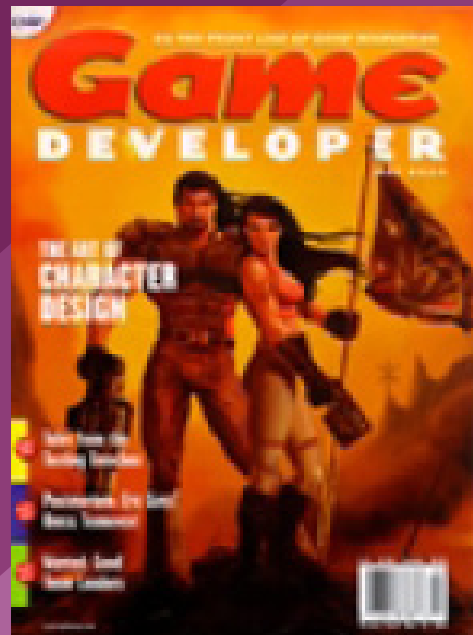




GAME DEVELOPER MAGAZINE

MAY 2000



Amazon.com: Part of the Problem

Walt Disney once said, "People spend money when and where they feel good," and even in these days of shopping from web sites, I think his observation still holds true. For me, Amazon.com was such a place for a number of years. For a long time I enjoyed the site and spent quite a bit of money on books and movies there. But recently that changed. I joined the Amazon boycott.

Back in 1997, Amazon decided to file a patent on "one-click ordering." This is a system whereby you select an item to purchase, and using a unique identifier on your computer (like a cookie), the server takes you to a checkout screen containing your previously-entered identification, greatly simplifying the transaction for the customer. Amazon received the patent for this system last fall (U.S. Patent Number 5,960,411).

The fact that this patent was awarded speaks little (or perhaps volumes) about the patent office. It's my opinion that this government agency, like many others, is underfunded, understaffed, and will probably continue to blunder along as technology zooms ahead. So don't be surprised if we continue to see other idiotic patents awarded in the future.

That's the government's excuse. What about Amazon? On one hand, I can see a company wanting to stake a claim on a process as a defensive move. After all, if someone's going to snag a patent that affects your business, and philosophically you detest software patents, common sense dictates that it's better you than your competitor. (As someone once told me, "If you're not part of the problem, you're getting screwed.")

But Amazon demonstrated that it wanted the patent for more than just an insurance policy. Within a week of being awarded the "one-click" patent, Amazon promptly sued its arch rival, BarnesandNoble.com, for patent infringement. Right out of the chute, Amazon used the patent as an offensive weapon. More recently in February, Amazon locked up a patent on Internet

affiliate programs in which associate web sites are given a commission for each referral to Amazon. It's not clear how aggressively the company plans to enforce that patent, but given its track record, I'm not optimistic.

If you're wondering what Amazon's patent has to do with game development, look around you. Any patent that puts a stranglehold on Internet commerce affects the games business. Selling games direct over the web is big business for game publishers, and if this sort of ridiculous patent is any indication of e-commerce's future, we're all in for a rough ride. This patent stampede has plenty of potential to trample us all, and whether or not you believe that Amazon will bully your company tomorrow, imagine if the next e-commerce patent — let alone one governing a well-established *in-game* process — is snagged by your direct competitor.

It's possible, even probable, that many Internet patents being awarded today won't hold up in court. But since the high cost of defending a patent case naturally favors large companies who can afford protracted court battles, it's the little companies that are most threatened by the rush to patent mundane computational processes. Burgeoning game development companies should spend their money developing games, not on lawsuits brought on by deep-pocket corporations.

I urge you to raise your voice against Amazon and other companies who use our overwhelmed patent office to claim ownership of well-understood and widely used techniques. Tim O'Reilly (president and CEO of computer book publisher O'Reilly & Associates) posted an excellent open letter to Amazon CEO Jeff Bezos, which you can read and add your name to at http://perl.oreilly.com/cgi-bin/amazon_patent.comments.pl. I urge you to lodge your protest against Amazon, and also to hit the company where it really hurts: in the wallet. Buy merchandise elsewhere, and don't sell your games through the site. ■



Publisher
Jennifer Pahlka jen@mfgame.com

EDITORIAL

Editorial Director
Alex Dunne adunne@sirius.com
Managing Editor
Kimberley Van Hooser kvanhoos@sirius.com
Departments Editor
Jennifer Olsen jolsen@sirius.com
News & Reviews Editor
Daniel Huebner dan@mfgame.com

Art Director
Laura Pool lpool@mfi.com

Editor-At-Large
Chris Hecker checker@d6.com

Contributing Editors
Jeff Lander jeffl@darwin3d.com
Mel Guymon mel@infinexus.com

Advisory Board
Hal Barwood LucasArts
Noah Falstein The Inspiracy
Brian Hook Verant Interactive
Susan Lee-Merrow Lucas Learning
Mark Miller Group Process Consulting
Paul Steed id Software
Dan Teven Teven Consulting
Rob Wyatt Microsoft

ADVERTISING SALES

National Sales Manager
Jennifer Orvik e:jorvik@mfi.com t: 415.905.2156
Account Executive, Silicon Valley
Mike Colligan e:mike@mfgame.com t: 415.356.3486
Account Executive, Northern California
Dan Nopar e:nopar@mfgame.com t: 415.356.3406
Account Executive, Western Region and Asia
Darrielle Saddle e:dsaddle@mfi.com t: 415.905.2182
Account Executive, Eastern Region and Europe
Afton Thatcher e:athatcher@mfi.com t: 415.905.2323
Sales Associate/Recruitment
Morgan Browning e:mbrowning@mfi.com t: 415.905.2788


ADVERTISING PRODUCTION

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

MARKETING

Marketing Manager Susan McDonald
Marketing Coordinator Scott Lyon

CIRCULATION


Game Developer magazine is BPA approved
Vice President/Circulation Jerry M. Okabe
Assistant Circulation Director Kathy Henry
Circulation Manager Stephanie Blake
Circulation Assistant Kausha Jackson-Crain
Newsstand Analyst Pam Santoro

INTERNATIONAL LICENSING INFORMATION

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

CORPORATE

President & CEO Gary Marshall
Corp. President, Business Tech & Channel John Russell
President, Business Technology Group Adam Marder
President, Specialized Technology Group Regina Ridley
President, Channel Group Pam Watkins
President, Electronics Group Steve Weitzner
General Counsel Sandra L. Grayson
Vice President, Creative Technologies Johanna Kleppe
General Manager, CMP Game Media Group Greg Kerwin

BIT

Blasts

News from the World of Game Development



New Products: Alias|Wavefront introduces Maya 3, Steinberg releases Nuendo, and Animal Logic offers Renderman output to Max users. **p. 7**

Industry Watch: Sony and Nintendo try to out-convenience each other in Japan, Connectix bests Sony in court, and Bonnell takes over at GT. **p. 8**

Product Review: David Stripinis looks at two facial animation tools, Famous Technologies' Famousfaces and Lipsinc's Echo. **p. 10**



New Products

by Daniel Huebner

Maya 3 Announced

ALIAS|WAVEFRONT has announced the fifth major release of its Maya 3D animation and visual effects software. Maya 3 includes a new feature called Trax, designed to benefit game artists who need to edit large amounts of motion capture data or mix together multiple animation sequences of the same character in a nondestructive, hierarchical, and time-independent manner. Trax allows developers to manipulate and blend animation in a way that can be reproduced in final game play, allowing subtle changes to motion capture data while leaving the original data intact.

In addition to Trax, other new features in Maya 3 include the full integration of Maya's subdivision surface tools into the animation and rendering pipeline. Maya 3 also introduces a completely new polygonal architecture that is more efficient in terms of speed

and memory, and capable of supporting a much wider variety of polygonal topologies. A new Bézier surface editor provides full access to curved surface representation.

Maya Complete 3 will be available this summer for IRIX and Windows NT. Prices are expected to start around \$7,500.

■ **Alias|Wavefront**
Toronto, Ontario, Canada
(416) 362-9181
<http://www.aliaswavefront.com>

Nuendo for Postproduction

STEINBERG released its Nuendo Media Production System, a modular system centered on an audio software application that integrates seamlessly with several software and hardware accessories. Nuendo is a 128-track audio recording facility and a complete 128-channel audio mixer, which supports surround sound in impressively flexible ways. The system can be configured for any surround format, including 5.1 and 7.1, and users can create their own setup or use a standard configuration from Nuendo's library. Nuendo features

up to 200 tracks of 24-bit, 96KHz digital audio, advanced surround mixing, a video track, and MIDI tracks, along with comprehensive functions for digital audio. Nuendo uses exclusively native signal processing with no DSP chips needed. Every

function is run from the computer's host processor.

Nuendo is available for Windows 98 and NT, and Steinberg also plans to release a version for BeOS in the near future. Pricing for Nuendo starts at \$1,295.

■ **Steinberg North America**
Chatsworth, Calif.
(818) 678-5100
<http://www.us.steinberg.net>

Animal Logic's Maxman Plug-in

ANIMAL LOGIC has created Maxman, a plug-in that allows 3D Studio Max users to output their content directly to Pixar's Renderman package as a rendering alternative.

Maxman produces a Renderman .RIB file from a 3D Studio Max scene while preserving all geometry, animation, and texture data and allowing the assignment of Renderman shaders. The Maxman suite of fully integrated and interactive plug-ins includes a top-level interface for 3D Studio Max, a set of modifiers, and materials and maps with which Renderman users can manipulate the finer details of the .RIB creation sent to Renderman. Maxman integrates into 3D Studio Max using standard scene controls. Renderman shaders are supported in all contexts, including global atmosphere, light, displacement, volume, and surface, with full integration of their parameters into the Max animation system.

Maxman can be used with Pixar's Renderman Toolkit, BMRT, or other Renderman-compliant renderers, and support for RenderDotC is forthcoming. User licenses for the full version start at \$2,000 for Windows NT 4 and 3D Studio Max 3.1.

■ **Animal Logic**
Sydney, Australia
+61 (2) 9383-4800
<http://www.animallogic.com>



The Nuendo Media Production System harnesses the power of your host processor, eliminating the need for DSP chips.



Industry Watch

by Daniel Huebner

FINAL FANTASY VIII AND SOME BEEF JERKY, PLEASE. Both Sony and Nintendo have made moves to place their products in Japan's ubiquitous corner stores. Playstation.com, Sony's online hardware and software sales site, counts 7-Eleven of Japan among its group of 11 investing companies, giving Sony sales and distribution access to 7-Eleven's 8,000 retail locations across Japan. Nintendo matched the move by purchasing a three-percent stake in Japanese convenience retailer Lawson, granting Nintendo access to 7,000 Lawson outlets. Both arrangements will include on-site sales of hardware and software and the option for Japanese customers to pay for and take delivery of products ordered online.

GRAPHICS WARS RAGE ON. Leading contenders in the hardware acceleration wars ended the fiscal millennium on decidedly different notes. Nvidia's fourth quarter revenues increased 96 percent to \$128.5 million from fourth-quarter totals a year earlier. Net income for the quarter doubled the previous year's result, increasing to a company record of \$14.6 million. For the entire fiscal year, revenues rose 137 percent to \$374.5 million, while net income for the year increased 822 percent to \$38.1 million. The numbers posted by 3dfx were less rosy. Though 3dfx's fourth-quarter revenues rose this year from \$60.7 million in the fourth quarter last year to \$109.4 million this year, the company recorded a quarterly loss of \$31.9 million as opposed to a profit of \$2.1 million in the same period a year ago. Revenues for the entire year of 1999 went up to \$360.5 million compared with 1998's total of \$202.6 million. Losses, however, increased to an astounding \$63.3 million compared with earnings of \$21.7 million in 1998.

RECENT ACQUISITIONS. ATI Technologies has entered into a definitive deal to acquire graphics chipmaker ArtX for approximately \$400 million in ATI common shares and options. The acquisition helps move ATI into new consumer markets; in addition to



ATI's Ho (left) and ArtX's Orton seal the deal.

developing graphics chips for DVD players and Internet terminals, ArtX is the provider of graphics technology for Nintendo's upcoming Dolphin console. ArtX founder Wei Yen has joined the ATI board of directors, current ArtX president Dave Orton took the newly-created title of president and COO of ATI Technologies, while ATI's K.Y. Ho will continue as CEO and chairman.

Electronic Arts went on a buying spree of its own, buying game maker Dreamworks Interactive. Under the terms of the agreement, Dreamworks Interactive will become a wholly-owned subsidiary of Electronic Arts while continuing to produce titles related to Dreamworks SKG properties. The company, originally founded as joint venture between Dreamworks SKG and Microsoft, has been publishing with EA for the past two years. Terms of the sale were not disclosed.

UNDER NEW MANAGEMENT. Infogrames chairman and CEO Bruno Bonnell has taken up the reins at GT Interactive as chairman and CEO. Bonnell is filling the positions left vacant by the departures of GT CEO Thomas Heymann and COO John Baker. Los Angeles-based Heymann and Baker both decided to leave in part because of the company's decision to remain headquartered in New York.

The moves follow Infogrames' acquisition of a 70 percent stake in GT, and could signal the start of Bonnell's efforts to merge the two companies. GT Interactive's third-quarter results were short on bright spots as the company announced net losses of \$118 million for the quarter, while revenues slipped to just \$102 million. Though GT blames the quarter's slump on a light release

schedule, as much as \$89 million of the quarter's loss can be attributed to charges incurred from restructuring and reorganization related to the acquisition. GT's reorganization includes an end to its European operation, folding its budget publishing business, and the closure of TOTAL ANNIHILATION developer Cavedog Entertainment.

CONNECTIX WINS APPEAL. Playstation emulator maker Connectix has won the reversal and remand of a court injunction banning the sale of its Virtual Game Station for Macintosh. Sony had contended that the Virtual Game Station infringed on Sony copyrights, especially in relation to the Playstation BIOS (basic input/output system), and tarnished the Playstation reputation. Connectix, backed by software makers and trade associations, argued that they had the right to replicate non-copyrightable functional elements including a reverse-engineered BIOS. The court agreed, and recognized Connectix as a legitimate and innovative competitor. Connectix plans to begin shipping the Macintosh Virtual Game Station immediately and a Windows version soon after. ■

UPCOMING EVENTS CALENDAR

PC Data Trends 2000

THE W HOTEL

San Francisco, Calif.

June 8-9, 2000

Cost: early-bird rates available

<http://www.pcddata.com/conferences>

MedPi Software 2000

CONGRESS CENTER AND AUDITORIUM MONACO

Monte Carlo, Monaco

June 28-30, 2000

Cost: variable

<http://www.unmf.fr/anglais/home-a.html>



Facial Animation with Echo and Famousfaces

by David Stripinis

Until recently, facial animation wasn't an issue in game production outside of prerendered cutscenes. Memory limitations prevented large-scale facial animation, while polygon budgets often kept characters from even having lips, never mind lip-synched dialogue. The advent of next-generation platforms like Sony's PlayStation 2 and Nintendo's Dolphin promise capabilities that will make these limitations nearly immaterial. The downside of this is that as producers and designers begin to promise publishers hours of detailed facial animation, the animators sit in abject horror. While the thought of doing hours of facial animation may appeal to some, the sheer scope of such a task sends many animators running and screaming to the hills. As technical limitations evaporate, ambitions grow and production schedules shrink. Animators looking for viable solutions for streamlining this oft tedious process would do well to examine two pieces of software: Echo from Lipsinc and Famous Technologies' Famousfaces.

ECHO works via voice analysis. The user provides audio files of dialogue, and Echo breaks the voice track into keyframe data and writes that data into a simple text file. This data can then be read directly by a game engine to provide on-the-fly animation data, or animators can use it to hand-animate characters.

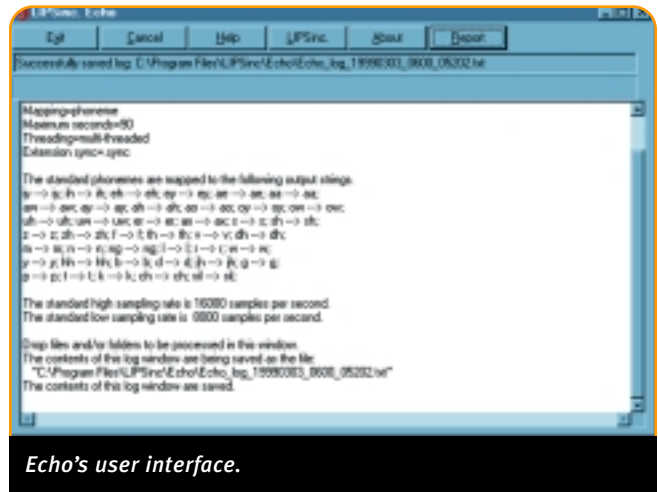
INTERFACE. On first starting the program, you are presented with a very simple interface. A text window gives a status report of all options currently set, but not much else. The only way to change program options is by editing the .INI file directly. While this may provide some with a sense of security that users will not mess up the

data coming out of Echo, I found it quite cumbersome. A simple Preferences menu would have been welcome. Once the options are set, however, using the program could not be easier. The user simply has to drag a selection of audio files onto the Echo window, and the analysis is done in a batch process.

WORKFLOW. Output is provided as a text file with a .SYNC extension. There are three flavors of output: flipbook, dopesheet, and viseme curve. With both the flipbook and dopesheet options you can specify either viseme or phoneme output. The viseme curve output is limited only to visemes. The program supports 16 visemes and 41 phonemes, the choice of which is controlled in the .INI file. This may seem confusing to animators familiar with the "classic" mouth shapes used for lip-synching, which are commonly referred to as phonemes; in Echo, "viseme" is the comparable term. The 16 visemes are, by default, referred to by a rather confusing set of words that approximate the mouth shape of the viseme. The mapping can be changed in the .INI file, including mapping multiple shapes to a single shape. The larger set of phonemes provides a much more detailed analysis report, though that amount of detail is largely unnecessary for most applications.

The simplest of the three output options is the flipbook. It outputs a time index and a viseme name. The time, in milliseconds, indicates at what point from the beginning of the sound file a phoneme or viseme begins. It is the least complicated of all the output options, and is possibly the format most suited for direct use in real-time applications.

