



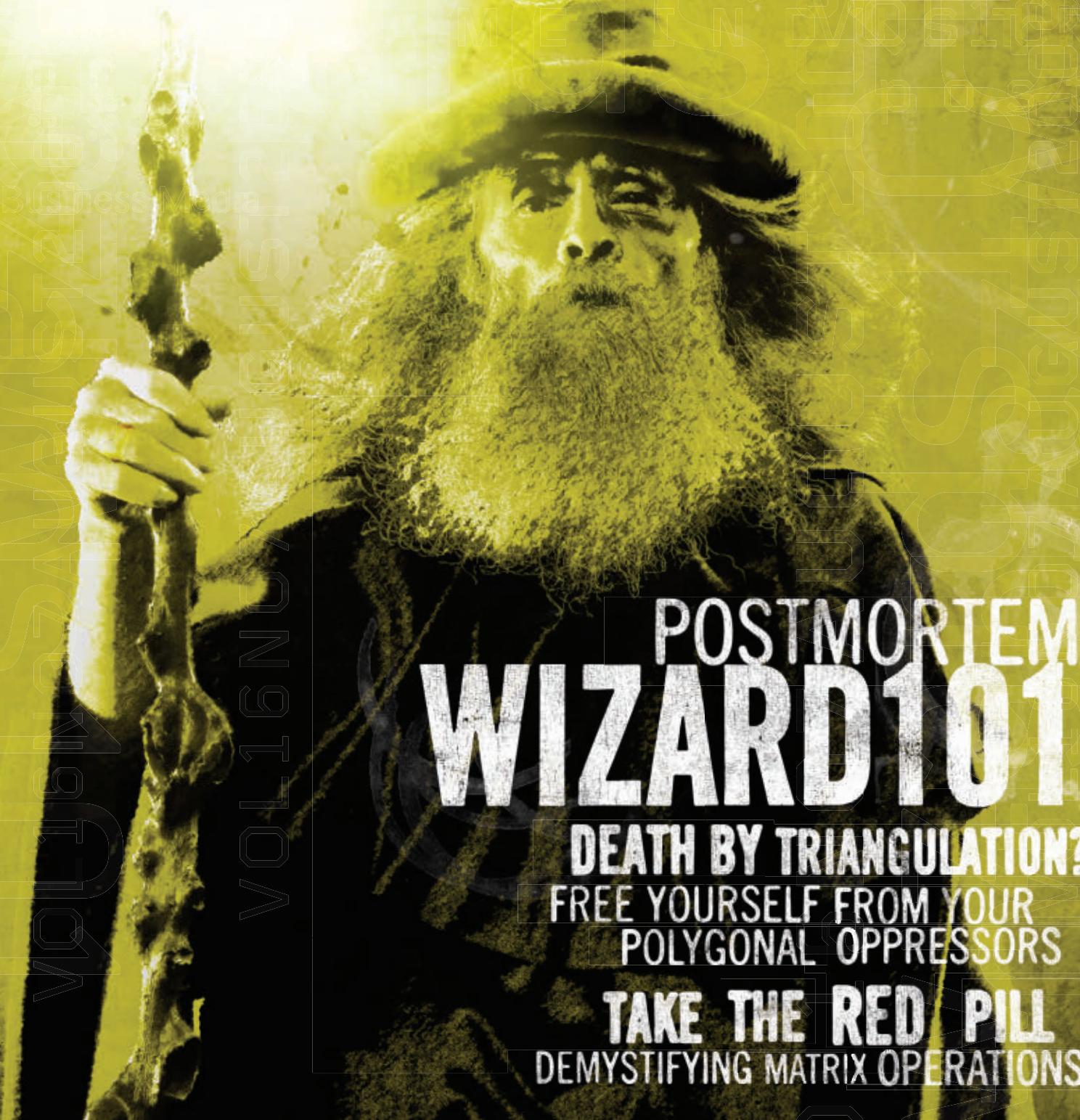
United Business Media

TOOL BOX: PIXELACTIVE'S CITYSCAPE 1.7

VOL16NO8SEPTEMBER2009

gamedeveloper

THE LEADING GAME INDUSTRY MAGAZINE



POSTMORTEM **WIZARD101**

DEATH BY TRIANGULATION?

FREE YOURSELF FROM YOUR
POLYGONAL OPPRESSORS

TAKE THE RED PILL

DEMYSTIFYING MATRIX OPERATIONS

EYETRONICS 3D

**3D SCANNING
SOLUTIONS**



CUSTOMERS INCLUDE:
ROCKSTAR
ELECTRONIC ARTS
UBISOFT
SONY
2K INTERACTIVE
MIDWAY, THQ, etc...



