAD RECKONING's development, we erred on the side of too much play while trying to keep stress levels down. I feel that the title still benefited more than if we had taken the opposite approach. We were aware of what players wanted and could more easily guess at what they would want in the future.

2. 3D HARDWARE SUPPORT. If there was ever one thing I knew for sure, it was that 3D hardware would take off. I knew that just like the sound card, 3D acceleration would become a standard, something that you just had to have if a game was to even make it. Unfortunately, I didn't buy stock — but I knew. We decided to support software rendering as well, because we thought the game would be out a little earlier and we wanted to reach the largest possible user base. Had we not supported 3D hardware, especially after the delay in completion, we would have been sunk or delayed even further. DEAD was designed to run well on any 3D card, and run extraordinarily well on a 3Dfx-based card. DEAD uses MMX as well.

3. CHOOSING THE RIGHT PEOPLE TO WORK WITH. Finding the right publisher is something I spent a lot of time on as well. I ended up with Piranha Interactive and have not regretted it for one second. I specifically targeted a mid-sized publisher with the main purpose of protecting the game itself. I wasn't ready to tangle with the monster companies, although as a matter of education, I approached and spoke with many. I ended up with a handful of smaller but promising publishers. The benefits for a freshman developer are more attention and collaboration — your design ideas are treated more seriously. With a larger publisher, you run the risk of getting lost in the shuffle and being told, "Do it this way, or else."

In the process of educating myself as a developer over the past several years, I read a lot, spoke to everyone I could, and attended all the major con-

ventions (E3, CGDC, Comdex, and so on). I heard all the horror stories about developer-publisher relationships and have a few of my own. I've come to see how many of these situations crop up, and rarely is it blatant malevolence on either party's part. Mostly, I fear it's the vast difference between how a guy in a suit thinks and how the developer thinks. There is much room for misunderstanding.

And when putting together the team for DEAD RECKONING, I looked for passion and attitude. There is nothing more valuable. Attitude is what keeps a person working through such a long and trying project as a computer game when there seems to be no end in sight. Passion is what makes one's work stand out. We had plenty of passion and attitude in all our team members. I'm still amazed that we did what we did with so few people and so few resources.

4. DIRECTX. This decision was made early on mostly because it seemed as though it would cut out a good portion of work for us, and it did. It allowed us to focus a great deal more of our energies on the game. The DirectX beta team was responsive and helpful. DirectPlay was particularly helpful, as was Direct3D Retained Mode for creating the level editor. We actually looked at other APIs, but not for long. We felt that DirectX would become the standard and be the best-supported API. I likened this decision to choosing the operating system for your game. You're going to want the largest market possible.

5. WORKING WITHIN OUR LIMITS WHILE SETTING HIGH GOALS. This is good design practice, especially for a small team. Working within one's limits is a conscious, not constrictive, process. Designing a really great game means leaving out features that you simply cannot accomplish. It prevents you from setting out on a journey you could never complete.

We started by listing what we all felt we could do well and what we felt we couldn't do well. For example, we concluded that implementing really good character animation would be more work than we could handle, and I believe we were right. We focused on the physics and behavior of the ships. We wanted time to make each arena unique, even down to the sounds and projectiles of the turrets and obstacles in each game level.

We also went from trying to create character art in 3D Studio MAX and by hand illustration to photographing models and photo-manipulating them with Power Goo, Photoshop, and a little 3D augmentation. This turned out to be a better route all around. It was cheaper, quicker, and much nicer looking.

What Went Wrong

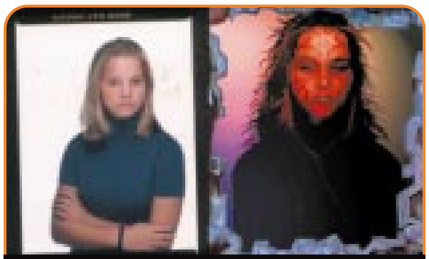
1. DESIGN DRIFT. Design drift can be very subtle or blatant. It can be individuals tearing out on their own paths during development or an unconscious drift by the whole team, much like the example that I gave earlier: we became too good at our own game and made it a bit too hard. We had to redo many aspects of the levels, AI, and game play.