For animators, a more familiar output option is the dopesheet. This for-



mat is very similar to a traditional animator's exposure sheet, listing frames and mouth shapes. If you have no way of automatically inputting the Echo output files into your software, the dopesheet format will most likely fill your needs quite well.

The most robust output option of the three is the viseme curve, which lists every viseme, including detailed information for each key. It lists frame time, keyframe value in a percentage, and spline values in and out of that key. For ambitious animation systems, this data can be read directly into the real-time engine. This is also the most useful for reading into 3D animation software.

If none of these options works for your application, Lipsinc does offer customization options. You can contact them for more details regarding your circumstances.

Overall, I found the output from Echo to be very accurate, especially when using the optional text companion files to remove all ambiguity as to what is being said. The analysis can be too accurate, in fact, providing too much detail and resulting in an overly busy animation. Echo also does not take into consideration any of the emotive qualities of the vocal performance, and cannot handle full facial animation, only lip-synching.

FAMOUS **FAMOUSFACES**, from Famous Technologies, offers a completely different approach. It is a stand-alone Windows NT application dedicated to facial animation and plug-ins for a variety of popular animation packages are also available.

David Stripinis is director of animation at Factor 5. When not rambling on about the sociopolitical ramifications of next-generation platforms, he can be found at his computer, desperately trying to finish his short film India. Contact him at david@factor5.com.





Setting up clusters with Famousfaces.

Famousfaces works by facial capture, a process very similar to motion capture. A performer's face is rigged with a number of reflective markers and videotaped. This data is then read in and tracked, producing the animation data while the voice recording is being done. Famousfaces supports both full 3D point tracking and 2D image tracking in the FamousTracker software that is included.

INTERFACE. I was very pleasantly surprised by the Famousfaces interface when I first opened it up. It was very clean and would be familiar to any animator working with the current generation of 3D software. A main window holds the 3D viewport, the menu and command panels sit along the top and right side, and a time slider rests along the bottom. Navigation in the 3D viewport was very intuitive to me as a Maya user, using a combination of mouse clicks to pan, zoom, and rotate. Actually, it may have been a little too familiar, as I found myself getting frustrated trying to use the Maya commands in Famousfaces. Too much of a good thing, I suppose.

WORKFLOW. Famousfaces works by importing a model via a variety of object formats and applying clusters to the face. Clusters are collections of vertices controlled by a common point, influenced by the controlling cluster points by varying percentages. These clusters are quite intuitive to set up using a painting tool to control the weighting of each cluster point. These points are then driven by animation channels derived from your

captured data, and may be edited and tweaked using a familiar suite of keyframing tools. The software allows you to view the video of the capture session linked to your playback and time-slider scrubbing, giving you the ability to check the integrity and quality of your animation.

One of the best features of Famousfaces is the

Enabler plug-ins, which allow you to integrate Famousfaces into Maya, 3D Studio Max, Softimage, and Lightwave. Famousfaces integrates itself into each program's particular workflow, for example running as a Maya Embedded Language (MEL) script in Maya while appearing as a modifier in 3D Studio Max, allowing artists to go back and forth relatively seamlessly between Famousfaces and their software of choice.

The quality of the animation produced by Famousfaces, even using a simple 2D track of a video, is quite stunning. Even more impressive is the full 3D capture that can be achieved via multi-camera setups. I was really amazed by the subtlety of motion achieved in such a short period of

time. I managed to set up the clusters, track the video, and apply the motion in a just few hours. After I familiarized myself more with the product, the process was even faster.

The quality of data may be the major shortcoming of Famousfaces. It works by pure vertex deformation, not by using weighted morph targets. This means that in real-time applications a large amount of data must be streamed into the animation engine, which may cause some problems.

Overall, I found Famousfaces to be quite viable in a production environment. While it comes with all the baggage and hassles generally associated with motion capture, such as data management, animation data density, and the need for input hardware, the results are worth the hassle. Being able to create animation of this quality and detail with such speed and ease should be a welcome blessing to animators everywhere.

GETTING A HANDLE ON YOUR WORKLOAD.

Facial animation in games is here to stay and it will only get more complex as time, technology, and expectations move inexorably forward. Animators must find viable solutions to streamline the process if they are to keep on schedule. Lipsinc and Famous Technologies provide tools that help, though neither product has yet achieved perfection. In the end, I found both still required some massaging of data and an animator's skilled hand, but they were extremely helpful to the animation process. ■

Echo: ★★★★★

Lipsinc
Cary, N.C.
(919) 468-7005
<http://www.lipsinc.com>

Price: Licenses start at \$10,000.

System Requirements: Windows 98/NT 4, Pentium II, and 64MB RAM.

- Pros:**
1. Variety of output formats and customization.
 2. Simple to use, especially while batch processing.
 3. Very accurate analysis.

- Cons:**
1. Only derives speech; emotion is left to the animator.
 2. Configuration options not easily accessible.
 3. Output can be too busy.

Famousfaces: ★★★★★

Famous Technologies
San Francisco, Calif.
(415) 835-9445
<http://www.famous-tech.com>

Price: \$4,990

System Requirements: Suggested requirements include Windows NT, 330MHz Pentium III, 256MB RAM, and OpenGL acceleration.

- Pros:**
1. Highly detailed full facial animation.
 2. Intuitive interface.
 3. Easy integration with other 3D software.

- Cons:**
1. Cost of equipment for capture.
 2. Density of data.
 3. Facial animation can't be done until final dialogue is recorded.

To Deceive is To Enchant: Programmable Animation

When we create a game, we create an illusionary world for the player. Hopefully this world is so rich and alive that the player may actually begin to believe in the existence of this game world. As game makers, we are in the business of creating life.

Games are getting pretty good at creating believable and immersive 3D game worlds. Graphics hardware has enabled multi-texture lighting techniques and higher polygon counts, greatly improving the environments where we play. It's not uncommon for a person passing by a computer monitor or television screen to mistake a game environment for a video broadcast or a still photograph of some real place.

There are few people, however, who would mistake a 3D animated character for a real person. In the early days of 3D games, it was sometimes even hard to tell what a character was supposed to be. Now that computing power has enabled us to create more realistic-looking characters, we need to make these creations come alive. It is not enough that the character looks as good as a still rendering. Characters need to have what Frank Thomas and Ollie Johnston of Disney called "the illusion of life."

Creating the Illusion

One of the great challenges that an animator faces is giving a model the appearance of life. Creating a walk cycle for a character where the feet move correctly and do not slide will not make it come to life. A good animation will give the character a sense of weight, purpose, and emotion, attributes which must be individually crafted for each character. This is one of the pitfalls many encounter when using motion capture to generate animation. It would seem that when cap-

turing the performance of a live person, you would get all the subtle nuances that gives the person life. To some extent that's true. However, what you are capturing is the performance of an individual of a particular size and type operating under certain physical limitations. These limitations affect the range of reality the actor can impart on a character.

An interesting example of this was a visual effects company that needed to create a superhero. They hired a very talented stuntman and attached him to various harnesses and had him leap, roll, pose, strut, and fight all over a stage while the performance-capture cameras were rolling. Unfortunately, when the motion was applied to the hero model, instead of looking like a superhero it looked like a guy hooked up to a harness jumping around. This is because the equipment accurately captured the performance

of this person. He was an actor, not a superhero. He didn't move the way we expect a superhero to move, and the company ended up hiring a group of animators to bring this superhero to life. Sometimes even captured reality is not enough to make a work of art come to life.

Winkin', Blinkin', and Nod

One of the things that can kill the illusion of life very quickly is to have a dialogue with a character staring at you continuously without ever blinking. This is very distracting, as we are very obviously accustomed to talking with people who blink occasionally. When this doesn't happen, it's immediately apparent that something is wrong. So, an easy step toward making your characters more realistic is to allow them to blink.

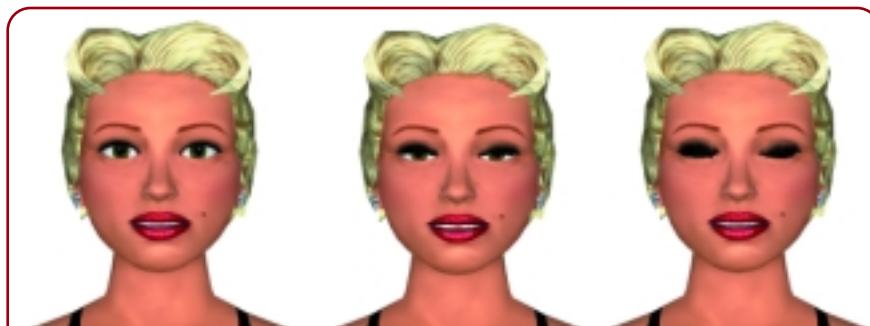


FIGURE 1. Programming a blinking animation can add realism to a character.

Jeff is becoming more and more concerned about what procedural functions are controlling his thoughts. If you are the one actually in control, drop him a line and let him know what is really going on at jeffl@darwin3d.com.



FIGURE 2. People tend to look around their surroundings when idle.

In a low-polygon model, there may not be polygons allocated to the eyes that would make a blink possible. In this case, it's possible to use texture animation to make a blink happen. This has been done in a few 3D games (GRIM FANDANGO, for example). However, if I really want to create expressive characters that can show a range of emotions, I need a model with enough facial polygons to convey the expressions I want. I could hand-manipulate the eyes throughout all of my animations, including an idle cycle. But this would mean the blinks would always happen at the same time in the cycle and it may look canned. Why not animate the eyes automatically through the game engine?

If you read my column on real-time facial animation ("Flex Your Facial Animation Muscles," Graphic Content, July 1999), you may remember that I advocated creating a series of facial morph targets that simulated the actions of the facial muscles. In this system, there is one muscle that controls the closing of each eye. By activating these two muscles, I can make my character blink whenever I want. It may seem overkill to have each eye individually controlled instead of having one action that closes both eyes. However, if you want the character to have the ability to wink, you will need this flexibility. I also don't really care for synchronized blinks. There are arguments over this among animators, but I happen to prefer it if the eyes do not close at the exact same time. I like the eyes to

close just off a half-frame or so. Creating my models with separate muscle targets gives me this flexibility.

Anyway, assume I have a model with individual muscle targets for closing the left and right eye. The muscle settings go from 0 percent (open) to 100 percent (closed). I know that I generally want the blinks to happen every four to ten seconds and each blink should take only two to three frames to complete. This gives me the outline for a procedural blinking model:

1. Pick a random number from 90 to 300 that signifies the number of frames until the blink begins (three to ten seconds at 30FPS).
2. Every frame, count that number down until it hits 0.
3. Pick a random number from 30 to 50 for each eye.
4. Add that number to each eye muscle, limiting to 100 percent.
5. Subtract that number from each eye to get back to 0.
6. Start over at step 1.

You can see a sample blink sequence in Figure 1.

This procedural sequence can run as part of my animation loop without any other controller. As the actual morphing system is part of the standard facial animation system, the blinking generates very little overhead.

This idea of a procedural function for generating motion can go beyond simple blinking. When people are standing

lar random facial effects can easily be crafted to move the eyebrows and mouth slightly, giving the character even more life.

The game AI can also drive the character animation system automatically. For example, the AI system may track many of the character's various attributes such as fatigue level, mood, and physical pain. Facial expressions can be crafted to exhibit these various traits automatically as the AI system changes them. This not only adds realism to the game, it provides a secondary feedback mechanism to the player.

Looking Beyond the Face

While simple facial adjustments can do a great deal to break a character's wooden stare, something needs to be done with the static poses. An idle animation or two can help a great deal. However, these animations are usually cyclical and the pattern is easy for players to detect. Adding some procedural noise to these standard animations can help a lot. If your animation system is composed of a character that is deformed by the use of a skeletal system, these kinds of on-the-fly modifications are easy to implement. For one thing, you could add a little random rotation to the head joint to match the wandering gaze of the procedural eye controller. If there is a bone

A character in a game should not stare rigidly forward waiting for input from the player.

idle, their attention drifts. They look around to get a sense of the surrounding environment. Likewise, a character in a game should not stare rigidly forward waiting for input from the player. If your facial controller allows you to orient the eyes, you can apply the same techniques as the blink function to make the character look around a little. Obviously, if the character is actually looking at an object or a person, this look-at constraint would override the random eye-wandering behavior. Simi-

controlling the finger rotation, the character's grip can be relaxed and tightened, which people often do when they are waiting around for something to happen.

I have often found it useful to create special bones in the character skeleton specifically for these sorts of custom procedural effects. One example is literally breathing life into a game character. In my column on skeletal deformation techniques ("Over My Dead, Polygonal Body," Graphic Content,



October 1999), I discussed the use of a full transformation matrix to deform a 3D character. A side benefit of this technique is the ability to use transformation components beyond simple rotation. A bone can also be translated and scaled to deform the character. Though it may not be obvious when these techniques are useful, this is one of those cases. Suppose I made a child bone of the chest and called it "breastbone." I could then attach the vertices at the front of the chest to this bone. By cyclically scaling this bone up and down, I can give the character the appearance of breathing. Say my character normally breathes 12 times a minute and this goes up to 20 times a minute when the character is very excited or fatigued. I can create a simple procedural formula that will automatically create a breath cycle in the game without using any animation. Something like $\text{breastbone.scale} = (\sin(\text{DEG2RAD}(\text{frame} * 1.2)) / 4.0) + 1.25$ will make the breastbone scale up from 1 to 1.5 once every 150 frames, or 12 times a minute at 30FPS. Increase that 1.2 to 2 and the character would breathe 20 times a minute. You can see how formulas can easily be crafted that would enable all kinds of automated behavior.

Ken Perlin has experimented more than most with using procedural functions and controlled noise to generate animation. The Improv animation system, developed at New York University's Media Research Lab and since licensed to Improv Technologies, uses controlled noise and a variety of animation blending functions to create unique and believable animation. The animation is controlled from a high level using a scripting language. You can see a Java interface to an Improv-driven facial animation performance in Figure 4. Improv Technologies is licensing its animation system for game development applications.

Muscles Flexing

A procedural noise or cyclical animation effect can be very interesting. However, sometimes an effect needs to be the direct result of an action. For example, a strong character may have biceps that bulge as the character bends its elbows. Just as I did with the breathing example above, I can

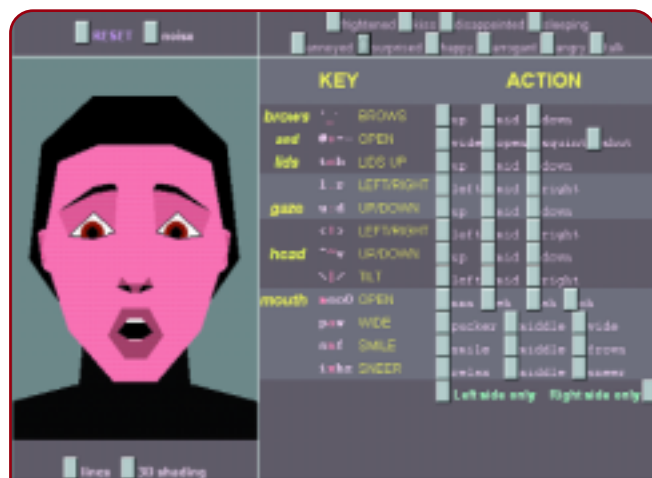


FIGURE 4. Improv Technologies' system uses animation blending functions and controlled noise for unique effects.



FIGURE 3. Trust me, this chest is heaving. Adding a breathing sequence can enhance a character's emotional response.

create a child bone of the upper arm and place it in the middle of where I desire the biceps "muscle" to appear. This logical bone for the biceps is then associated with the vertices along the top of the upper arm, exactly where the biceps would appear. I just need to scale the biceps bone up a bit to make the muscle bulge. In order to detect when I want this to happen, I could watch the rotation of the forearm and adjust the biceps as the forearm rotates. However, I really do not want to create all this code to monitor the bones involved in these effects. I just want it to happen automatically.

In many 3D animation packages, you can create a structure called an "expression." An expression creates a mathematical link between two objects in the animation system. I want to use this same idea to simplify my muscle-bulge situation. When the lower arm rotates around 120 degrees about its local X-axis, I want the biceps to be scaled up to 1.5 of their original size. An expression that performs this task may look like $\text{bicep.scale} = (\text{forearm.rotX} / 240.0) + 1.0$.

As you can see, if the forearm is not rotated, the scale would be one. When the bone rotates, the scale will increase. Extra care needs to be taken to make sure that the scale doesn't go below one or get too great. However, this is easy to accomplish with degree-of-freedom restrictions.

Effects similar to this can be achieved simply by animating the biceps directly along with the forearm. Unfortunately, this will not work if the player can dynamically pose the character through inverse or forward kinematics. The use of expressions also saves animation space by eliminating the need to store the data directly. And while this is just a simple example, you can see that using expressions to generate real-time animation data can be a very powerful tool.

Looking Around the Room

Many algorithmic techniques for animation cannot be triggered simply by a mathematical expression or stochastic procedure. They require some input from the user. A simple example is the look-at constraint. When the player



(or AI system) directs a character to look at a location, this direction takes the form of a request for an animation solution that solves a geometry problem. Given a character who has a head that can look at a limited subspace, we need to find the orientation for that head that points in the direction of a target location.

Recall that a 3x3 orientation matrix is composed of three orthogonal unit vectors that define the local coordinate axes of the rigid body. These vectors are often known as the right (R), up (U), and forward (F) vectors which define the local X-, Y-, and Z-axes in the right body. If I can determine the direction for these three vectors, I can combine them to form the orientation matrix for the head of my character.

$$\text{Orientation} = \begin{bmatrix} R_x & U_x & F_x \\ R_y & U_y & F_y \\ R_z & U_z & F_z \end{bmatrix}$$

When the character looks at a location, the head is aligning one of these three axes with the vector between the root of the head and the look-at target. In my case, I have constructed the head such that it normally looks down the positive Z-axis. So to make the head look at something, I create a vector between the root and the target and normalize it so it becomes a unit vector. This defines the forward vector in the above matrix, giving me one piece of the puzzle. I know that generally I still want the head to be aligned upright along the original Y-axis. So for now, I am going to set the U vector to be (0,1,0). This may not be correct (the look-at may cause the head to tilt a bit), but for now it will be fine.