E Y E T R O N I C S
www.eyetronics.com



800.205.9808.EU +32.16.298309



POSTMORTEM

22 KINGSISLE ENTERTAINMENT'S WIZARD101

WIZARD101 is an MMO with a unique business model. The game not only offers the free-to-play (with microtransactions) model, but also a subscription alongside it. Texas-based KingsIsle shares with us the very honest ups and downs of developing a new MMO, from the boons of free trials to botched UI. *By James Nance*

FEATURES

7 ALTERNATIVE GEOMETRY

Polygons rule the game development roost, there's no question of that. But that doesn't mean voxels, NURBS, and other methods of achieving non-polygonal geometry have nothing to offer. This article investigates the state of the art and how you can try to implement non-polygonal 3D into your game, or at least your pipeline.

By Bijan Forutanpour

15 DEMYSTIFYING MATRIX LAYOUTS

Matrices tend to live in shaders, physics systems, and other time-critical portions of games. These important structures are often difficult to optimize and design, so here the authors show best practices for getting the most out of your matrix operations on modern systems, both console and PC. *By Jelle Van Der Beek and Arjan Janssen*

DEPARTMENTS

- 2 GAME PLAN *By Brandon Sheffield*
Clone Versus Genre

[EDITORIAL]

- 4 HEADS UP DISPLAY
Vintage computing tools, *Game Developer* advisory board update, free *Game Career Guide*, and more.

[NEWS]

- 36 TOOL BOX *By Marc-André Guindon*
PixelActive's CityScape 1.7

[REVIEW]

- 39 PIXEL PUSHER *By Steve Theodore*
Jumping to Occlusions

[ART]

- 43 THE INNER PRODUCT *By Noel Llopis*
Data-Oriented Design

[PROGRAMMING]

- 47 DESIGN OF THE TIMES *By Damion Schubert*
Resonance

[DESIGN]

- 50 AURAL FIXATION *By Rob Bridgett*
Stereo Downmixing

[SOUND]

- 56 ARRESTED DEVELOPMENT *By Matthew Wasteland*
My Story Isn't Dumb

[HUMOR]



CLONE VERSUS GENRE

WHEN ART IMITATES ART

WE OFTEN TALK ABOUT "CLONES" OF GAMES, OR COPIES of ideas—but at a certain point, if an idea is copied, expanded, and massaged enough times, nobody says clone anymore. When does an idea become large enough to lose its own identity, and flourish into a full-blown genre? Perhaps it's better said that the idea gains a new identity, one that's more generally applicable.

Most recently, we stopped referring to "TOWER DEFENSE clones," and began discussing tower defense games as an entire genre, which has since been expanded to include a number of variations, by companies as diverse as Square Enix and PopCap. "DOOM clones" stopped appearing around the time QUAKE came out, and we started to deal with the FPS as a genre. BEJEWELED, itself inspired by earlier matching puzzlers like COLUMNS, is no longer copied per se, we now have the "match-three" genre, which includes the robust PUZZLEQUEST in its ranks.

WILL THE ORIGINAL CONCEPT PLEASE STAND UP?

» The question occurs to me as I work on an iPhone game that's heavily inspired by one specific existing game. This original game has had one or two "clones" already, but they're not well known enough to have become pervasive. So how do I know if I've expanded my own version of the concept enough to safely claim I'm not making a "clone?"

Let's consider some existing examples. PopCap's ASTRO POP successfully iterates on a concept popularized by the Data East game MAGICAL DROP (grab colored balls from a well, form groups of three to remove them) by adding powerups and changing the setting to outer space. MAGICAL DROP's creator Data East was no longer around to make another entry in the series, so who would begrudge PopCap's renewal of the still-enjoyable concept? One step further, there's the mobile game puzzler CRITTER CRUNCH, which looks familiar, but changes the chaining structure to such an extent that it feels like a totally different game.

Then there's ZUMA—also from PopCap—versus PUZZ LOOP from Mitchell Corp. In this case, only the scenario was changed, the gameplay was not significantly altered or upgraded from the two existing PUZZ LOOP titles. This is much more likely to be labeled a clone, because while the platform has changed from the original arcade game to the PC (and other devices), the game feels very similar. The game I'm working on may well be more similar to the original

than it is different. Though the platform, characters, and controls will have all changed significantly, the core concept is still quite reminiscent of the original. So how will my game be labeled?