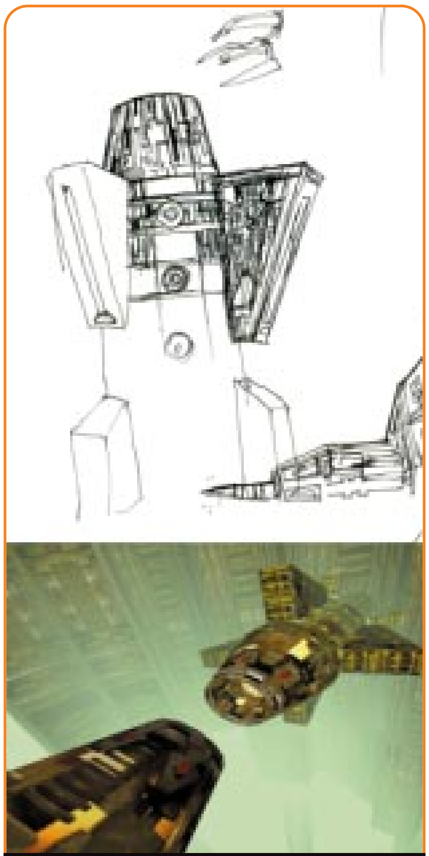
Be mindful of design drift. Publishers are more likely to be too rigid when allotting more time or money for a neat feature, but developers tend to want to make the perfect game and go overboard. In the case of DEAD RECKONING the publisher wanted more time and money spent on the title and supported Goldtree's decision to make a major shove at redeveloping portions of the game at the end — giving us more time, money, and support to make DEAD RECKONING a better game. But this was a conscious decision by both parties and not a drift.

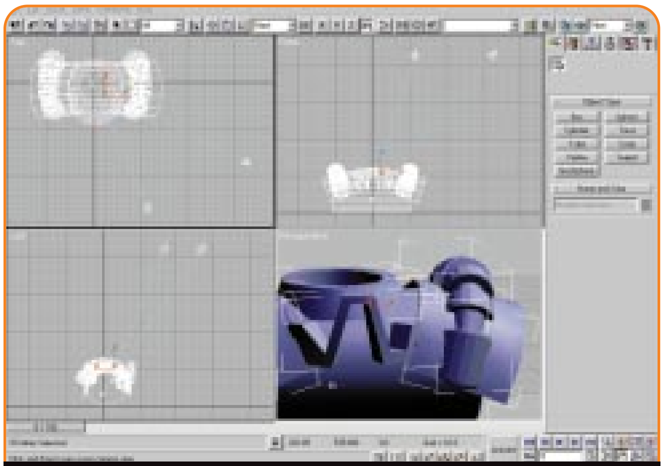
Drifting is when you're unwittingly off course. One thing that's useful is to keep the vision in the hands of the producer, project manger, or one person designated to filter all changes through the entire team.

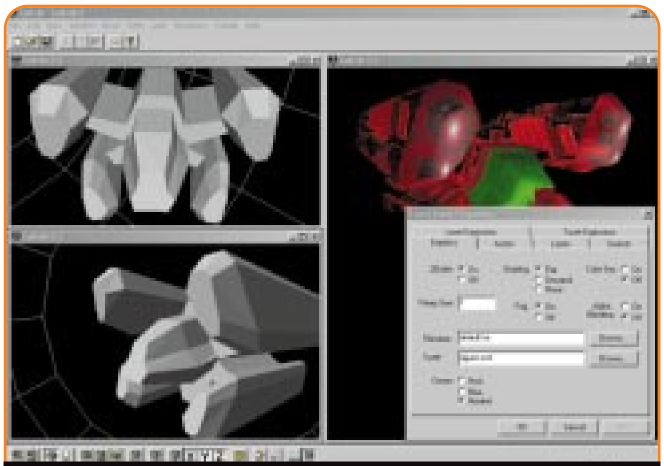
We faced many drifts that were usually the result of talented and perfectionist individuals trying to make the game as good as possible. It's a hard judgment call to make when dealing with talent trying to perfect some aspect of their craft. Do you stop their creative process to make the milestone, knowing that you may impact the quality of the game, lower morale, and/or cause the passionate individual unconsciously to adopt a new lower standard?