To determine the R vector, I am going to use the fact that the cross

product of two vectors is perpendicular to them.

$$R = U \times F$$

But I still need to fix up U since my guess may not have been correct. This is easily accomplished by crossing the F vector with the new R to determine the actual U vector.

$$U = F \times R$$

That gives me all the pieces to the orientation matrix and I can make the head look at any

point in my 3D world. However, this could lead to problems if the point is behind the character. The head would spin around like Linda Blair's in *The Exorcist*. In most cases, you will want to have limits on look-at constraints so the characters will only do what is physically possible. For the head, I probably want to restrict the character so it can turn its head only 60 degrees or so in each direction. In order to make this happen using the above method, I would need to take the orientation matrix I calculated and convert it to Euler angles and make sure none of the angles was outside the range of plus or minus 60 degrees. That is kind of a pain and mathematically intensive so let's take a look at the problem in another way.

From a geometric perspective, the problem of looking at an object can be thought of as a two-degrees-of-freedom problem. I want to find the angle around the local Y-axis (or yaw) and the angle around the local X-axis (or pitch) that the head needs to rotate in order to look at the target. I can solve the problem by projecting the target point onto the local XZ plane to determine the yaw, then projecting the target on the local YZ plane to determine the pitch. Each of these steps then becomes very easy. Figure 5 shows the target position projected onto the XZ plane.

I know from trigonometry that the value for $\tan\theta$ is equal to T_x/T_z . So I can determine what yaw the head needs to turn by taking $\text{atan}(T_x/T_z)$. I can do the same for the pitch using $\text{atan}(T_y/T_z)$. If those values are within the range of

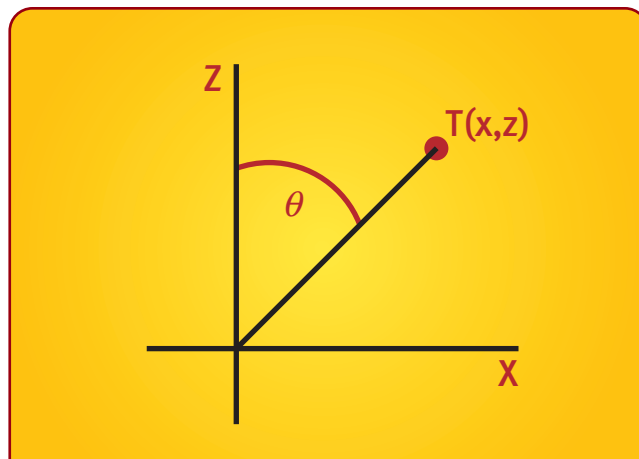


FIGURE 5. Projecting the target position.

motion for the head, the character can animate to that position.

The Rise of the Programmer/Animator

I think we are quite a ways from being able to create complete, believable, and interesting animations programmatically for interactive 3D characters. But an interactive character is not like an actor in a video clip or a 3D cutscene movie. In order for this character to create the illusion of life while the player interacts with it, we need to accentuate the 3D model with programmable animation techniques. Through creative use of methods such as procedural animation and programmable expressions, we can supplement the basic animation with interactive elements that react with the game environment. In this respect, game developers are clearly treading across new ground. These challenges and opportunities are unique to interactive animation. However, this is certainly not the domain of game programmers alone. Creating tools and production procedures that can get artists, with their creative vision for the game characters, involved in the process will be critical if we are to deceive and enchant our audience successfully. ■

Acknowledgements

Thanks to Lisa Washburn at Vector Graphics (<http://www.vectorg.com>) for the model of Vivian used in the article.

FOR FURTHER INFO

Disney Animation

Thomas, Frank, and Ollie Johnston. *The Illusion of Life: Disney Animation*. New York: Hyperion Press, 1995.

Improv Technologies

For information on Improv see Ken Perlin's web site as well as Improv Technologies'.

<http://www.kenperlin.com>
<http://www.improv-tech.com>

Skin Deep 3: Animating and Texturing a Patch Surface

For the past two months, we've been examining some of the latest and greatest surfacing strategies for real-time 3D content. The motivation for this has been the fact that as the rendering and processing power of gaming platforms increases, the amount and complexity of gaming content

will, in general, rise to take advantage of the increased capability. In order to accommodate this increased capability, we have presumed that a shift in production techniques is now or will soon be appropriate, so that we as artists will be able to plan for and take advantage of this increased horsepower.

Of all the surfacing methods we've examined, I identified the B-spline patch surface to be one of the most effective, and last month we took a look under the hood at how to construct and implement a RT3D patch surface model using 3D Studio Max. Since it's unlikely that your job as an artist ends there, we'll round out our discussion of surfacing techniques this month by examining texturing and animation methods for our B-spline patch surfaces.

Task Breakdown: Modeling, Texturing, Animating

In last month's column, I demonstrated the advantages, from a modeling standpoint, of using a B-spline patch. Following the guidelines of our predetermined ideal process (that is, using a control point lattice to separate the modeling process from the complexity of the final result, as shown in Figure 1), we were able to use the surfacing tools in 3D Studio Max to create a resolution-independent RT3D character model. And since the technique we used was totally resolution-independent, we could scale the complexity of the model instantaneously to generate an arbitrary number of levels of detail. The example

character we generated took only a few days to create. By contrast, to arrive at the same result using standard polygonal methods would have required several additional days' worth of modeling time.

Though significant, the time saved in generating the character geometry is only one of the benefits of using a surfacing technique. The task of modeling a character is by far the smallest and

least time-consuming part of the character creation process. The texturing and animation setup for the character can take anywhere from two to five times the amount of time spent in modeling the geometry. And, using standard polygonal methods, the time and effort spent in texturing and animation setup is directly proportional to the complexity of the geometry. In other words, the more complex the

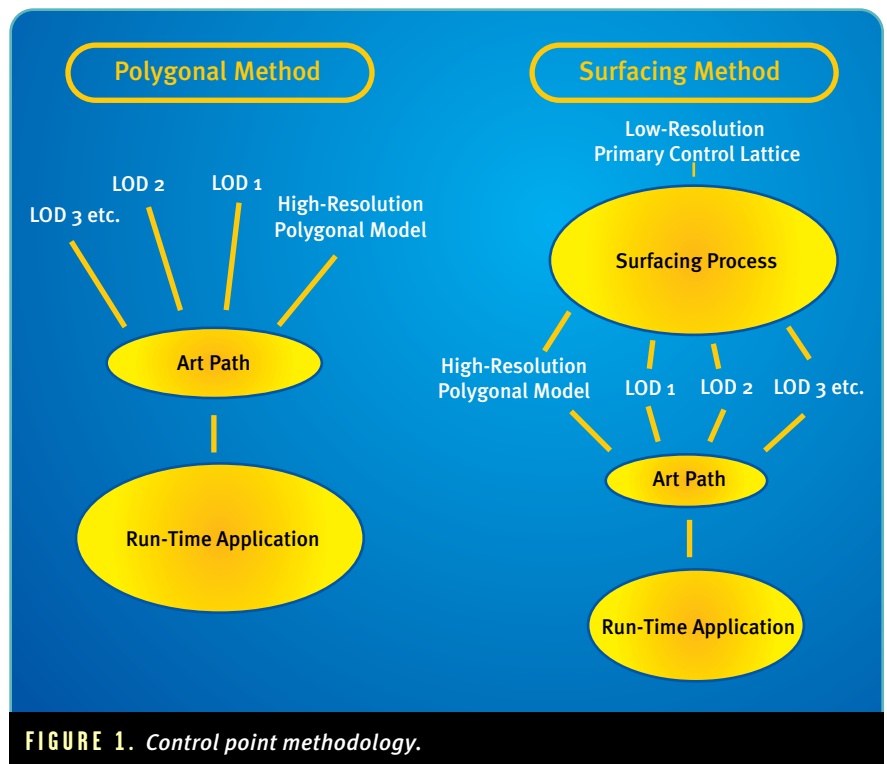


FIGURE 1. Control point methodology.

Mel Guymon has been animating in the gaming industry for several years. When he's not at his desk pushing polygons, he can usually be found at the local Barnes and Noble, slumming for reference materials. Mel can be reached at mel@infinexus.com.

character model, the longer it takes to apply the mapping coordinates and weight the character to its skeletal hierarchy. In order for our chosen surfacing technique to be valid, we need to make sure that the efficiency and cost savings we saw in the modeling tasks are fully realized in the texturing and animation setup for the character as well.

Texture Mapping

Among the various production tools, the processes for applying a texture to 3D geometry are reasonably uniform. In general, a pregenerated texture map is applied first using a global mapping object (planar, cylindrical, spherical, and so on), and subsequent local modifications to vertex UV coordinates are made by manipulating the points on a 2D projection of the geometry, as seen in Figure 2. Although there have been some advances with procedural textures and spatial coordinate field mapping, the

steps and procedures for applying textures follow these basic guidelines. And since the global mapping objects offer only gross control over the UV coordinates of the geometry, much of the time and effort spent texturing the model is used to tweak the local coordinates directly, using one of the various unwrapping tools available. So the more vertices an object contains, the more UV coordinates need to be stored and manipulated, and the longer it will take to apply and adjust the texture mapping. Until the basic techniques for texturing are changed, the time and effort spent texturing will be directly linked to the complexity of the content. By implementing a surfacing technique like our ideal process (Figure 1), we can break this link between complexity and texturing time.

Figure 2 shows the same texture map applied to two similar geometries. At the top is a relatively high-resolution polygonal mesh and its corresponding representation in the local mapping tool (UVW Unwrap). Note

the excessively high number of UV coordinate points, corresponding to the number of vertices in the mesh. It's clear that any local manipulation of the mesh's UV coordinates will be a difficult and tedious process. Compare this with an equivalent surface created with patches, shown at bottom. Regardless of the resolution and number of patch subdivisions, the artist need only work with the control point lattice when it comes to mapping. Not only will the artist spend less time on the mapping process, but the result will be exact, since the UV coordinates are smoothly interpolated along each patch boundary when the model subdivides procedurally. Furthermore, once the control point lattice is textured, any number of LODs can be generated instantly, whereas with the polygonal mesh each LOD must be texture-mapped by hand.

Animation Setup

As is the case with the texturing process, the tools and techniques for setting up a character to be animated using a weighted skeletal hierarchy are fairly consistent between production packages. The process is achieved in two steps: first the animator applies a skinning modifier to associate the geometry globally with a pregenerated skeletal hierarchy (bipedal hierarchies now come ready-made in most packages), and then modifies the weighting values by hand to ensure proper deformation of the animated skin. Similar to the texturing process, the initial global skinning process is seldom accurate enough, and much of the time in animation setup is spent making minute adjustments to the vertex weights to ensure that the skinned geometry deforms properly.

The tools for adjusting the weights are fairly advanced, and include the ability to paint the weights directly on the surface of the geometry or to adjust a "bounding envelope" graphically, which is associated with a specific hierarchical node. However, the number of vertices or control points in the weighted geometry remains the limiting factor in time and effectiveness. This is even more important than in the texturing step: with a polygonal mesh, as the complexity of

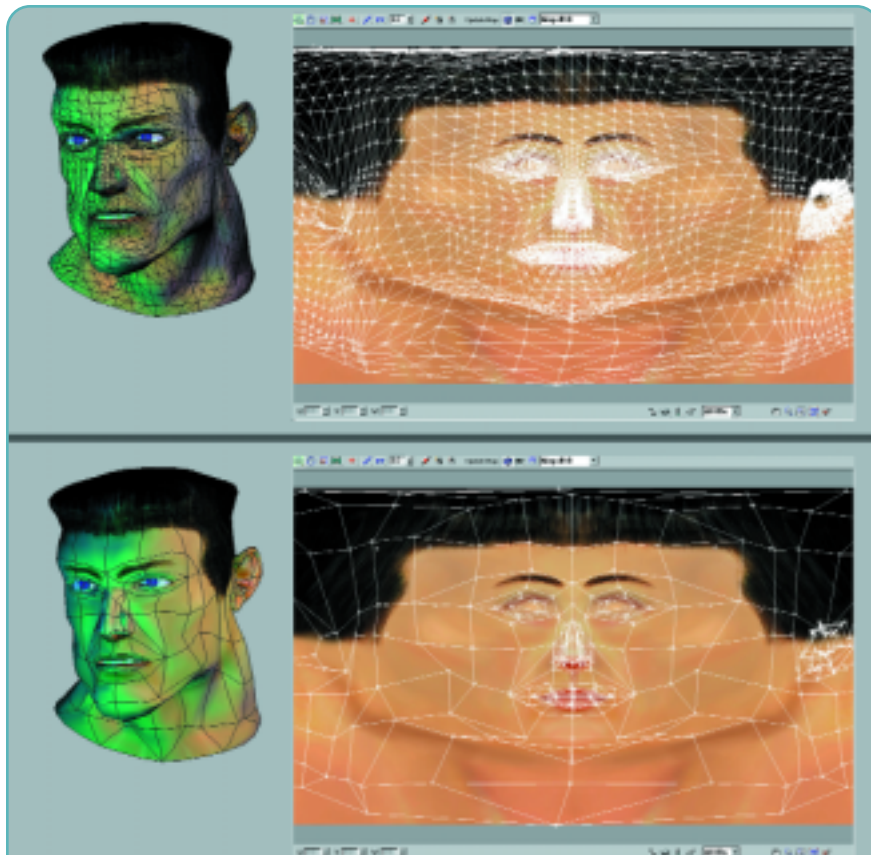


FIGURE 2. The Unwrap modifier in Max, comparing the high-resolution polygonal map (top) with an equivalent surface created with patches (bottom).

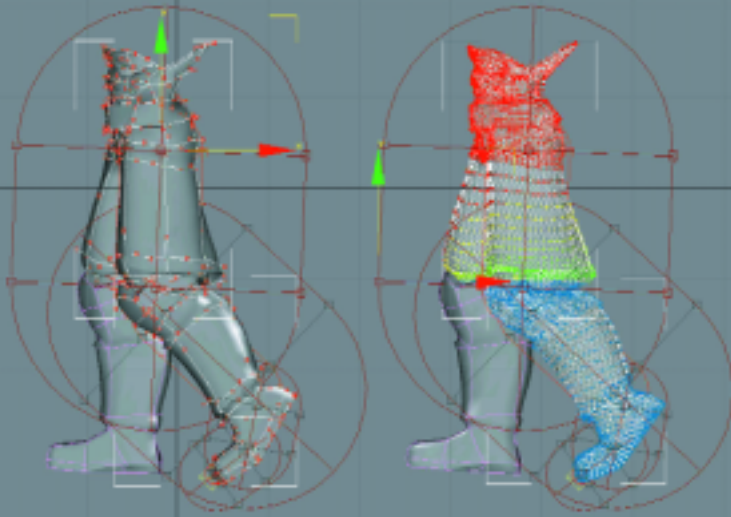


FIGURE 3. Comparing the number of skeletal envelopes and vertices between a patch-based leg and boot with weighting applied (left) and a polygonal one (right).

the model increases, so too does the number of vertices which may be slightly off in their weighting values, which when animated can create a host of visible kinks and tears in the surface.

Removing or preventing these weighting errors and creating a smoothly contiguous animating surface is the primary goal of the animation setup process. Clearly, when using standard polygonal methods the animator's effort and effectiveness are linked directly to the model's complexity. In order to produce good, quick results, we again need to sever this link between the animator and the complexity of the data. A visual representation of how this occurs can be seen in Figure 3. In the example on the right, a fairly detailed polygonal mesh has been weighted to a skeletal system. The skeletal envelopes and vertices of the mesh are clearly visible. An animator tasked with setting up this mesh has a long day ahead of him. On the left, we see a patch-based model of equivalent complexity. The number and density of the polygons present at run time are identical, yet the number of vertices the animator has to weight is substantially less. As a result, there will be fewer potentially errant vertices, and subsequently fewer total errors to correct and troubleshoot.

Just as with the texturing process, in a standard polygonal implementation each and every LOD generated needs to go through a similar setup process. With a control point lattice implementation such as patch surfaces, the process need only be executed once, and the higher-level LODs can be derived instantaneously from the original surface simply by increasing the number of subdivisions.

One additional advantage of B-spline patch surfaces specific to 3D Studio Max 3 is that the control point handles can be given weighting values independent of the control points to which they are associated. This gives the animator an additional layer of controls over the deformation process and,

when successfully implemented, can enable the animator to simulate complex surface behaviors such as muscle and cloth deformation. Figure 4 shows an example of this. On the left, the control point handles have all been weighted the same as their respective control points. On the right, the handles have been weighted differentially so that when the joint flexes, the surface simulates a muscle bulge.

Taken in combination with the time saved actually modeling the geometry, the additional benefits in texturing and animation add up to a significant production cost savings. To quantify this, Figure 5 shows how the two approaches might compare in a real-world art path. Clearly, the cost savings associated with such a surfacing strategy are well worth the time invested in adjusting to the new technique. In contrast with the standard polygonal mesh approach, the control point lattice method has the potential to minimize production effort while maximizing the fidelity of the final result. The patch method can unfetter you from the burden of content complexity, allowing you to focus your efforts instead on the more creative aspects of your craft.