I AM NOT A REPLICANT

» So where is the line drawn? How much is "enough" concept evolution to make a game a step forward within a genre shard versus being labeled a clone? I say "shard," because as games evolve, genres themselves become more granularly defined. Though TETRIS, SOKOBAN, SUDOKU, and BEJEWELED are all puzzle games, you wouldn't paint them with the same brush. Falling-block puzzle, action puzzle, brain puzzle, and match-three are the more correct terms.

SOKOBAN (a game in which you push boxes into the correct areas, trying not to trap yourself) is an interesting one, because most games based on its

concept are in fact true clones, to the point that they use the SOKOBAN name in their titles, and reuse the puzzle maps from the original. Variants like CHIP'S CHALLENGE and BOULDER DASH seem to escape the "clone" distinction by adding different goals and scenarios.

Then we have genre-defining games like GRAND THEFT AUTO's 3D iterations. Certainly open world games had existed previously, and sandbox games were used to describe something like SIMCITY rather than GTA. But after GTA III, any game with any similar elements was called "GTA-like," and to some extent that's still true. Perhaps this is because we heard a lot of publishers saying "Make it more like GTA" at the time. But perhaps it's also partially the fault of the press for wanting to compare everything to a specific gold standard.

In the case of the game I'm working on, the original concept is simple. I think that's the major factor here. It seems that the simpler the core concept, or the fewer core concepts exist within one game, the more likely a successor is to be labeled a clone. But of course, we don't call MADDEN and NCAA clones of each other, nor clones of the sport on which they're based, even though they follow the original core concept of American football to the letter. Ultimately, it's a question of semantics, evolution, and feeling. If a game simply feels too much like another, it's going to get called a clone. If it works to distinguish itself within a solid concept, every game like it will work in concert to create a new genre.

—Brandon Sheffield



Data East's MAGICAL DROP and Capybara Games' CRITTER CRUNCH.

Think Services, 600 Harrison St., 6th Fl.,
San Francisco, CA 94107
t: 415.942.6000 f: 415.942.6090