Sometimes, decisions were made by individuals that affected the rest of the development and these decisions were


The more human-like characters ("Panther" is on the right) were created from photographs. In addition to Photoshop work, you can see where the artist used Goo to reshape the head for a more feline appearance.


The concept sketch and an in-game screen shot of the Energy Square in the Sentry arena.


The armor for the Biomechanoid character was built in 3D Studio MAX. The artist took a snapshot of the finished model, which was then composited with the rest of the character's image using Photoshop.


The DEAD RECKONING level editor.

not communicated to the rest of the team. For instance, the decision to take the animated sequences down from 640×480 to 320×200 upset the animator. His work was deteriorated and he wasn't told that he didn't need to render scenes out in 640×480. This wasted his time.

We also faced some small drifts that were caused by the relative meanings of plain old words in the English language. A programmer may say "large file" meaning a 100K and the artist may think 100MB.

Also, 3D game modelers are often different animals than the commercial 3D modeler. A game modeler knows what low polygon count means and is adept at building objects face by face. A commercial or animation-grade modeler often builds everything using multiple primitives and very few texture maps. Usually, after explaining that the gorgeous 30,000-face demon has to be optimized to, say, three hundred faces in order to run in the game, the learning curve that the latter modeler has to overcome will take several extra weeks at a minimum.

Any aspect that's overlooked, whether out of ignorance or inexperience, will cause a drift. This is the reason for proper planning and continued monitoring of the project.

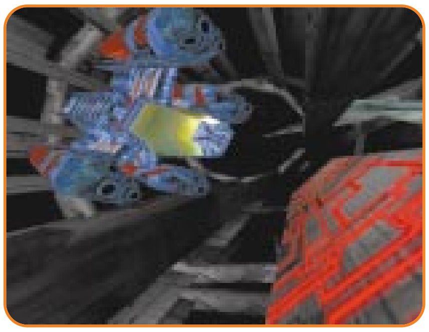
2. MORE INPUT EARLIER IN THE DEVELOPMENT OF THE GAME. I'm not just talking about compatibility testing and input from the publisher. I wish we had had more comprehensive testing and feedback from game players throughout the entire development of the game. I think many of the best ideas for a game, indeed what makes a game a success, are the users' extensions of that game; MODs, add-on levels, and so on. We should have planned the level design contest earlier, anticipated add-on levels earlier in the coding and started work on the level editor and its documentation much sooner.

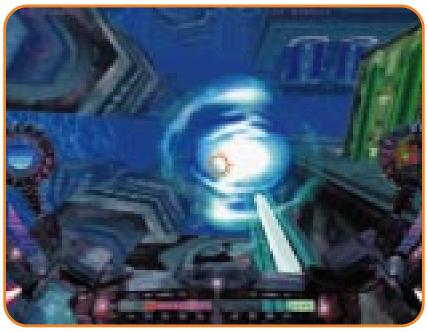
I believe this would have also helped us more accurately anticipate what the market would want and deliver it, or at least be prepared to explain why weren't able to deliver it.

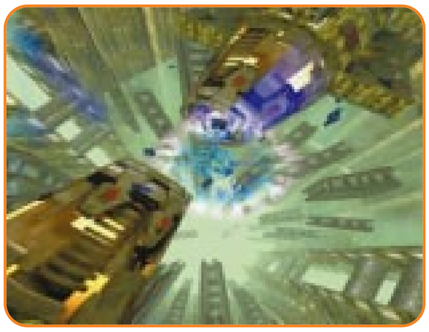
3. A BETTER LEVEL EDITOR. We should have spent a lot more time designing and developing the level editor up front. I think we all felt pressured and excited to dive into the game and simply forged ahead. I'm sure the editor was low on the programmers' priority list at the beginning of development as they were faced with a mountain of problems to solve for the game itself: physics, AI, networking, learning new technologies, and playing DIABLO until sunrise.