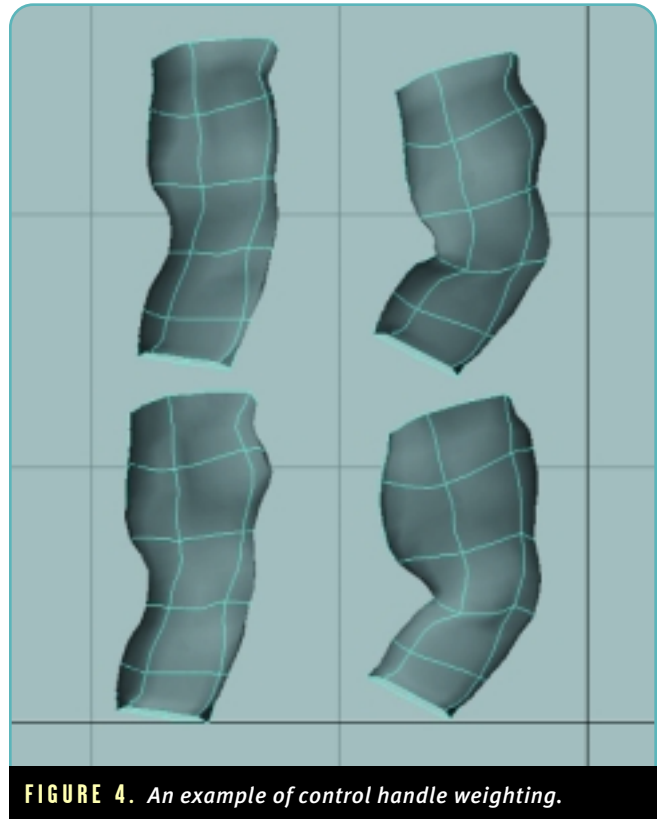


FIGURE 4. An example of control handle weighting.

Be Fruitful and Subdivide

This wraps up our discussion of surfacing techniques, though we may eventually pick it up again as the next generation of tools becomes available to us. It's interesting that as artists our primary motivation for exploring these surfacing techniques is related directly to issues of content production — that is, development time and cost. As such, it hasn't been necessary to focus on how these surfacing techniques can benefit from an application which has been optimized for nonpolygonal rendering. Clearly though, this is the direction in which the technology is headed.

The benefits of using a procedural subdivision technique are manifold, generating huge cost savings in transform and calculation speeds as well as data compression and rendering speed. For example, in an

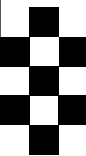
Stages of Main Character Creation	Standard Polygonal Mesh	B-Spline Patch Surface
Modeling:	Time/days:	Time/days:
LOD 1 / 400 polygons	0.5	2.0
LOD 2 / 1,200 polys	1.0	0.0
LOD 3 / 4,000 poly	2.0	0.0
LOD 4 / 12,000 polys	2.5	0.0
Texturing:		
Texture Maps	3.0	3.0
LOD 1 Mapping	0.5	2.0
LOD 2 Mapping	0.5	0.0
LOD 3 Mapping	1.0	0.0
LOD 4 Mapping	2.0	0.0
Animation Setup:		
LOD 1	0.3	1.0
LOD 2	0.5	0.0
LOD 3	1.0	0.0
LOD 4	1.5	0.0
Total Time	16 days	8 days

FIGURE 5. A snapshot schedule showing the time saved using the patch surface method compared with the traditional polygonal method.

engine optimized for rendering patch surfaces, the data (geometry, texture coordinates, weighting information, and so on) sent to the engine can be kept at its lowest resolution, giving the engine the ability to subdivide the surface only as necessary. Furthermore, because all of the necessary information required to describe the surface exists within the low-resolution control point lattice, the transforming and lighting calculations can be executed before the surface is subdivided, drastically reducing the processing power required to render the complex surface. This translates directly to faster rendering and more efficient data storage, which to an artist, is the equivalent of an almost unlimited canvas. ■

Acknowledgements

Special thanks to Dave Coathupe and Louise Smith.

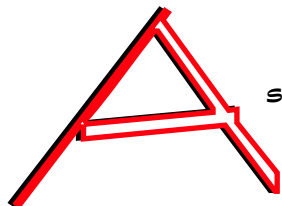


Building Character

by Toby Gard



Duke Nukem © 3D Realms, Lara Croft © & ™ Core Design Limited and © & published by Eidos Interactive Ltd. All Rights Reserved. Indiana Jones © & ™ Lucasfilm Ltd. All rights reserved.



As games evolve into an increasingly complex and sophisticated medium, game characters are also experiencing a considerable metamorphosis. Just a few years ago, a game character had to be simple enough so that it could be represented clearly under very

severe artistic limitations. Essentially, game characters were just icons, amorphous blobs, or tiny men rendered from a handful of pixels. But steady technological progress has slowly opened up possibilities for more believable and realistic characters. The question now is, how does a game developer leverage all of these additional technical resources to create more compelling characters?

This article attempts to set out the elements of design that need to be

addressed in order to create

a memorable and power-

ful game character. 



TOBY GARD started working in the games industry in 1994 at Core Design in Derby. While at Core, he conceived the game TOMB RAIDER and its main character, Lara Croft. Toby was both the designer and lead artist on the project. His roles covered everything from story development, storyboarding, FMV generation, in-game animation, character design and modeling, level flow, title and load screens, box art and marketing/PR materials. In 1997, having left Core Design, he set up Confounding Factor with business partner and lead programmer of the original TOMB RAIDER, Paul Douglas. Confounding Factor is currently producing the game GALLEON due for release this Christmas.

The greatest genre for characterization has always been the adventure game. Early Infogrames classics like PLANETFALL and LucasArts' later adventures such as FULL THROTTLE showed how effective "game actors" could be. These characters used spoken dialogue and were portrayed with emotional personalities. Now these game actors are moving into new cross-genre 3D games, and it is here that they really thrive. In fact, they may be becoming a little too successful; we're beginning to see games sold purely on the strength of these characters. Worse still, it seems that characters today are inserted into games that simply don't require them, perhaps because it is now seen as a marketing necessity.

The single most important rule in character design is "the game comes first." The type of game you're developing will determine most of your character-creation decisions. A character is just a tiny element of any game, and in many cases, it is a superfluous element. If a game works without a character, it shouldn't have one. The rules of elegance apply — look for the clearest, simplest way to represent an idea.

Character Identification

You can split games broadly into two groups: those with a first-person point of view (POV), and those with a third-person POV. Although that difference between them may seem slight, it is absolutely fundamental, as the psychology of the two POVs is drastically different.

A first-person game invites players to immerse themselves in the game, to play as though they themselves are in the game experiencing the events firsthand. On the other hand, the third-person game makes a distinction between the player and the on-screen character; they are separate entities. In a third-person game, the player is controlling a character rather than becoming the character.

This difference utterly splits character design into two entities that I will

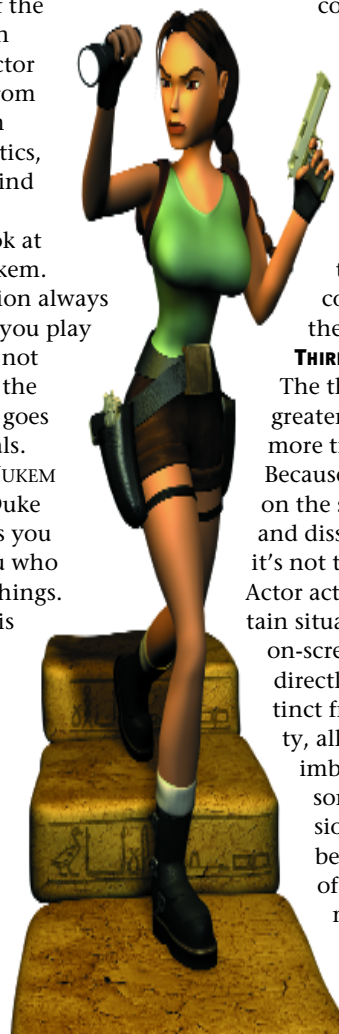


FIGURE 1. *There is a fundamental difference between a first-person character and a third-person character.*

refer to as the "Avatar" and the "Actor." The Avatar is simply a visual representation of the player's presence within the game world. The Actor is a character distinct from the player, with its own personality, characteristics, and, to some extent, mind (Figure 1).

For example, let's look at Lara Croft vs. Duke Nukem. Although some projection always occurs in games, when you play TOMB RAIDER, it is Lara, not you, who gets eaten by the Tyrannosaurus rex and goes around shooting animals. When you play DUKE NUKEM though, even though Duke shows a personality, it's you who gets killed and you who goes around shooting things. Many games muddy this distinction and they lose a great deal of impact on players as a result.

FIRST-PERSON POV: THE AVATAR. A first-person game should make it as easy as possible for players to believe that it's actually themselves in the game. The main character (the Avatar) must not interfere with the



players to create their own. This second method is even more valid when it comes to multiplayer games.

In a first-person perspective, many of the techniques of storytelling and characterization common to other mediums can't be used, simply because you don't really know what the main character, being completely controlled by the player, is going to do.

THIRD-PERSON POV: THE ACTOR

The third-person POV allows far greater freedom to tell what is a more traditional story form. Because the character (the Actor) on the screen is a separate entity and dissociated from the player, it's not too disturbing when the Actor acts of its own accord in certain situations. Even though this on-screen entity is controlled directly by the player, it is distinct from the player's personality, allowing the designer to imbue the Actor with a personality of its own and occasionally control how it behaves. This extra element of control over the game makes it possible to use some of the less intrusive storytelling and mood-enhancing devices that have evolved in film.

Making an Actor

From a design point of view, game characters can be sorted in order of design detail:

- **AVATAR.** These characters require visual design only.
- **ACTOR.** Full character design, but with a necessarily one-dimensional personality so that the player can flesh out its motivations. The trick is to strike a balance between establishing the actor's personality without letting that personality disturb the player.
- **NON-PLAYER CHARACTERS (NPCs).** These require full character design.

We all use very powerful subconscious mechanisms to judge people visually, whether we realize it or not. When you meet someone, the amount of information you gather from them using your eyes is incredible. You take into consideration their shape, height, sex, race, physical attractiveness, hair, clothing, makeup, cleanliness, facial hair, age, weight, stance, facial expressions, body language, movements, and so on. You perceive a vast amount of information almost instantly and without really trying. Your brain then begins to make assumptions about that person using built-in pattern-recognition techniques, most often based on your personal set of stereotypes. In contrast to these visual cues we pick up on, the slow linear stream of spoken information is incredibly small. After a while, our opinions may be re-formed based on a person's personality, but for a long time it is still filtered through our preconceptions based on our first impression. So to create a really good character, you have to control all of the visual clues that people use to judge each other and establish a clear, unified message to make players interested in — and ultimately like — your character.

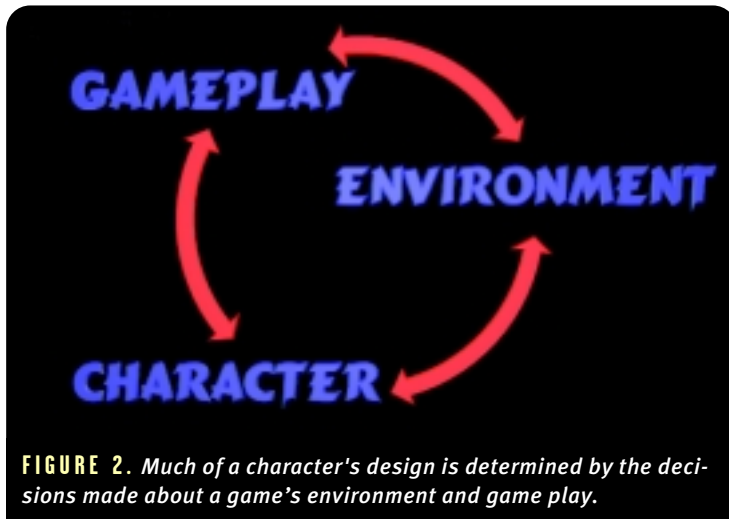


FIGURE 2. Much of a character's design is determined by the decisions made about a game's environment and game play.

could be achieved, most games would consist of overly complex, messy, and irrelevant details. Similar to cartoons, games use simplified representations of real-world ideas, stripped of the massively complicated rules found in reality. Therefore, to make the greatest impact, we have to caricature; we must amplify the aspects we want players to focus on. This is the route to making games a more powerful medium.

32

Style and Exaggeration

In the early 1930s, Disney animators were struggling to bring the same depth of acting skills to their cartoon characters that actors were achieving in live-action films. Cartoons are a deliberately simplified representation of reality, stripped of the incredibly complex subtleties we are accustomed to in the real world. These animators realized that they could never portray the same subtleties through animation, since the medium was too broad by nature. Instead, they exaggerated all the subtle body language and emotional expressions made by actors until they became almost a pantomime of good acting. Through exaggeration, cartoons are able to elicit very powerful emotional responses from an audience, because cartoon acting is a concentrated version of live acting.

Computer games are much more akin to cartoons than films. Games aren't very good at imitating reality, because elegance requires them to be visually limited. They can't mimic the incredibly complex world we live in. If such detail

The Three Linked Elements

In a character-based game, there are three intrinsically linked elements: environment, game play, and character (Figure 2).

A powerful character must be well adapted to its environment. Good characters typically have some element about them that makes them especially suited to their world. Take Indiana Jones, for example. Indiana hangs around all day in tombs and ancient sites that are filled with dangerous traps and angry natives. He is a tough, strong person, but more importantly he is an archaeologist, and this is what makes him so well suited to his environment. From a character creator's point of view, you would probably come at this in reverse: the character is an archaeologist, so therefore his environment will be tombs and ancient sites.

While the link between character and environment is true for noninteractive media as well as games, game developers must also consider an even more important factor: game play.

Game play affects how an environment works. There is a link between what the player can do and what the environment contains. Game-play decisions are also dictated by a character's special abilities, so game play and character design are linked,



The evolution of Mickey Mouse.

© Disney Enterprises, Inc.



TM & © 2000 Marvel Characters Inc. All Rights Reserved.

too. Look at the character Bob in Shiny's MESSIAH. This little angelic character goes around and possesses people, and this attribute has massive game play significance, dictating exactly how his environment has to be populated and designed. Thus, as you change the attributes of any one of these three elements, the other two elements are affected as well.

Simply stated, the character design process cannot be isolated from the game design process. Many elements of a game character are completely decided by game play and environment.

Visual Design

The visual design of a character can be split broadly into two aspects: physiological form and the clothes worn (if any). Physiological differences between one human and another are fairly slight; there is some variation in skin tone, size, hair, build, and weight. Gender is the only major variance, and apart from that I'm afraid all humans look alike to me. Clothing, however, varies greatly in color, shape, purpose, and significance. That is why costume design is so important.

There has been a sudden surge of female main characters recently, which is good since it redresses the gender imbalance in our predominantly male industry. The choice of a character's gender is critical, and not simply from a marketing perspective. (I won't talk about any aspect of character design from a marketing point of view since I don't think it is wise to approach any aspect of design from that angle. If you design a character to be liked by players, marketing opportunities will follow of their own accord.)

A character must have dignity. Any design that objectifies the character (that is, encourages you to think of it as an object rather than a living being) will prevent players from empathizing with it and relating to it. Creating this living essence is the trick to making people like a character. Far too many female characters have been put into games simply as tokens, usually as sexy

bodies for use by marketing departments. This is something to avoid.

Male and female players react to the gender of a lead character in different ways. Players usually want to protect a good character of the opposite sex, so drawing on a person's primeval and innate response to the opposite sex is a powerful tool. If the character is attractive, believable, and commands respect, players will grow fond of it. On the other hand, someone playing a good character of the same sex usually grows to admire the character and its characteristics. If the character has been designed well, the character can even develop into a role model for some players. Whatever the gender of the character, the fundamental rule for getting people to respond positively to the character is that the character must be likeable and admirable.

The Halo Effect

Some great psychology experiments have been conducted about the "halo effect," the results of which can apply to character design. Briefly, the halo effect postulates that we treat attractive people better than we do ugly people. Not only that, but we often make all sorts of subconscious assumptions based on looks. Good-looking people are often assumed by

strangers to have other positive traits, such as being "poised, independent, sociable, interesting, exciting, and sexually warm" according to Brigham. On the other hand, unattractive people are apparently seen by strangers as more "deviant," according to Jones and his colleagues (see the References section at the end of this article).

I've heard arguments that game developers should not create a cast of highly attractive characters, either because it provides unrealistic role models for children or because some equate creating sexy characters with sexism. I don't consider it sexist to represent males and females in an equally distorted way, but action comics have been criticized for years because they portray exaggerated strength and sexiness in characters. Note however, that as a medium, comics command one of the largest groups of enduring and instantly recognizable characters.

Thus character designers should do everything in their power to make characters as attractive as possible. People's first impressions of characters will almost certainly come not from what they do, think, or say, but what they look like. If the character makes a good first visual impression, players will likely stay focused on it, allowing you to further entice them with the character's personality.

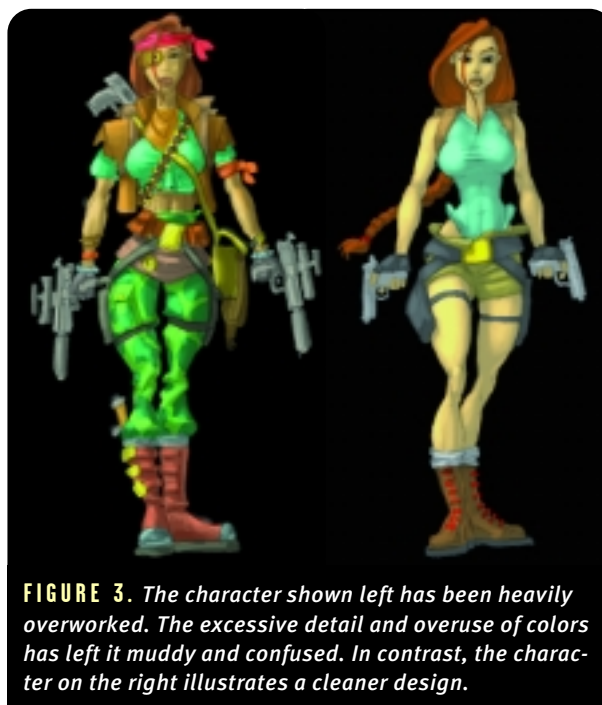


FIGURE 3. The character shown left has been heavily overworked. The excessive detail and overuse of colors has left it muddy and confused. In contrast, the character on the right illustrates a cleaner design.

Costume Design

Keeping a consistent costume throughout a game is the best way to help imprint the character in a person's mind, so

costume changes should be avoided as much as possible until the character's visual design has become fully established. Once established though, giving a character some costume changes will increase its believability.

Let's look at Indiana Jones again, as he appeared in films. Indiana wears his costume with some variations; sometimes without the jacket, sometimes without the hat, and in certain brief scenes he wears a completely different outfit. All in all though, Indiana keeps a strong sense of consistency which contributes to a solid, consistent image in our imagination.

The simpler a costume design is, the easier it is for a person to recognize and remember it (Figure 3). Complex, muddy, drab, and over-rendered clothing results in confusing, muddled characters. Try exaggerating essential elements of your character until you can strip it down to its essence, the simplest representation that gets across the meaning you are trying to represent.

Color schemes should be kept bold and within a limited palette. That way the colors begin to symbolize a character, just as blue, gray, and a dash of yellow invokes Batman (Figure 4).

It's always a good idea to try to put elements into a design that help symbolize the character's essence. Obvious examples of this technique include Hermes's winged feet or the web designs on Spiderman's costume. You can also leverage the subtler associations people make about clothing and accessories to provide more clues about the character: glasses for intelligence, cardigan sweaters for massive sexual magnetism, or whatever.

Personality Counts

Once you have a good, strikingly designed character, the next step is to work on conveying its personality. Again, the extent to which you need to portray personality depends on

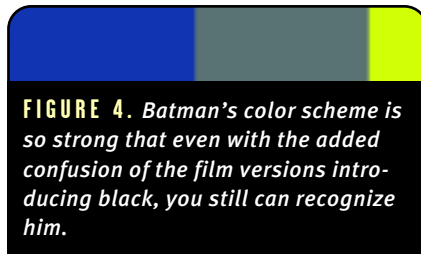


FIGURE 4. *Batman's color scheme is so strong that even with the added confusion of the film versions introducing black, you still can recognize him.*

whether the character is an Avatar or an Actor.

I can't overstate the importance of body language when creating images of any character. Not only do people make dozens of snap assumptions based on a person's physical appearance and apparel, they also make strong judgments based on the way people carry themselves and their physical presence.

Just the way people stand reveals enough information for others to read all sorts of traits about them. The trick is to be aware of this, know what messages you want to give, and provide those cues clearly (Figure 5).



FIGURE 5. *Even with all other visual clues stripped away, a person's pose can send very clear messages about a character.*

Far too many characters are portrayed in static poses designed to look "hard." Unfortunately, the quintessential hard look is the emotionless, squinty-eyed, Charles Bronson-style stance. This does not allow people to "read" a character at all, but it works with Bronson and Clint Eastwood because that's the point of the "Man with No Name" tough guy — he's supposed to be unreadable. So many characters imitate this look that they all fade into an obscure morass of similarity. Instead, create some attitude through poses that provide clues about the character's personality.

Poses are especially important for the static artwork typically used on game boxes and by marketing people. Dynamic poses are far more interesting, striking, and memorable than static ones. If a character is meant to be an action character, then for goodness's sake show them in motion.

Consider taking the final step and exaggerating a character's pose to the point that it actually begins to signify the character. If you create a set of

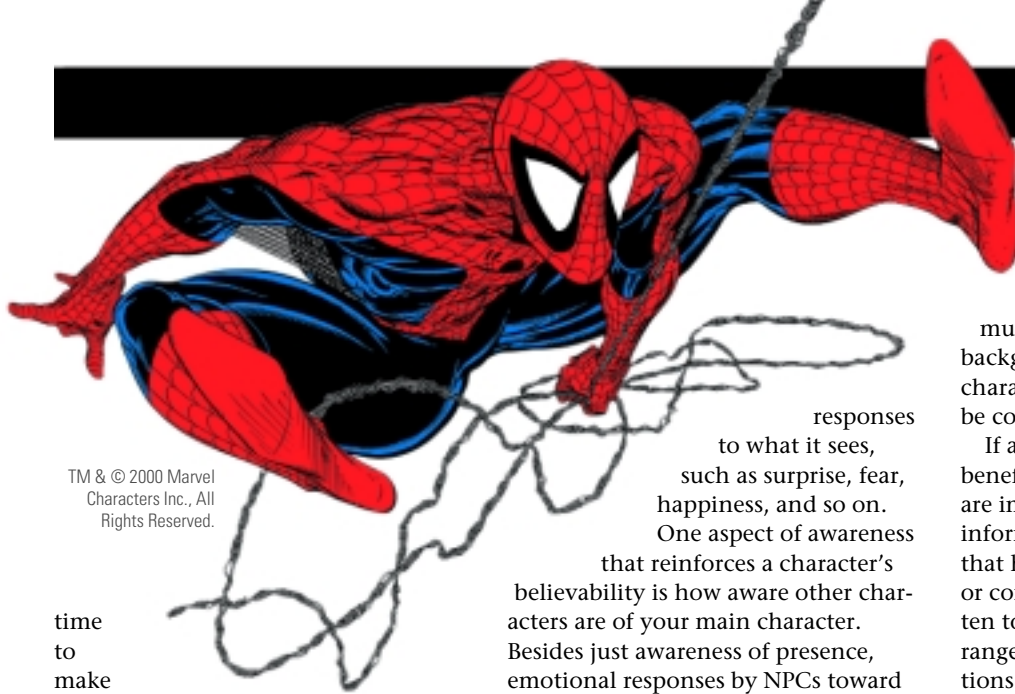
strong, distinctive poses for your character, people will recognize these poses even when seen in silhouette or from a great distance. Spiderman is an excellent example of a character that has a great array of extremely strong, immediately identifiable poses (Figure 6).

Motion, The Fourth Dimension

Just as with visual design and poses, it's incredibly important to consider how a character moves, and design around that aspect. The most important step is making a character move in a convincing way. That means showing weight, balance, and inertia. Unless you pay particular attention to the solidness that a character demonstrates while interacting with its environment, people will never accept it as anything but a group of weightless polygons. Every time a foot slides or a character snaps between animations, the illusion of life is totally shattered. Since computer game characters constantly repeat their animations, it is worth the extra



TM & © 2000 Marvel Characters Inc., All Rights Reserved.



TM & © 2000 Marvel Characters Inc., All Rights Reserved.

time to make movement animation as flawless as possible.

Apply the concept of unique poses to the actual motion of a character. The way people walk suggests vast amounts of information about them, such as how they feel about themselves and their surroundings. If a character's movements are a consistent, exaggerated representation of its inner self, you can build up its personality while it moves about its environment. (As a side note, there is a vast difference between realism and believability — I feel you can always get a stronger, more universal emotional response from high-quality hand animation than you ever can from motion capture.)

Awareness

The key that Disney animators ultimately found to creating the illusion of life was showing their characters thinking. Making a character aware of its environment has an incredible impact on its believability. If your character examines its surroundings and the other characters in it, it automatically appears to be thinking about what it's looking at. Awareness doesn't just end with where a character looks, it extends to its reactions to its environment. A character can give emotional

responses to what it sees, such as surprise, fear, happiness, and so on.

One aspect of awareness that reinforces a character's believability is how aware other characters are of your main character. Besides just awareness of presence, emotional responses by NPCs toward the main character add immeasurably to its substance and believability. Note, however, that the player will be affected by how NPCs react to the main character. Unless you want to lower a player's opinion of the main character, NPCs should generally react positively towards it.

What's Your Story?

I like to work out some sort of background history for a character, even if it only helps to flesh things out in my own head. Don't go too far when creating the history, though — it's risky to give a main character loads of hidden motivations that might conflict with the player's, and the character could react in ways that make the player feel uncomfortable. At the end of the day, a game character shouldn't

have anything more than superficial personality traits since, whatever the POV, the player needs to retain as much control as possible. A bit of background just helps solidify the character design process so that it can be consistent.

If a character is going to speak, the benefits of having a decent voice artist are immeasurable. We glean a host of information from each other's voices that has nothing to do with the nature or content of the words spoken. We listen to the timbre, the accent, and the range of a voice to make basic assumptions. More importantly though, we listen to the rich subtle inflections that hint at whether a person is sarcastic, sincere, intelligent, or has a sense of humor. Only highly skilled actors can evoke all of this hidden information as they deliver their lines. If anything is lacking in the voice acting, then you can't inject personality into a character even if all its other design elements are spot-on.

So much about character design is subjective. I mean, what is attractive? On that point alone you could argue for hours. But one very important thing remains to be stated. Any of the points I've discussed can, and probably should be turned on their heads if you want to create new and exciting characters. Guidelines such as these are just guidelines: ingenuity, humor, and originality require that rules be broken. ■

FOR FURTHER INFO

Thomas, Frank and Johnson, Ollie. *The Illusion of Life: Disney Animation*. New York: Hyperion Press, 1995.

McCloud, Scott. *Understanding Comics*. Northampton, Mass.: Kitchen Sink Press, 1994.

REFERENCES

Brigham. "Limiting Conditions of the 'Physical Attractiveness Stereotype: Attributions about Divorce.'" *Journal of Research in Personality* 14 (1980): 365-375.

Jones, Hannson, and Phillips. "Physical Attractiveness and Judgments of Psychotherapy." *Journal of Social Psychology* 105 (1978): 79-84.

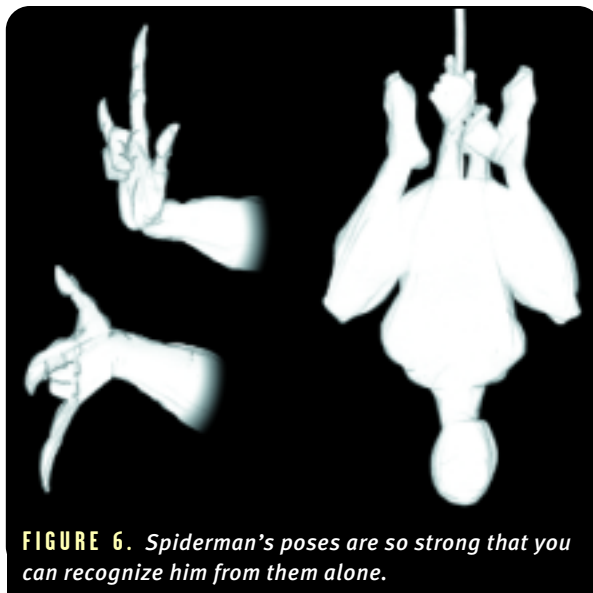


FIGURE 6. Spiderman's poses are so strong that you can recognize him from them alone.

Testing When You're Worlds Apart

by Mark Thomas

38

There are a number of situations in which a testing team can find itself working with an external developer. Probably the two most common scenarios are the publisher-developer relationship (which itself can operate under a number of models) and when the testers work for an outsourced testing company. Testing games can be difficult under the best of circumstances, but when developers and

testers report to different companies and are separated geographically, complications naturally arise. But regardless of your business structure, the testing and development teams should share the same goal: to release the highest quality game possible. Here are some guidelines to help an outside testing team achieve that goal.

Kick It Off Right

It's important to kick off the product testing process well. Here at Microsoft we try to spend some social time with the development team and work through our anticipated project issues. Whether you're a developer or a tester, you should see how the "other side" operates and meet everyone you'll be working with as early in the process as possible. The hard discussions will begin sooner than anyone would like,

but they become easier when some personal contact has already been made. People also tend to be less flexible in e-mail than they are in person.

Early in the game's development, the testers and developers should sit down and develop a shared glossary of terms that are relevant to the project so there's no miscommunication later. Terms such as "milestone," "alpha version," and "code complete" mean different things to different teams, so establishing and documenting terms from the outset will head off misperceptions. What terminology you chose isn't important; what is critical is that all parties understand that terminology. Take time to define what "quality" means to the testers and developers,

and make sure that everyone buys into that definition. Do the same with the definition of "bug."

Decide the methods by which you will measure the status of the project, and how these measurements will be conducted. There are many ways to measure a product, and you will probably use a wide variety of metrics to judge progress. You can hinge milestone acceptances on bug count, feature implementation, play-testing results, multiplayer success rates, performance, or all of the above. Clarifying this from the beginning should minimize arguments later, when questions such as "What is 'good enough'?" and "When do we know when the game is 'done'?" arise. This agreement

Mark Thomas is a test lead in the action and strategy games group at Microsoft and is currently working with developers on two continents. He can be reached at markct@microsoft.com.

should be sought at the beginning of the product, and continually reinforced as the project proceeds. It is best to document these agreements in writing once a consensus is reached. This gives the development team a goal to shoot for, and there is no confusion as to where the bar is. At Microsoft, milestone criteria not only list the features to be completed, but also specific bug-tracking goals and multiplayer success rates. For example, a milestone might have the criterion "no active severity 1 bugs, and at least 60 percent success on four-player ISP games."

Testers should continue to provide these measurement criteria to the development team as early and often as possible. I try to prepare the metrics and expectations for the next set of deliverables as soon as the last set has been signed off on. If your milestones are more than four to six weeks apart, try to set measurable goals and targets no more than six weeks from your last checkpoint.

I recommend sharing your upcoming build verification tests and milestone acceptance tests with the developers. As soon as you have written them, send them to the development team. Encourage the development team to run these tests before each build is delivered. By doing so, the developers will know exactly what is expected of their game and they'll have the means to assess its progress prior to submitting the build to the testers. This also helps manage the expectations of the development team, and prevents them from saying, "We didn't know the game was supposed to do this!"

Setting up and documenting how testing will be handled is critical. There are myriad details associated with testing a game, and you should make every effort to lock down the steps of your testing schedule as early as possible. Set up processes you can live with. In general, you will need to integrate your testing with the development schedule. There are times, however, when it is very important to make sure that the development schedule takes the testing needs into account. When schedules are being developed, I try to analyze them from two perspectives. First, I try to figure out what feature is going to be the most difficult to develop. Will it be the AI, the multiplayer code, or something else that must be finished before

content is created? Features like these are always risky, and when possible, front-loading them in the schedule helps make sure that they get completed and stabilized. Second, I look for features that will have a lot of impact on game play or take a long time to test. While these features may not be technically difficult to implement, they tend to require a lot of tweaking and tuning. If you get the core functionality of these features done and tested early, you should have plenty of time to adjust them and perfect the game play as the product evolves. Every title has its own list of risky features, and the sooner you can isolate and control them, the better.

Builds

Builds are important steps in the testing process; they are the critical and ultimate deliverable to the testing team. The testing and development teams should agree when and how often builds will be delivered. Typically at Microsoft we make this decision collectively with the producer, lead developer, and test lead.

Set up a schedule for delivering builds that makes sense and try to stick to a pattern. Maybe it is the first of the month, or every other Friday. I find it helps to have a set day on which the build will be sent. If the entire development team knows that every Thursday at noon is the last time to check in code before the build is made, it starts to become a habit. Typically at Microsoft the schedule for builds changes as the project progresses, and builds are delivered more frequently as the shipping date approaches. As a test lead, I begin to get a bit nervous whenever I go longer than one month without a new build being delivered for testing, but I expect the development team to be building the product internally at least once a week.



A configuration lab is one of the best resources a publisher can provide.

The testing team should make it clear to the developers what the expectations are for each build, and a written list is a great tool to have at the project kickoff meeting. Does the testing team want to receive the build via FTP or CD-ROM? Should a release executable or debug build be sent? Should a .PDB (program database) file be sent? Are there any support tools that need to be delivered at the same time, such as an editor or a file compression tool? Whatever you do, structure the builds such that the development team doesn't fear the prospect of sending a build to the test team.

Avoid requesting surprise builds from a development team. Assuming you've chosen a reasonable interval at which a build should be delivered (say, every other Friday), make sure that everyone understands what is expected. I recommend assigning responsibility to one person on the development team and one on the testing team for the build. The former is responsible for building the product and sending it off to the test team, and the latter receives or downloads the build and prepares it for release to the team. At Microsoft we typically do the setup for the product, so we ask developers to send us the raw files, and then the test lead integrates the build with the set up and distributes it to the rest of the test team.

There should be an understanding between the testing and development teams as to the importance of always

ID	Title	Area	Status
900	Getting a vital to another player in a multiplayer game is too difficult. User should only have to be one key.	Multiplayer	Open
901	Player doesn't get hit if enemy opens, gets hit for spotted escape pod.	Spac travel	Open
902	FD: Business amount of debris comes from enemy lighter escape pod.	Cable 104 Working	Open
903	One of the upper wires in the rag bomb lead is directly in front of a gate. Shows some burning into it frequently.	Cable 104 Working	Open
904	Careless Nex through the roof after Henry ship during exam.	Cable 104 Working	Open
905	Mission: Aliquam quickly takes out James tower on Latah in mission 10.	Cable 104 Working	Open
906	Mission: Speed obstacles shot on Latah when it you're already blown it up.	Cable 104 Working	Open
907	Mission: Maza jumps to the window if you're right at end of mission 10.	Cable 104 Working	Open
908	Mission: Carter doesn't release sky lighters by Latah in mission 10.	Cable 104 Working	Open
909	Mission: Losing one of the game's ships, you can't complete mission 10 after same is present.	Cable 104 Working	Open
910	Mission: Get success plus bonus after hitting both cargo ships get destroyed.	Cable 104 Working	Open
911	Mission: Alan wing is out of action after in mission 11.	Cable 104 Working	Open
912	Mission: Don't have to participate the same order in mission 11.	Cable 104 Working	Open
913	Mission: Speed calls 45 inside ship to take out tanks, then leaves in mission 11.	Cable 104 Working	Open
914	Mission: Two main time doing walking while others get up pods and get clear in Mission 11.	Combat	Open
915	German - TAC - Alliance Squads - leader: still home (staying over)	Localization	Open
916	German - TAC - Alliance Squads - profile not correct.	Localization	Open
917	Mission: Col. McGowan indestructible in mission 9.	Cable 104 Working	Open
918	Ship: Accidentally killing an spotted wingman gets you crash mission.	Spac travel	Open
919	Landing Car - Polish - Polish is misspelled.	Cable 104 Working	Open
920	Ship: You don't get a hit for destroying a Hanzo bomber.	Spac travel	Open
921	Mission: Get success + bonus after all 4 ships get destroyed in mission 6.	Cable 104 Working	Open
922	UI: Video completion in upper left corner of screen between cut scenes stops running to launch and Stefan.	Cable 104 Working	Open
923	UI: The language of the game should be UK, for localization mission.	Cable 104 Working	Open
924	UI: Can't map some keys in Control Configuration screen when using French keyboard settings.	Cable 104 Working	Open
925	Nanny Robot Cam: Camera goes through upper edge of Henry.	Cable 104 Working	Open
926	Mission: Fighters leave action sphere in mission 7.	Cable 104 Working	Open
927	Ship: No collision on most of an Eater after it's destroyed in mission 7.	Cable 104 Working	Open
928	Shooting Alliance debris gets you crash mission.	Cable 104 Working	Open
929	If a player hits the 2 keys on the keypad, he will get locked to the desktop. Player's have intended that for multiplayer games.	Cable 104 Working	Open
930	1-2 Players getting IPV during deathmatch games.	Cable 104 Working	Open
931	FX: Player can't remain inside exploding Strategic without damage in Mission 14.	Cable 104 Working	Open
932	German - TAC - Alliance Squads - General info window transition.	Localization	Open
933	Mission: Return Nex through FX Carter while driving in mission 9.	Cable 104 Working	Open
934	FX: Tagging drive wing rotations for captured large pods in Mission 9.	Cable 104 Working	Open
935	UI: WSPFXDLLA performed an invalid memory access. When Alan/Fab sit within Maza or EAC.	Cable 104 Working	Open
936	Mission: Maza won't continue Fliegen escape in mission 5.	Cable 104 Working	Open

FIGURE 1. We track everything in RAID. The database back-end gives us the ability to search and query on almost any element of a bug.

about the bug itself, there should be a more verbose description of the problem, and the shortest possible list of steps to reproduce the bug. The last component of a good bug description should be the expected behavior, had there been no bug. Often this seems obvious, but often the expected behavior of the product in this situation isn't clear or is different from the specification. If the tester takes the time to clarify what is expected of the product, the developer can quickly determine if there is confusion. Also, if a change to the product is needed, the developer knows what the tester expects the product to do before changes are made. We also make sure to include any debug information (call stacks, log information, and so on) and details about the hardware and operating system of the machine where the bug was found. If you can save the developer a phone call or an e-mail to ask a question, it's worth the extra time.