Since the level editor was never intended for use beyond the completion of the game, this list of priorities wasn't something that felt wrong at the time. But we should have anticipated this discrepancy. A better level editor would have made it easier on the artist and easier for others to work on level-design–related tasks. The level editor was difficult and ended up being reworked and overhauled for the level design contest. Had we made the level editor easier to use up front, I'm sure the game would have had more and better levels.

4. IMPROPER CODE VERIFICATION, PROGRAM DOCUMENTATION, AND ARCHIVED ASSETS.







Never put all your eggs in one basket, always verify code independently, keep redundant back-ups, and document everything. At least one other third-party programmer should be familiar with your code for numerous reasons, and all creative assets should be archived — and not just the final versions of the files. A flattened Photoshop image is impossible to manipulate, undocumented code is hard to work with, and applications with no file sizes, standards, or parameters are almost useless. When any member of the team moves on or dies, their work will be hard to alter or modify if disorganized, undocumented, and improperly backed up.

During DEAD RECKONING's development, I laid out the schedule for backing up and archiving assets, but never enforced it. When the time came to retrieve and utilize these assets, they weren't easy for me to use. In the end, I had to spend more time digging through files with which I should have already been familiar.

For me personally, the most frustrating instance of this was having to go back and work with flattened, low-resolution images. I waited until too late to try to get these files, as I assumed they were being saved. During development, the artist created Photoshop files of the characters in the game; some of these images were as large as 25MB. As he needed more hard-drive space, he deleted these large images. It was months before the publisher began requesting these images for the ads, boxes, and other layouts. Needless to say, I felt foolish for not having them available.

The level editor was also totally undocumented. When we decided to host the level design contest, we had to document the level editor — thoroughly. With all that was happening in the last few months of development, this was a great burden.

5. MISUSED "PROFESSIONALS." Technically, this "wrong" happened during the development of CYLINDRIX, but the effect was so profound that it had a direct impact on DEAD RECKONING. Also, during the development of DEAD RECKONING, I was able to fix the mistakes made during CYLINDRIX (not that I didn't make all new mistakes during DEAD). But because couldn't fix this problem, I have to mention it here.

As developers, many of us may feel we lack the acumen needed to deal with publishers, employees, and other business situations, apart from the technical and creative. Many years ago, I attempted to balance this (perceived) weakness of mine and hired a consultant. There were bad decisions set in motion during the very beginning of CYLINDRIX that affected the development of DEAD RECKONING many years later. This was due mainly to the restricted cash flow caused by the debt incurred during the development of CYLINDRIX.

In hindsight, I realize that all I lacked at the time was the self-confidence to handle these decisions and situations that initially frightened me. When all this well-meaning advice was being enacted, I was deeply conflicted. My instincts screamed that things weren't right, but I didn't follow my conscience. Of course, when all the bad effects of those decisions started surfacing towards the end of development, I was forced to handle ten times what I would have had before hiring the professional.

The problem is that most businessmen don't understand the computer games marketplace, and fewer understand the game developer. Having a "suit" show up in my office and do the back-slapping, hand-shaking, and other customary behaviors that seem so insincere (to most people) served only to make the team very uncomfortable. I found out later it made them all downright suspicious and I don't blame them.

I was also advised to maintain a "proper" office with a receptionist and several management functions. They were expensive and unneeded in a game development environment, and overkill that only served to sap development resources. What is defined as a proper office in one profession is all wrong in another.

The end of my consultant phase was when I was pushed to self-publish CYLINDRIX. This desire was based on the greed factor and not the facts. To make a long story short, six figures of advertising alone will not get your game anywhere near a retail shelf. You need a publisher.