Make sure that the development team has equal access to whatever you use to track bugs. Ideally, every developer should have real-time access to their bugs at their desk. Whenever I set up a database for a new product, I open the first bug and fill it with a bunch of information about the process the team has agreed upon for bug tracking: things such as the life cycle of a bug, a template for a new bug, an explanation of the version numbering system, and anything else that's appropriate to the product. If anyone forgets the process or new people join the team, they can always open up bug number one and access this information.

Access to the bug database should be available to as many members of the team as possible. Testers, developers, artists, writers, designers, producers — everyone — should be able to get in to view bugs. Your bug list shouldn't be hidden from members of the development or testing team; it is the most accurate description of the state of the product. Also, avoid any bug-tracking solutions based on two different tools — the development and testing teams should use the same tool to capture this information so that tracking the status of bugs and sharing information back and forth between the two teams is easy and seamless. One shared database that everyone can access promotes communication and ensures that

keeping a buildable code base on hand. It is very important to build the project frequently, as a build is the best indicator of the status of the product. A process that results in frequent builds gets your product to a known level and gives you a great way to see progress. By keeping to a frequent build schedule, any big snag or step backwards is visible to the entire team very quickly.

Making a build for the testing team should not be an event; it should not take days. At Microsoft we encourage developers to invest in a build process. We try to help developers get a system set up that lets them build their product every day, just by pressing a button. If you cannot build the product, or something breaks this process, this is a big warning flag. With a frequent build schedule, such problems are exposed immediately.

Bug Tracking

Standardize a uniform way to report and track bugs using some sort of database system, one that lets you query the bug database for specific issues and get a quick snapshot of the product's status. A database is a lot eas-

ier for a team to work with than a spreadsheet. At Microsoft we have an internal tool called RAID ("kills bugs dead!"). It runs on a SQL back end and is accessible from every desktop at Microsoft. With some effort, RAID can also be set up and shared with our external partners. We put everything into RAID (Figure 1).

When a bug is entered into the database, it is assigned a priority and severity, tied to a specific area of the product, and then assigned to the appropriate person for fixing or research. When the fix is made, the bug status is changed from active to resolved, and then assigned back to whoever opened the bug. The tester then regresses the bug and verifies the fix before closing the bug. Only testers close bugs, which ensures that a fix is well tested before the issue is closed.

Bug reports should be as detailed as possible. When you are not at the same location as the developers, this is even more important. Reports should contain all the information that will be needed to reproduce the bug on a developer's machine. A typical bug report also contains a title, which is the one-line description of the bug. A good title makes clear in just one sentence exactly what the bug is. Next, in the details



everyone has the same information and priorities.

Team Communication

Every company has different styles and methods of communication. For instance, some development teams prefer to funnel testing information through their producer, while others want their developers to interface directly with individual testers. As such, it's important to determine how the testing and development teams will interact.

I've found that many producers want to restrict the contact between the testing and development teams to promote consistency in bug tracking and prevent developers from talking with random testers every time they call. Whatever communication system is put in place, it's important that the test team has a way to get questions and concerns addressed. However, once the "code complete" stage has been reached, it's important for testers to communicate directly with individual developers. Direct communication at this stage increases the productivity of both teams.

Triage

Bug triage meetings are much easier when the development and testing teams meet in person. Early in the development of the game, all those involved should agree upon the method of triage and who will conduct it. At Microsoft, the program manager, the test lead, the user education lead, and the support lead attend bug triage meetings from the Microsoft side. On the developer's end, usually the producer, designer, and development lead attend triage meetings, although this varies from team to team. Whatever the staff structure, make sure that the primary stakeholders in the product are represented.

As you triage bugs, make sure everyone understands each bug completely before deciding to fix it or blow it off. Because these are often difficult decisions with substantial consequences, and because bug reports can sometimes fail to clarify the scope of a bug, try to review bugs in the order of the tester

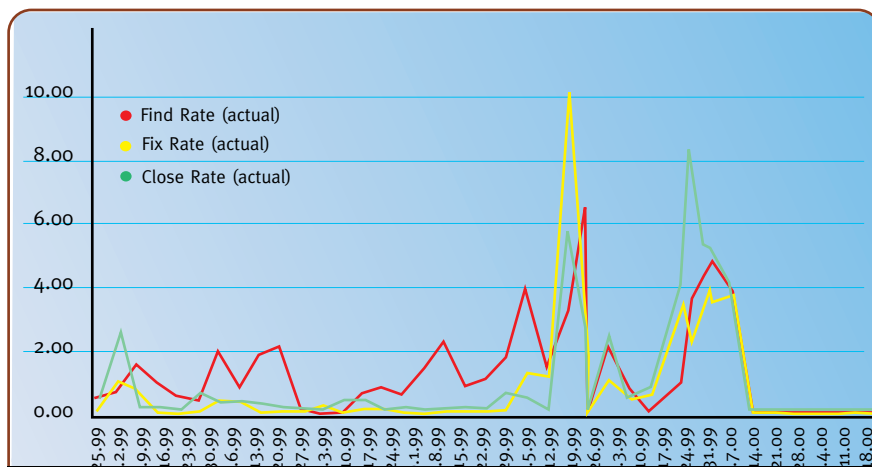


FIGURE 2. It is valuable to track how bugs are addressed. Your fix and find rates can help you figure out when your product is close to shipping.

that reported them and then invite that tester to the meeting when the bugs are discussed. This takes a little more time than just reading through the list with the committee members, but allowing testers to present their bugs often reveals reasons to fix the bug that no one had thought about. It also makes the test team feel that their issues are taken seriously, even if some bugs aren't fixed.

Honesty, above all, is critical to a good bug triage meeting. As a developer, be honest about the scope of the fix and any involved risks. As a tester, be honest about how severe the bug is, and your comfort level accepting a fix. Ideally, the decision to fix or ignore a bug should be a unanimous one.

Face Time Is Critical

There are some things that you can do throughout the process that will increase the likelihood of your test team delivering a high-quality game. Above all, share information you have about the product with other team members, both up and down the chain. Status reports on the team and the product are never a bad idea. Most product leads at Microsoft write status reports at least weekly, which often include rolled-up status reports from their team. These reports are typically available on an internal web site or sent out in e-mail, and are available to the group management and the entire team. Part of a tester's job is to communicate the status of the product — at

Microsoft that is even in the tester's job description. In my experience, more software development problems stem from poor communication than either technical or financial issues.

Face time between testers and developers is vital. Don't be afraid to put people on an airplane and hold meetings between the testing and development teams. Where you hold such meetings should depend on the meeting topic. For instance, as a publisher, there are things that we do at Microsoft that our developers aren't as equipped for, so we often have the development teams fly to our office and spend a few days or weeks on-site to solve specific problems. Take advantage of testing space and park whoever is fixing your configuration bugs in the configuration lab — do whatever is appropriate for your project.

One project at Microsoft was within a few weeks of shipping when suddenly there was a major problem with the multiplayer technology. It was so bad we couldn't even complete LAN games, let alone games played via the Internet. Since it was so close to our target ship date, we scrambled to round up as many testers as possible to analyze the problem, and had three development team members join us for almost two weeks. The problem unfolded before us like the layers of an onion; we fixed one bug only to find another right underneath. Having the developers work alongside testers helped fix the problem rapidly. We eventually shipped the game on time with a solid multiplayer component.



Understand Your Role As a Tester

Testing is a key part of the process of delivering a game. However, outside development teams may not realize the exact value you can add to their game. There is a great deal of variation in the quality of testing resources in our industry, so development teams are often leery of letting a new testing team exercise too much influence over their product.

To combat that situation, make your motivations as a testing team clear to the developers. Often this can be as simple as stating to the developers, "My job is to help you make this the highest quality product possible." One of the best ways I found to make developers comfortable with our test team is to establish ourselves as a service for them. This doesn't imply that we testers are a lower class of people. We just let the developers know that we think of them as our client, and that our relationship with them is based upon this belief. We want to know what we can do to make their jobs easier. We ask developers if we can unit test their code before it gets checked in, if we can run specific tests right away upon build delivery, and so on. Take the time to figure out what will make the process easier for the development team and see if you can provide it.

Often developers don't know how to make the best use of testing resources. Sometimes they don't really understand what testers do exactly. If you make yourself as available as possible to them, you will find that most developers become quite excited about the prospect of having testers work with them directly. I've seen situations where a developer and tester worked so well together prioritizing which bugs to fix in their areas that together they convinced the team when not to touch something. A cooperative developer-tester relationship like that can make much greater strides toward releasing a bug-free product than if the developers and testers work against each other.

During the course of a game project I've seen producers and developers ask testers to accept or change something about the game. (The common request is, "Can we let this slide a little?") When you're asked to alter the test plan in some way, determine the costs

and risks to your team and the product if you accommodate the request. In some cases, a little bit of flexibility can go a long way, and ultimately you have to do what's right for the product, not cling blindly to "the way things should be done." On the other hand, never put yourself or the test team in an uncomfortable position.

As a rule, show-stoppers should be where you plant your flag and demand a bug fix. The definition of a show-stopper is different for every company and every product. (Chances are that you will know one when you see one. A show-stopper is often a bug you find two days before you are supposed to ship that gives you that sick-to-the-stomach feeling.) The realities of software development prevent most developers from shipping 100 percent bug-free products, but you should look at every bug and consider its effect on the consumer. How many people will see it? How severe are its effects? Is it a crisis, or merely an irritant? Is there some sort of data loss associated with the bug? Is there a workaround for the player? How long will it take the player who hits this bug to get back into enjoying the game? What's the likelihood that it will generate a call to your technical support department? Can the player proceed past the bug at all? The answers to these questions must be compared with the risks and costs of fixing the bug. Look at it from the perspective of the consumer's expectations.

At Microsoft we don't have many hard-and-fast rules on how bug-free a product must be before we ship, but we try to maintain a high bar for the product. All bugs must be closed before a product ships. Being "closed" isn't always the same as "fixed," but it means that a tester agreed that it would be O.K. to ship the product with that bug. In general, we fix all known bugs

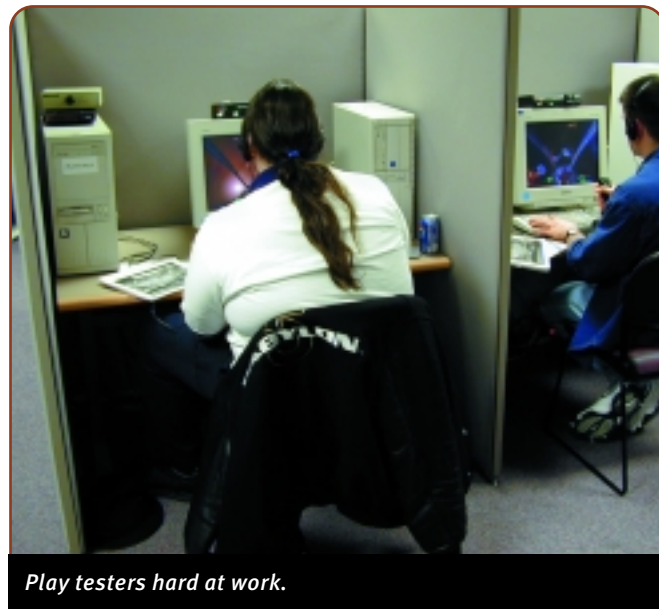


Photo by Bill Metters

Play testers hard at work.

that cause system crashes. We also compare the game against previous titles. For example, we compare our frame rate at our minimum configuration against other games running on the same hardware.

When the Budgeting Axe Falls

Testing is often shortchanged — sometimes never even considered — when the product development budget is drawn up. Other times, a "standard" cost will be inserted into the development budget's testing line item, without considering the game's scope. Ideally, the test lead should be involved in preparing the product budget, but that's rare. The sad truth is that testing funds are often the first to get scaled back when budgetary goals must be met.

When faced with a situation where staffing and financial constraints prevent you from properly testing a product, make it your responsibility to explain to management that these cuts or shortfalls will lead to insufficient testing and affect product quality. My test manager responds to the threat of budget cuts simply by asking senior management, "What part of the product would you like us not to test?" Usually the response is a bewildered look. Here are some other techniques for defending the testing budget:

- Remind those in charge that up-front testing helps avoid post-ship

expenses; technical support costs are often figured on a per-call basis, and depending on your company's structure, those calls can be a very expensive reminder of the benefits of testing.

- If management offers to patch instead of test, explain that patching a product can be a grim proposition that actually involves double costs: the time the team takes to fix bugs and prepare the patch for release, plus the opportunity cost of delaying the next product.
- Play the reputation card. Remind senior management that consumers remember low-quality games, and that brand reputation is as real a corporate asset as cash.
- Finally, bring up the fact that there have been a number of very expensive product recalls in the past couple of years due to problems like setup bugs that caused users to lose data, and the discovery of unauthorized content on CDs. That should be the nightmare scenario of anyone involved with managing product costs.

Ignorance of the benefits of testing isn't confined to upper management. There are developers out there who believe that they don't need any help from a test team; they are confident that they will find all bugs themselves and deliver a perfect drop to the publisher on the day of release. I suppose it is possible that some developers are capable of such perfection, but I've never met any. I've only met developers who *think* they are this good. Working with this type of developer tends to be painful and confrontational because such an individual won't value your contribution and probably won't be flexible about meeting the needs of the test team.

Unfortunately, no single solution shakes such developers from this mode of thought. Sometimes they react well when testers find bugs in their code, other times they don't. One strategy that has worked well for me is positioning testers as programmers' "aides." Explain to the developer that you can take some of the cleanup work off his hands. Offer to find invalid and unexpected cases in the program after they have unit tested their code. Market yourself as a service to the developer.

Make Each Project a Lesson

Every time you go through the testing process with a different developer you learn a new way to conduct tests. Don't forget these lessons. If something goes poorly, remember the episode and employ that knowledge in future projects. Examine the factors that led up to it so you'll be aware of them next time. And of course, if something works well, think about how you can apply those same tactics to a new situation.

Many companies, including Microsoft, hold thorough postmortem meetings after a game has shipped, in which everyone from the development and testing teams gathers together to talk about what went well and badly. Whether or not your company holds such meetings, prepare as if you'll attend such a meeting some day. On the day you start a new project, create a personal postmortem document, and make entries when things go well and badly. Having a written record helps you address problems for the next project, and constantly updating that document prevents incidents from disappearing from your memory once the product ships.

Above all, as a tester you must be loyal to the product's quality. It's your job to ship the best game possible. Whenever you find yourself unsure about what course of action to take, ask yourself, "What will make this a better product?" Once you ask yourself that, choices usually become easy. ■

FOR FURTHER INFO

Black, Rex. *Managing the Testing Process*. Redmond, Wash.: Microsoft Press, 1999.

Kaner, Cem, Jack Falk, and Hung Quoc Nguyen. *Testing Computer Software*. Boston: Thomson Computer Press, 1993.

Kit, Edward. *Software Testing in the Real World*. Reading, Mass.: Addison-Wesley, 1995.

Maguire, Steve. *Debugging the Development Process*. Redmond, Wash.: Microsoft Press, 1994.

McCarthy, Jim. *Dynamics of Software Development*. Redmond, Wash.: Microsoft Press, 1995.



Epic Games' UNREAL TOURNAMENT

by **Brandon Reinhart**



NREAL TOURNAMENT, released in November 1999, was, in a way, an accident. After the original UNREAL was completed, Epic wanted to follow up the project with some sort of add-on pack. UNREAL multiplayer code was very poor, so the team felt that an expansion that improved multiplayer would be ideal. As feature lists grew and patches to UNREAL were released, the add-on turned into a complete and independent game.



UNREAL TOURNAMENT has certainly seen a very nontraditional development cycle, one that I feel would not have succeeded in any other genre. Ultimately, our decisions paid off, because the game earned more than five “Game of the Year”

Brandon “GreenMarine” Reinhart is a 21-year-old programmer for Epic Games Inc. UNREAL TOURNAMENT was his first game after being recruited by Epic from the UNREAL and QUAKE 2 mod community. He is obsessed with games, game programming, and game design. When he isn't playing games, he can be found reading Michael Moorcock, painting miniatures, or listening to the latest in Norwegian black metal. Blodu Ok Jarna!



A close-up shot of the Black Thunder skin on Shane Caudle's Male1 model. This was one of the first new skins developed for UNREAL TOURNAMENT.

awards and is consistently rated in the top ninetieth percentile in reviews. The online community is producing excellent expansions and modifications to the game and we feel that UNREAL TOURNAMENT will be around for a long time to come.

Early Development

A proper look at the development of UNREAL TOURNAMENT begins with the completion of UNREAL. The Unreal engine was four years under development and the team was wearing down. When the game shipped, it met with a large amount of acclaim, but that positive image was tarnished over time as hardcore players began to complain about the terrible network support. The UNREAL team was now faced with several more months of work on the game, essentially to bring it to the point it should have been at when it was put on shelves.

Early in the process, plans were discussed to work on an official Epic add-on to UNREAL. The add-on would introduce much-improved network play, new maps, and probably some new game features. The original ideas for the add-on were never put on paper and it never had a name. I was hired by Epic in August 1998 to assist with patching UNREAL. Eventually I started to write new code for the add-on with Steve Polge.

Initial work on the add-on in early summer 1998 was made difficult by the fact that Epic was a virtual company. The last year of UNREAL's development took place in Canada, with the U.S.-based Epic team flying back and forth to work with Digital Extremes in London, Ontario. When UNREAL was finished, no one at Epic wanted to travel anywhere, but at the same time the team recognized that they needed to move to a central location. The team decided to relocate all of its employees to Raleigh, N.C.

By September 1998 everyone was together or had a travel plan. Work started to come together rapidly on the add-on project. Steve Polge had laid the groundwork for several new game types, including Capture the Flag and Domination. The level designers had five or six good maps ready for test-

ing. Throughout sporadic but intense meetings, the team agreed to focus the add-on entirely on improving the multi-player aspect of the game with new features and better net code.

The amount of content grew and we soon realized we had a much larger project on our hands than we had originally thought. In November, after meetings with our publisher GT Interactive, Mark Rein suggested we turn the add-on into a separate game. Initially, the team opposed the idea. We wanted to finish the project quickly and move on to something fresh. The promise of a much higher profit potential, coupled with our recognition of the state of the project finally led us to agree with GT. In December, the name UNREAL: TOURNAMENT EDITION was chosen, with "Edition" subsequently dropped from the title.

A Game Takes Shape

Epic's internal structure is extremely liberal, probably the most liberal in the entire gaming industry. Programmers work on the projects they want to work on, with major features being assigned to whoever steps forward to take on the task. Artists work with level designers but are given significant design freedom. Level designers work on the kinds of maps they think would be cool. This design philosophy pervaded UNREAL TOURNAMENT's development.

In December, I downloaded a sample of a new UNREAL mod under development by an Australian named Jack Porter. The mod, UBrowser, was a server browser using a Windows-like GUI. It was impressive, so I showed it to James Schmalz, lead designer at Digital Extremes, who said, "We need that, we need to hire this guy." A few weeks later Jack was a part of the team, expanding his UWindow GUI and reworking UNREAL TOURNAMENT's menus to use the system. Jack fit into the team perfectly, bringing a complete solution for the interface and menus as well as his own independent programming initiative.

Weekly meetings infused order into our chaotic corporate structure. Everyone would debate and yell about what fea-

UNREAL TOURNAMENT

Epic Games Inc.

Raleigh, N.C.
(919) 854-0070
<http://www.epicgames.com>

Digital Extremes

London, Ontario, Canada
(519) 657-4260
<http://www.digitalextremes.com>

Release date: November 1999

Intended platform: Windows 95/98/NT, Linux

Project budget: \$2 million

Project length: 18 months

Team size: approximately 16 developers

Code length: 350,000 lines of C++ and UnrealScript

Critical development hardware: Pentium II 400s with 256MB RAM and Voodoo 2 or TNT-based cards

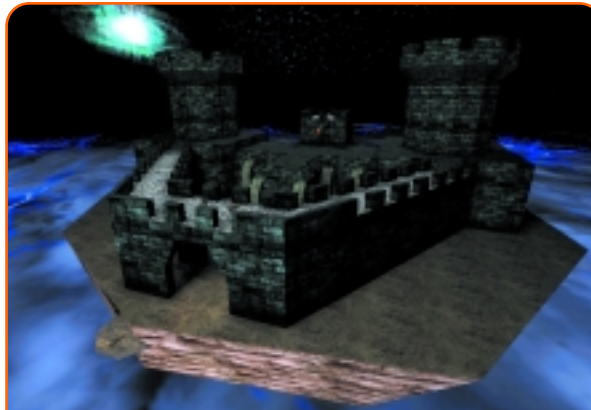
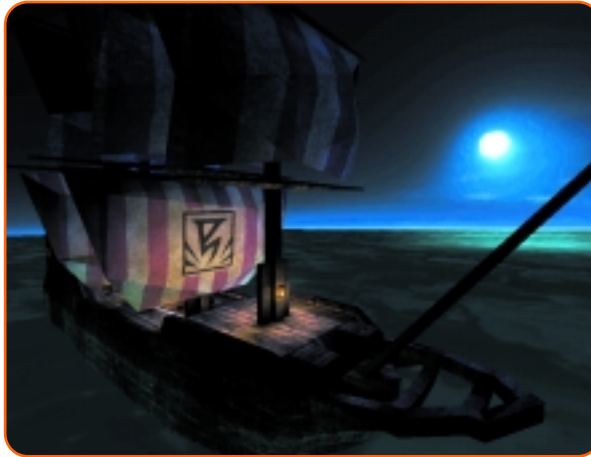
Critical development software: Microsoft Visual Studio, 3D Studio Max, UnrealEd

tures were cool and what features sucked. The assignment of major features was largely automatic. Tim Sweeney worked on improving net code and engine fixes. Steve Polge wrote the original AI code and focused on adding player orders and other improvements (in addition to filling out the new game types). Jack had the windowing system and a lot of menus to work on. Programmer Erik de Neve was in Europe putting together level-of-detail code as well as experimenting with next-generation technology. I worked on the single-player game, game-play features, scoreboards, HUDs, special level actors, tutorials, and wrote a lot of the game's story and character background content.

The best features were added entirely by the initiative of individuals. Level designer Cedric "Inox" Fiorentino designed CTF-Face, an extremely popular Capture the Flag map. I added the Multi-Kill system after a short discussion with lead designer Cliff Bleszinski sparked the idea, and Jack implemented decals shortly before we shipped. It was this individual creativity that ultimately bound the team together. Each new feature infused everyone with the enthusiasm to add more.

Once the first batch of new player models, weapon models, and maps was completed we realized we had a game quite different from UNREAL. Feedback from the UNREAL deathmatch community (including the highly vocal QUAKE community's complaints) also drove our designs. Subtle alterations to player movement and control changed the feel of the game completely. Some changes in game play — such as whether to enable weapon-stay in single player — were controversial, so we held polls on popular UNREAL message boards.

Throughout the spring and summer of 1999, Epic was pursuing contract renegotiations with GT Interactive. Everyone believed the game could ship at any time, so development became stop-and-go. We would be in a code lockdown one week and adding major new features the next. The result of



UNREAL TOURNAMENT's deathmatch maps were not constrained to any one particular theme or timeframe. Cliff Bleszinski's DM-Barricade, shown above, is a castle floating above a storm, while Pancho Eekels' DM-Galleon is a massive ship sailing the ocean (shown top).

this jarring development cycle was good and bad. The periods of code lockdown allowed us more time to play-test and fix bugs, which contributed greatly to the game's overall polish. On the other hand, it prevented us from adding many features that would have otherwise been included and it was detrimental to the morale of the team. We liked working on UNREAL TOURNAMENT, but it still felt like old technology to us. The world had seen the Unreal engine; we were ready to move on.

New Code, New Features

As it turned out, though, we had a lot of time to enhance the engine. UNREAL was before its time and a lot of the content and code was rushed by

the need to ship. With UNREAL TOURNAMENT, the team had a lot of time to use previously unexplored engine features. Erik de Neve's level-of-detail code ended up really speeding the game up, giving us room for beefier characters and more map decorations.

Early on we experimented with using 16 256x256 textures per player, but opted for three or four 256x256 pieces out of memory considerations. This quadrupled the detail available to our skin artists for the player models. Reserving one of the 256x256 textures for the head alone allowed us to mix and match body skins with heads, yielding a massive amount of customization with only a small amount of work. Another one of the 256x256 textures was reserved for team color bits, so that a player skin could encompass all five possible team colors (none, red, blue, yellow, green) without too much memory use.

Level design didn't stand still either. Changing from single-player to deathmatch-oriented design was refreshing for the designers, but not without its unique challenges. One issue was the task of balancing the number of "hardcore" maps with "theme" maps. A hardcore map focuses entirely on layout

and game play, while the overall style of the map comes second. Theme maps, on the other hand, focus on a unifying idea or look and build from that. For example, the Koos Galleon, designed by Pancho Eekels of Digital Extremes, is a large sailing ship. It's a very beautiful level, but focuses on the theme of being a ship more than being a deathmatch map.

The UNREAL TOURNAMENT team decided that mixing the two styles was the best approach. While most magazine reviews have expressed frustration at the theme-oriented maps, we didn't want to appeal to only the hardcore crowd. Including maps that were designed for their look and feel increases the game's interest to average players who aren't skilled enough at the game to benefit from hardcore designs. Realism through textures and architec-



ture is one of the Unreal engine's strengths and it was critical that we exploit that strength. Ultimately, we shipped UNREAL TOURNAMENT with somewhere around 45 to 50 maps, offering more than enough variety and replay value for everyone.

Another task we faced was choosing which of UNREAL's weapons to keep and which to ditch. UNREAL TOURNAMENT has two firing modes which makes designing a weapon like designing two weapons in one. UNREAL's stinger and dispersion pistol were not needed in UNREAL TOURNAMENT. Those weapons

were good in UNREAL, because a player needed to start with simple, weak weapons and build up. In UNREAL TOURNAMENT, all the weapons had to be equally effective and carefully balanced. A player good with the minigun needed to be lethal with it. A player good with the pulse gun needed to be lethal with it. Eventually we settled on the current load-out, but made quite a few gameplay changes to the weapons that stayed from UNREAL. Each weapon was also given a much more beefed-up look and sound.

An interesting little anecdote: GT started doing promotion for UNREAL TOURNAMENT before the new rocket launcher was finished. They produced a lot of marketing material with old screenshots showing the eightball launcher from UNREAL. If you look at the gold trophy used in the print ads, you'll see the characters at the top are holding eightballs, a weapon that isn't in UNREAL TOURNAMENT.

In the End, It All Worked Out

While the talents and devotion of individual team members created the content, the overall team spirit tied it together. UNREAL TOURNAMENT's design process was often reckless, but the game that resulted is nevertheless very polished and a hell of a lot of fun. The deathmatch-focused first-person



UNREAL TOURNAMENT used from three to four 256x256 textures per model. This allowed us to focus a lot of detail in the head and face area. Within the game a player can choose a skin and then swap through several different faces. This means players on a team can wear identical armor and clothing but have unique faces.

shooter doesn't need a story, dialogue, or scripted sequences, which are all features that more or less require an organized design. Had we applied our hodgepodge design approach to a more focused genre, we probably would not have had such a successful game. UNREAL TOURNAMENT should not be seen as a lesson in how to design a game, but as a lesson on how to organize a small team of developers.

What Went Right

1. SMART INTERNAL MARKETING TEAM. At the front of Epic's public relations were Mark Rein and Jay Wilbur. Their job was particularly difficult during the development of UNREAL TOURNAMENT. The media perceived us as impossible upstarts, taking an engine with terrible net-play and attempting to compete against id Software, the industry multiplayer champion. Both Mark and Jay fought hard to win over supporters in the online and magazine press. Mark made sure that the team stayed professional and that everyone was saying what he needed to be saying. Jay hunted down potential engine licensees, and helped establish a level of curiosity among the community and media.

UNREAL TOURNAMENT was able to garner significant magazine coverage because of the ongoing "QUAKE killer"

debates. Mark and Jay worked to turn the initially negative public response into something positive. While we felt that our game would definitely stand on its own, we had to ensure that the positives were being clearly broadcast. Epic was very careful to avoid mentioning QUAKE 3: ARENA whenever possible, keeping the focus solely on UNREAL TOURNAMENT's features and staying away from comparative previews. Most interviews and previews would ask us the inevitable "What about QUAKE 3?" question, to which we tried to answer with complete respect for id's project. Everyone knew that UNREAL TOURNAMENT

and QUAKE 3 would be pitted against each other. Mark and Jay established very early on that the competition would be friendly.

2. LIBERAL INTERNAL STRUCTURE, OPEN DESIGN DISCUSSION. The laid-back environment that both Epic and Digital Extremes fostered greatly enhanced the quality of UNREAL TOURNAMENT. Everyone was free to suggest or implement an idea. Programmers had as much design freedom as anyone else on the team. Cliff Bleszinski (Epic) and James Schmalz (DE) were the lead designers of their respective companies and served as content filters. They worked towards focusing the ideas put forth in the meetings. In addition, both of them contributed significantly to the final game content. James designed two of the player models and created many skins and faces, while Cliff designed many of the game's best maps.

Team members were allowed to come into work when they wanted and stay however long they felt like being there. The only requirement was that every member attend a weekly design and focus meeting. This system worked because Epic was very careful to hire mature, dedicated employees and the core development team was kept small. The open hours often saw team members working a 24-hour day, sleeping on a couch for six hours, and then working another 24-hour day.

In addition to fostering a hardcore work ethic, the system created a side-ways information flow. A programmer would go straight to the artist he needed something from, instead of through an art director. The fast communication allowed the programmers to stomp out bugs relatively quickly and the level designers to talk directly to the texture artists. An example of this was the single-player ladder system. Shane Caudle designed the art and I wrote the code. The fewer people we had to consult in order to complete the task meant a much faster turnaround. Everyone participated in giving the "coolness factor" thumbs-up or thumbs-down, but the actual development process was intentionally kept thin.

3. DIRECT COMMUNICATION WITH THE GAMING COMMUNITY. Nearly every Epic and Digital Extremes employee frequented message boards dedicated to the subject of UNREAL and UNREAL TOURNAMENT. The majority of Epic employees were drawn directly from the gaming community, either through mod projects or independent game work. Keeping in contact with the gaming community allowed Epic to focus on the target audience during the design process.

Beyond our direct communication with the UNREAL community, we also trolled QUAKE 3 message boards, reading the discussions of the fans of our lead competitor's game. Learning what people liked in a first-person shooter and why they liked it helped us change the marginal multiplayer experience in UNREAL to the much faster paced game play in UNREAL TOURNAMENT.

The gaming community can really help set the tone for your game. When UNREAL was released, the online community became extremely vocal and angry about the state of the net play. While most magazines had reported positive experiences with UNREAL's single-player mode (reflected in positive reviews), the media eventually came to reflect the cries of the hardcore gaming community. This was in part because the net play was poor, but also due to the fact that many members of the gaming media are themselves hardcore game players and visit those same message boards and community outlets.

We also learned that while the hardcore community is very vocal, it is also relatively small. Designing a game to



The characters in UNREAL TOURNAMENT were designed to be futuristic pit fighters. The selection of characters include ex-military specialists, criminals, and alien warriors such as the Necris Phayder Assassin pictured above.

appeal to that community alone is a critical mistake. Early in 1999 we started work on tutorials for each game type. The tutorials are far from definitive, but they did cover the basics of playing a 3D shooter. Testing on the parents and grandparents of team members demonstrated that the tutorials were useful for attracting and keeping new players.

This community-mindedness greatly contributed to the quality and completeness of UNREAL TOURNAMENT. We

had a very good idea of what players wanted. As I mentioned earlier, we often posted controversial design questions on public message boards to gauge public reaction. The results of these polls were taken into consideration when the feature in question was implemented.

4. STRONGLY OBJECT-ORIENTED ENGINE DESIGN. The Unreal Tournament engine's strong object-oriented design makes it extremely modular. This modularity allowed our programmers to make massive changes to parts of the game without affecting other features. Each subsystem is connected to other subsystems through a clearly defined interface, and platform-specific code is consigned to separate libraries. Creating the Linux port, for example, was simply a matter of rewriting an input and sound device and writing a Linux version of the platform-specific library behavior.

Throughout UNREAL TOURNAMENT's development, Tim Sweeney and Steve Polge worked on improving the networking code. The modularity of the engine meant that their work didn't disturb anyone else's work. Some features, such as Jack's decal system, were added very late in the project. The decal system added a lot of depth and feedback to the game, and took less than a week to get working and fully debugged. Erik de Neve's mesh level-of-detail code touched only a handful of source files.

This ease of use is also reflected in the engine's scripting language, UnrealScript. Calling it a scripting language is a misnomer; it's actually a lot like Java. Weapons, pickups, level events, AI nodes, and other world actors are all independent objects. A weapon can be added to the game without touching any source files but the new object definition. This highly extensible language meant that each programmer could add extensive new game-play features with a very limited set of potential side effects. In the end, 90 percent of UNREAL TOURNAMENT's game-play code was written in UnrealScript.