But when I did decide to get a publisher, we overestimated drastically the title's worth. Because the price was so

far off base, we obviously didn't get a deal. Finally, I went solo and turned my efforts to trying to find a publisher for CYLINDRIX at reasonable terms and cut my losses. By this time, CYLINDRIX was very dated and all the ads that had run turned off potential publishers. They wanted to launch it themselves and didn't want to try to compete with all the marketing and publicity already generated. The publisher I finally went with turned around and sublicensed the game into obscurity.

But CYLINDRIX was not a failure, because it didn't end there. We took all that we learned and used it to create DEAD RECKONING. The fact that CYLINDRIX was a great game and was well-known for that period of time also allowed us to get the publishing deal with Piranha.

In the End...

During the development of DEAD RECKONING, I've had some lofty highs and dismal lows, and the design document kept me focused. Drift got bad for awhile; then I dusted off the design document and got back on track. And despite the effectiveness of the design document, there are a thousand things I'm doing differently this time around. It is most exhilarating to see a vision fulfilled and know the next one will be even grander. ■

Back

by Terry Bratcher

The Game Developer's Resource Center

I have been tortured for the last seven years by a harebrained idea that just won't go away. The gaming community needs a Developer's Resource Center — a permanent location where new technologies can be explored and developers can access all kinds of services.

Illustration by Pamela Hobbs

It should be open to everyone, administered by a neutral party, such as the Computer Game Developer's Association (CGDA), and be supported by the community as a whole.

About nine years ago, I got involved as a playtester at a then-small developer startup, Lucasfilm Games. We had what every developer dreams of: a famous benefactor with unimaginable financial resources, two of the best licensing properties in history, and an interest in making the absolute best games. We had five testers with good access to the appropriate equipment needed for good compatibility.

Hardware companies started contacting us. It began with Creative Labs. They were trying to establish a new standard for audio called Soundblaster. They sent their reps in to teach our programmers how to use their API, and gave us hardware and as much technical support as they could. Soundblaster was an improvement over existing technology, moderately easy to use, and made our products better, so we used it. The same thing happened for any number of products and initiatives over the next two years. Technology changes started coming at a faster rate and the test group grew in response. By 1991, the test group had grown to 35, and a full-blown compatibility lab was in the works. We were Lucasfilm and the world of new technology beat a path to our door. But this wasn't the case for everyone.

I had been to a couple of Computer Game Developer's Conferences by this time, and had seen what a struggle it was for smaller developers to keep up with the latest technologies and APIs. This seemed to be diametrically opposed to the theory that to grow an industry or promote a better technology, input from as many sources as possible was best. The word would spread faster if there was a centralized point of contact for people interested in using a given technology. The game industry would grow faster if some of the advantages that Lucasfilm had were made available to everyone. The rich

Continued on page 71.

In the last six years, Terry Bratcher has directed technical outreach groups that helped bring 3D graphics, MPEG and DVD video, positional audio, and online technologies to the computer game industry. He is currently Director of developer services at Business Development International, and is serving on the Board of Directors of the CGDA. Your comments are welcome at terry46@bigplanet.com.

Continued from page 72.

networking environment that the CGDC offered only came once a year. That wasn't enough. I became convinced that for the benefit of the game community as a whole, we needed a Developer's Resource Center.

The heart of the Developer's Resource Center is a Hardware Compatibility Lab. It is also the heart of the funding plan. Unlike small developers, hardware companies have money to spend. More than one-hundred fifty hardware companies, peripheral manufacturers, and tool developers spend between $250,000 and $3,000,000 each, annually, in an effort to get their products in front of game developers. In doing so, they manage to effectively supply less than half of all developers. The test labs at Electronic Arts, Activision, Lucas, GT Interactive, Interplay, Hasbro, and Cendant, among others, are inundated with every new technology. So much so, that much of this new technology never finds its way into use unless someone from OEM sales proposes a bundle deal. The small-to-midsize shop has one-tenth the attention and no way to afford enough different hardware to do a proper compatibility test.