The Unreal Tournament engine's modular package system coupled with UnrealEd makes the game a mod-creation system out of the box. We designed a lot of our code with amateur extension in mind. Everyone at



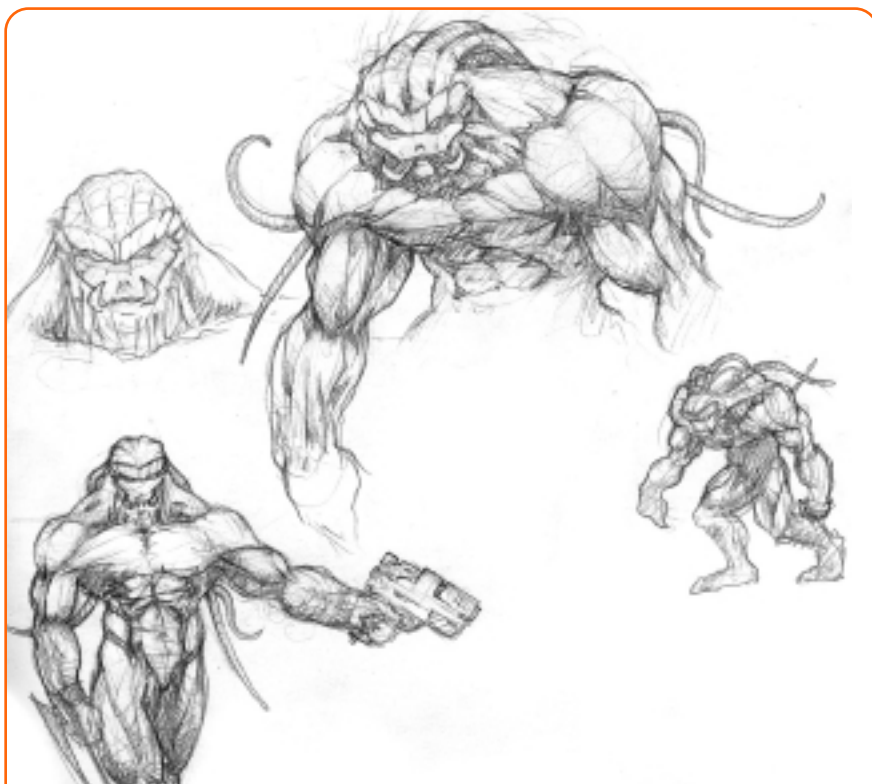
Epic recognized the value of the mod community and we wanted to make the game attractive to new artists and programmers. Constructing game code in this way made it much easier for us to prototype our own new features. Early UT weapons and pickups were child objects of UNREAL. The two games can easily coexist even now.

5. GOOD TIMING. As I said earlier, UNREAL TOURNAMENT was developed in the same time period as id Software's QUAKE 3: ARENA. The two games promised to be of the same genre and the two companies were known for a high level of competition. While we tried to avoid the "QUAKE 3 vs. UT" comparisons, they ultimately worked in our favor. The high level of public interest in the new engine war greatly increased our visibility. Magazines and web sites often posted split previews instead of focusing on one game in particular. Interviews with id employees would always lead to UNREAL TOURNAMENT questions and vice versa.

UNREAL TOURNAMENT took almost exactly a year and a half to develop, giving the team a lot of time to pack in features. We didn't have to focus on writing an engine from scratch, so we were free to focus entirely on improvements. At this point, we've released three patches for UNREAL TOURNAMENT that have solved a handful of relatively minor problems. The team has had a lot of time available to spend on adding even more features to the game since its release, instead of fixing outstanding issues. By the time this article hits the stands, we'll have released our first bonus pack: a free collection of new models, maps, and game-enhancing features.

What Went Wrong

1. BAD TIMING. Many aspects of the game's timing worked against us. While the QUAKE 3 vs. UT hype increased our exposure, it also set a very hard deadline for completion. It was critical that we complete the game before QUAKE 3 was released. The media advantage belonged to id and we believed that if UNREAL TOURNAMENT launched after QUAKE 3, we would be forgotten in the storm. At the same time, however, we were



After the release of UNREAL TOURNAMENT, the Epic team started working on a free bonus pack containing additional models. These are concept drawings of the Skaarj Warrior model for the pack.



Every weapon in UNREAL TOURNAMENT has two distinct firing modes. This made designing and balancing each weapon twice as complex as a normal first-person shooter.



The UNREAL TOURNAMENT development team felt that several of Unreal's weapons were a lot of fun. Here is a bot carrying the Shock Rifle, an updated version of UNREAL's ASMD.

caught up in grueling contract renegotiations with GT Interactive. We did not want to deliver the completed game until we knew the contract would work in our favor. Many times during the development of the game we were promised that a resolution to the contract issue was close at hand. The team would race to reach a point where the game could be shipped,

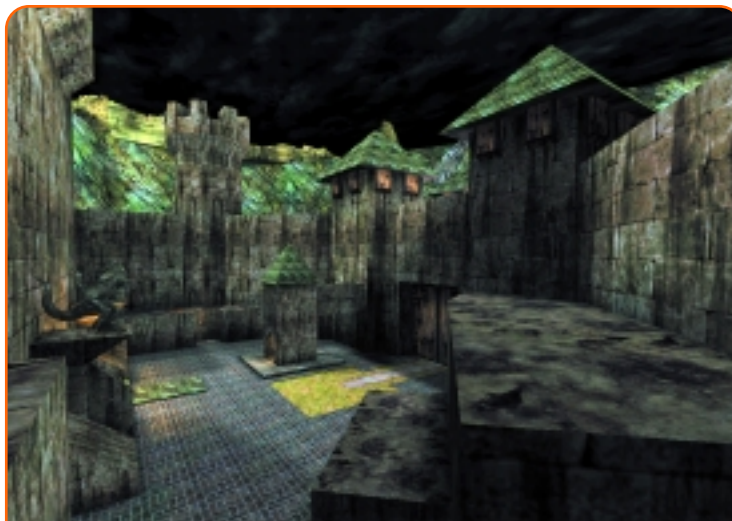
only to have negotiations drag on.

The gold master was delivered to GT days after a final contract was agreed upon. Unfortunately, the game hit shelves in November, pushing us very close to QUAKE 3's release date. While UNREAL TOURNAMENT often performed better than QUAKE 3 in reviews, we believe that sales would have been much higher still had we released in

October. Word-of-mouth is a powerful force and the extra month would have given us time to build a larger community before Christmas.

2. NO CENTRAL DESIGN DOCUMENT. While I am a big supporter of open, cabal-style design, I have to stop and wonder how UNREAL TOURNAMENT would have turned out had we a strong initial design. It's quite possible that the game's weaker elements would have been much stronger if we had put together some concept art and focus material. In reviews, we have been criticized for not having enough variation in characters. If UNREAL TOURNAMENT had had a library of concept art to draw from, we might have had more interesting alien warriors. The story is more or less nonexistent in UNREAL TOURNAMENT, but at times we considered having in-game cutscenes as rewards for a player's progress. The idea was dumped, but a design document might have made it easier to visualize those scenes.

I suppose this isn't really a "what went wrong." It's simply more of a "what we should have done." I think it's important to think about the game in that light. UNREAL TOURNAMENT is a very fun game with a lot of features packed into a short amount of development time. Those features were largely added through spur-of-the-moment decisions. A more unified approach to design would have allowed us to construct features that play on features, or even think of ideas we didn't have the perspective to realize. Epic will always be a very open, liberal company when it comes to the design process. If we develop a design document, we'll use it with the understanding that it can be modified at any time. That having been said, I think there is a definite positive argument for having some sort of central guide to everyone's ideas. Having the ability to sit down and look over the big picture is very valuable.



In the Assault game type, players have to enter a heavily defended base and complete map-specific objectives to win. Assault was the most difficult UNREAL TOURNAMENT game type to design, balance, and play-test.



Steve Polge, our AI and game play programmer, made the bots understand the unique advantages and disadvantages of each weapon. Here a bot is moving in very close to use the powerful Flak Cannon.

3. CO-DEVELOPMENT ACROSS TWO COUNTRIES. Epic Games and Digital Extremes co-developed UNREAL TOURNAMENT. The Digital Extremes team was located in Canada and Epic was located in the U.S. Epic supplied the programming team and a large group of content designers. Digital Extremes provided level designers, a sound guy, and texture artists. James Schmalz, the high-up man at Digital Extremes, contributed two of the game's player models and a lot of art. This co-development worked well for the most part, but near the end of the project it became very difficult.

During UNREAL, Epic team members flew to Canada to work at Digital

Extremes' offices. With UNREAL TOURNAMENT, it became Digital Extremes' turn to do the traveling. Unfortunately, flying and driving back and forth every couple of weeks is a very draining experience. Many of the Digital Extremes team members spent several weeks away from their wives and girlfriends. Near the end of the project, they grew increasingly frustrated with the situation. To compound this problem further, Digital Extremes and Epic were attempting an expensive merger. As UNREAL TOURNAMENT came to a close, it became clear that the merger would not happen. It was prohibitive-

ly expensive for a small company to move across the border. Many Digital Extremes team members already had apartments and plans for living in Raleigh, and the news of the terminated merger process was devastating.

Much to Digital Extremes's credit, the company quickly recovered and moved to its backup plan of developing its own game with the Unreal Tournament engine. Nonetheless, the process of co-developing the game had taken its toll on everyone. The ups and downs of the merger process had a negative effect on team morale. Had the co-development happened between companies more closely situated, it would not have been a problem.

4. NOT ENOUGH ARTISTS. On the content side, UNREAL TOURNAMENT was held back by the number of available artists. Epic's artist, Shane Caudle, is a supreme Jack-of-all-trades, creating skins, models, and levels of the highest quality. He spent most of his time working on new player models and skins for those models. Digital Extremes brought a few texture artists to the table, but not enough to create the huge libraries of new textures needed for the game. In order to supplement the skin and texture production, Epic turned to contract artist Steve Garofalo.

Even with the additional help from external sources, the team was unable to produce enough new textures. Level

designers who wanted custom textures for their maps had to make do with their own texturing ability. While the final texture and level count in UNREAL TOURNAMENT is quite high, the levels would have been much more impressive had the team been able to act with full freedom. Since the completion of UNREAL TOURNAMENT, Epic has hired both Steve Garofalo and John Mueller to strengthen the art team for future projects.

5. VISUAL BASIC EDITOR INTERFACE. The Unreal Tournament engine uses UnrealEd as its level design and content management tool. For several years, UnrealEd has used a windowing interface written in Visual Basic. The VB code is fragile and very old. Add to this the fact that nobody at Epic except Tim Sweeney knows or cares about VB, and you have a level design team that is stuck with a tool that's not easily updated.

Several interface bugs have plagued UnrealEd for some time and nobody on the team had the time or inclination to fix them. If we had a more easily extensible tool, the team would have been able to add extra features to the editor for level designers to use. As it stood, the editor was considered "off limits" for new features.

For our next few projects we will most likely use a new C++ editor that Legend Entertainment developed. For UNREAL TOURNAMENT, however, we simply didn't have the time to work on a new editor. Fortunately, our time spent using UnrealEd taught us the dos and don'ts of tool design.

Where We Go from Here

The things that went wrong are, all in all, much less significant than what went right. UNREAL TOURNAMENT could have benefited from a more focused initial design and a more solid ship date, but it turned out to be very polished and a lot of fun. Many of the factors that worked in our favor, like timing, also worked against us to some extent. "What went wrong" is a good way of looking at what we could have done to make UNREAL TOURNAMENT even better.

Epic has developed some pretty clear plans of where we want to go from here. We've been working on free con-



Epic hired several extremely skilled contractors to assist with art production. This is an extremely detailed female skin by Steve Garofalo. In February, Epic hired Steve as a full-time team member.

tent to release to support UNREAL TOURNAMENT. We are also looking into doing some kind of Playstation 2 version of the game. After that, we want to focus on an entirely new engine technology for the PC. In the short term, Jack Porter is working on his terrain system and Erik de Neve is putting the finishing touches on the skeletal animation system. Tim Sweeney has been developing an entirely new programming language to support the next engine, with some very powerful features such as parameterized functions.

UNREAL TOURNAMENT served as a good learning tool for the team. We have a good idea of what processes we need to adopt to produce larger, more story-driven games in the future. We see UNREAL TOURNAMENT as a good lesson in how to organize a team and produce a game in a short amount of time. The team has grown socially, and everyone is much more experienced in the process of game development. We feel very prepared to face the upcoming challenges and, hopefully, to continue to be seen as innovators in the industry. ■



Take Me to Your Leader...

Cabal. Committees. Design by attrition. Mean anything to you? Bring on a migraine from the experience you had on your last project? Make you cringe while

thinking about how work went today on your current project? I bet.

You see, making computer games today is not rocket science. It takes hard work, talent, and dedication. It also takes leadership. That's right, the dreaded *l*-word. Everyone seems to want it, no one seems to do it right, and the ones most qualified to be in the captain's chair are usually sporting a tight red shirt, beaming down with the next away-team. In order to succeed, a game has to have leadership throughout the course of its development. Sure there are exceptions, but even in cases where it seems a single person isn't at the helm, there really is someone who had a vision and maintained it throughout the development process. (He's the guy behind the curtain.)

I didn't pick up a mouse until I was 27 and even though I've been making art for computer games for almost nine years, I'm still astounded by the lack of emphasis many companies place on structure in the development process. I know why, though. It's that leadership thing. Maybe I'm just more acutely aware of it having served six years in the Air Force, but it bugs me greatly that in the time I spend pushing pixels and vertices, concepts like "chain of

command" are deemed too militaristic and constrictive for our industry.

What's wrong with a little order? What's wrong with a little discipline? What's wrong with knowing without a doubt who the right guy to turn to is? What's wrong with making sure the person in charge of the vision for the game has at least the basic social skills needed to communicate to the rest of the team? Nothing. But lapses in leadership hap-



pen all the time. Whether it's a case of the wrong person leading the team or no one being willing to make a decision without ratification by a ten-man committee, more often than not games are made in a storm: a storm of inclement social and team dynamics. Such bad weather can always be traced to the top. Think about it. When the director of a project is passionate, committed, communicative, and willing to be flexible while creating his or her opus, guess what happens? It's infectious. All the

other team members get those same symptoms leading to one hell of a product. But before we get into the importance of having a good captain at the helm, let's talk about the ship.

Without the right people, nothing else matters when making a game today. Money doesn't matter. Swanky location doesn't matter. Nothing matters. This crew has to be a group of talented, hard-working, passionate individuals working together as a team. As individuals, each person must be more than just competent. They have the ability, and more importantly, the desire to solve problems. They're professionals in their given areas of expertise and proud of their work. Eager to impress, they nonetheless understand

the value of teamwork and clear communication. They don't have to have one, but an occasional pat on the back goes a long way, even if they don't always show it.

Feedback and direction from the project leader sustains the members of the team as they plod through the long, hard trek that is making computer games today.

Not knowing if they're doing a good job, a bad job, or even an adequate job will hurt morale. Not knowing what they're supposed to be doing and whether what they're doing is correct will quickly herd a team into disaster. This is where the leader comes in, not only through carefully monitoring the morale and status of the team, but by successfully communicating his or her vision to them.

The importance of having a coherent

CONTINUED ON PAGE 63

Paul Steed is a little-known, shy artist-type serving as the modeling and animation department at id Software. He often laments his art director days at other companies where he spent many fruitful hours delegating responsibilities to others. Now he just tries his best to keep up with the rest of the pack occasionally writing a tutorial or enlightening editorial.

CONTINUED FROM PAGE 64

vision for a game idea can't be overstated. The project leader has to hold on to the vision, know it, and convince everyone else of his or her commitment to it, so that everyone else can in turn embrace and become excited about it. This is most evident through a well-written (and evolving when necessary) game design document. An effective design document gives form to the vision and adds to the team's comfort level as they settle in for the long haul. Without this bible in hand, it's impossible to maintain and communicate the vision for the game to the rest of the team. But the term "bible" should be used loosely.

The key to a good game design document is lucidity and flexibility. It has to identify important aspects and concepts of the design clearly and quickly so that they can be translated into workload. It is a road map and a starting point for all phases of preproduction and production. Don't make a phone-book-sized tome and expect everyone (or anyone) to read it. A well-written design document exists in three iterations: one page, ten pages, and one hundred pages. All will read the one-page version, most will read the ten-page version, and few if any will read the one-hundred-page version. It should be visually interesting, not bogged down by flowery prose. Revisions to the document should be followed effortlessly. Flexibility in the design document means that changes can be incorporated if existing ideas aren't working out or can't be executed. Clinging tenaciously to an aspect of the design when it's apparent it won't work is bad. It's a sign of pride over reason and a sign of bad leadership.

So what about the leader himself? What does it really take to be the captain of the starship? Good organizational skills, good communication skills, and good people skills are a must. Expressing endless passion and unrelenting drive helps as well. When people sense your commitment, they become committed as well. If they think you have your act together then they'll strive to keep their act together and execute the plan you've devised with aplomb. Of course, a dictatorship doesn't inspire any sort of devotion or esprit de corps. Barking orders and disregarding what others have to say won't work. Having confidence in your leadership ability means just that — throwing temper tantrums when things don't go your way is the surest and quickest way to turn off your team. Resolving sticking points during the development process by demonstrating tact and diplomacy isn't the easiest thing to do, but it is the most effective.

Does leadership necessarily have to rest on the shoulders of one person? Yes, it does. Leadership by committee just doesn't work, especially if the team is small. Too many chiefs, not enough Indians; too many cooks in the kitchen — whatever saying comes to mind can be applied here. Going to one person for answers saves time and confusion, allowing for (gasp!) projects to be done on time. By contrast, committees are invariably bogged down by politics and wasteful bickering. In our industry, too often the decision-making process is way too complicated and involves way too many people. That's not to say democracy should be thrown out the window, quite the opposite. If a leader is on the ball, he

or she already knows what the general consensus on any issue is. Leaders should never be afraid to make decisions and deal with the consequences.

Finally there's commitment: doing whatever it takes to get the job done. Effective, confident leadership creates an environment where synergy and camaraderie grow through the understanding that effort won't be wasted and will be appreciated. But in the end, everyone, not just the leader of the team, has to commit unflinchingly to their abilities and the game's premise in order to celebrate a successful finished result.

At the heart of it is this: Good leadership is about conveying your vision to the rest of the team *effectively*. A leader is not a shy person. A leader is not one of the socially challenged. Having embarked upon the arduous journey that becomes the game development process, a team needs to know that a competent and skilled captain is at the helm. All the team members need to know well beforehand what is expected of them and what they need to accomplish to support the project's various milestones. The leader of the project must provide direction so that members of the team know at all times that what they are doing is right and what they need to be doing in the future. When communication falters, so does the project. The team will flounder, turn on itself, dissolve, or complete a project that results in a marginal game at best.

Successful games don't happen by accident. A good leader commits, motivates, communicates, and in the end inspires the rest of the team to stick it out and care about what they do. A good leader *leads*. ■