In the last seven years, the business environment has changed as radically as the technologies. Developers have fallen victim to the instability of the fiscal environment as well as the lack of a stable development platform. Consoles, sub-$1,000 PCs, location-based entertainment projects, the Internet, and set-top boxes using a variety of new technologies, not to mention new and in some cases proprietary operating systems, promise to expand the market for our products. We are experiencing an unprecedented consolidation on the publishing side. There are hundreds of developers out there with great games that never see the light of day because they lack access to an artist, a musician, programming support, an experienced game designer, a test group, or in some cases, a publisher.

Community support has to be key for the Center, for purposes of funding, compatability, and developer outreach. The CGDA is the non-profit, neutral entity that is necessary to the success of the endeavor, to bring all the widely dispersed elements together. No other organization has done more for the benefit of the game community. The CGDA's Board of Directors sees the creation of a center of this kind as a significant benefit to all its members. A centralized database of available talent, equipment, and actual workspace provided at discounted rates to CGDA Members could stimulate the production of new games. Freelancers could support themselves more easily on a project basis — an incubator, if you will, run by our own community.

The heart of the Developer's Resource Center is a Hardware Compatibility Lab. It is also the heart of the funding plan.

Until recently, there have been a scant few places where people interested in learning about our industry could take classes. Schools such as Full Sail in Florida have begun to offer classes in all aspects of game design, production, marketing, and so on. The Developer's Resource Center could provide experienced teachers with a location for their classes as well. As for the location itself, traditionally the San Francisco Bay Area has been the focus of both hardware and software development in the game industry. This makes it a natural location choice. It has been suggested that a local university might be willing to host the Center. I might add that if successful, plans are in place to locate two smaller centers in the U.S. within three years, and two more internationally within five. In the meantime, there are plans to create a Developer's Roadshow to help bring the benefits of The Center to the community in general. The Center can provide outsiders quick interaction with top developers. Our industry needs to develop a positive business model; stimulate the development of stable, usable technology; and grow quickly to meet the demands of the future.

Over the last seven years, I have spoken about and gotten ideas from a number of people regarding the Developer's Resource Center. I've run ISV programs for more than five years. I would *instantly* give one tenth of my budget to have a place where my hardware was properly displayed — where I could, by appointment, help developers test their own products for compatibility and answer programming questions face-to-face, where I could have space for Developer's Conferences or brainstorming sessions with anyone interested in advancing technology. Would I donate hardware? You bet your life! And I'm willing to bet that other people in the game community feel the same way that I do. In fact, I know they do.

The Center will be a unique entity, and I've seen support from businesses and organizations across the industry. The Developer's Resource Center will not pose a competitive threat to any lab, business, or event currently in existence. Hardware labs, for example, are willing to test for you, for a fee. Several have volunteered to act as extensions of the Center in exchange for access to more cutting-edge hardware. I have spoken to members of both Microsoft and Intel's ISV teams, and they support the idea as well. Intel currently sponsors such a center for the Film Industry.

The Developer's Resource Center is also no competition for Plugfest or Meltdown events. (The Center could act as a permanent home for both events.) These events happen on given dates, and are restricted to those who can afford the time and money to attend then. The Center is intended for those whose schedules or budgets don't correspond with a given event. Furthermore, the Center takes nothing away from the number one event for everyone in the development community, the Game Developer's Conference. The Conference lasts just four or five days, whereas the Center is a year round home for anything that anyone wants to expose to the community, that when utilized, gives back to the community immediately. The Game Developer's Conference has expressed interest in helping with the project. Several publishers have asked about Sponsorship.

The idea is just to do a better job, more efficiently, for everyone. The CGDA is currently forming an advisory board to help with planning for the Center. Volunteers are needed. The final step is to open the idea up to the readership of this publication for criticism, and hopefully, support. If you are interested in contributing, please contact me at terry46@bigplanet.com. ■