SUBSCRIPTION SERVICES

FOR INFORMATION, ORDER QUESTIONS, AND ADDRESS CHANGES

t: 800.250.2429 f: 847.763.9606
e: gamedeveloper@halldata.com

EDITORIAL

PUBLISHER Simon Carless | scarless@gdmag.com

EDITOR-IN-CHIEF Brandon Sheffield | bsheffield@gdmag.com

PRODUCTION EDITOR Jeffrey Fleming | jfleming@gdmag.com

ART DIRECTOR Joseph Mitch | jmitch@gdmag.com

SENIOR CONTRIBUTING EDITOR Jill Duffy | jduffy@gdmag.com

CONTRIBUTING EDITORS Jesse Harlin | jharlin@gdmag.com

Steve Theodore | sttheodore@gdmag.com

Noel Llopis | nllopis@gdmag.com

Soren Johnson | sjohnson@gdmag.com

Damion Schubert | dschubert@gdmag.com

ADVISORY BOARD

Hal Barwood Designer-at-Large

Mick West Independent

Brad Bulkley Neversoft

Clinton Keith High Moon Studios

Bijan Forutanpour Sony Online Entertainment

Mark DeLoura Independent

Carey Chico Pandemic Studios

ADVERTISING SALES

GLOBAL SALES DIRECTOR

Aaron Murawski e: amurawski@think-services.com
t: 415.942.6227

MEDIA ACCOUNT MANAGER

John Malik Watson e: jmwatson@think-services.com
t: 415.947.6224

GLOBAL ACCOUNT MANAGER, EDUCATION AND RECRUITMENT

Gina Gross e: ggross@think-services.com
t: 415.947.6241

COORDINATOR, EDUCATION AND RECRUITMENT

Rafael Vallin e: rvallin@think-services.com
t: 415.947.6223

ADVERTISING PRODUCTION

PRODUCTION MANAGER

Robert Steigleider e: rsteigleider@ubm-us.com

REPRINTS

WRIGHT'S REPRINTS

Ryan Pratt e: rpratt@wrightsreprints.com
t: 877.652.5295

THINK SERVICES

CEO THINK SERVICES Philip Chapnick

GROUP DIRECTOR Kathy Schoback

CREATIVE DIRECTOR Cliff Scors

AUDIENCE DEVELOPMENT

GROUP DIRECTOR Kathy Henry

e: khenry@techinsights.com

DIRECTOR Kristi Cunningham

e: kcunningham@techinsights.com

LIST RENTAL Merit Direct LLC t: 914.368.1000

MARKETING

SERVICES MARKETING COORDINATOR Laura Robison
e: lrobison@think-services.com

UBM TECHNOLOGY MANAGEMENT

CHIEF EXECUTIVE OFFICER David Levin

CHIEF OPERATING OFFICER Scott Mozarsky

CHIEF FINANCIAL OFFICER David Wein

CHIEF INFORMATION OFFICER Kevin Prinz

CORPORATE SENIOR VP SALES Anne Marie Miller

SENIOR VP, STRATEGIC DEV. AND BUSINESSADMIN. Pat Nohilly

SENIOR VP, MANUFACTURING Marie Myers



United Business Media

WWW.CMPGAME.COM

STOP THE ZOMBIES

GIVE 'EM BRAIN

We are tired of stupid zombies populating games.
Help us and **give 'em brain**. Sign up **NOW** for **FREE**
and download the **NEW** xaitment BrainPack SDKs

Set up and control complex game logic in a few steps. Generate your perfect
navigation mesh with a single click. Create realistic behavior in no time.

Join the xaitment community now and turn your idea into a stunning prototype without
paying any license cost. By signing up for free, you'll receive our complete modular AI Engine
plus our world-class support. Experience the future of next gen game technology and work
with the smartest AI technology available.

Contact xaitment today for more information about the BrainPack Program under
brainpack@xaitment.com or visit our website www.xaitment.com



*GIVE 'EM
BRAIN*



xaitMENT
INTELLIGENCE ENGINEERED



MODERN TOOLS FOR VINTAGE COMPUTER DEVELOPERS

Classic 8-bit computers of the 1970s and 80s remind us of a much simpler time, when game development usually meant one programmer and a home computer instead of a team of hundreds and a colossal (but creatively stifling) budget. Luckily for modern developers, if you're interested in replicating that warm and fuzzy experience on a vintage machine, you're no longer bound by the primitive development tools of the past. For example, coding can usually be done in a comfortable, modern PC environment and transferred to older machines using modern technology. Listed below is only a tiny fraction of the tools available for the "modern vintage hacker." There's a lot more out there (not just for the platforms listed), so consider this a jumping-off point into the world of retrocomputing. —Benj Edwards, vintagecomputing.com

APPLE II

Designed by Steve Wozniak and released in 1977, the Apple II pioneered the mass-produced, preassembled computer concept and started an enormous following that continues to this day. While limited in graphical and sound capabilities compared to later 8-bit machines, the Apple II's open nature allowed programmers and hardware hackers alike to squeeze the most out of the system, a practice they continue to this day.

CFFA

<http://dreher.net/?s=projects/CFForAppleII&c=projects/CFForAppleII/main.php>
Price: TBA
» With 1970s-era floppies and disk drives growing more unreliable with each passing year, developing new software for an old machine can seem like a risky proposition. That's why the

wonders of flash storage technology have taken the retrocomputing community by storm in recent years, and the Apple II is no exception.

The CFFA is a plug-in board for the Apple II series that allows users to store data on CompactFlash cards or IDE hard disks. Through its CompactFlash support, it allows relatively easy data transfer between a modern PC and an Apple II.

The original CFFA is currently sold out, but its creator is working on an improved version called the CFFA3000 that may become available soon.

COMMODORE 64

As the best-selling computer model of all time, there's no doubt that many game developers today got their start on Commodore's inexpensive but powerful wunderkind. The Commodore 64, released in 1982, shipped with 64KB of RAM, impressive color graphics capabilities, and the world-

famous SID synthesizer sound chip that remains highly coveted by "chiptune" artists today.

MMC Replay

www.c64-wiki.com/index.php/MMC_Replay
Price: About ?9 Euros ([MMC Replay](#)) or 50 Euros ([RR-Net](#))

» The MMC Replay combines an impressive number of functions into one standard-sized Commodore 64 cartridge. It simulates two C64 disk drives by loading floppy disk image files off of Secure Digital (SD) flash storage media. It also contains a hardware clone of the classic Action Replay cartridge which includes development tools, game cheating functions, and the ability to dump the entire C64's memory contents to disk. Even better, one can buy an add-on RR-Net Ethernet module to add TCP/IP access to the C64.



into a .prg binary file that can be executed on a real or emulated C64. This provides an excellent environment for beginners who might not be familiar with classic C64 development tools.

ATARI 8-BIT

The Atari 8-bit computer line, originally intended to be a follow-up game console to the Atari 2600, emerged in 1979 with the release of the Atari 400 and 800. The original models shipped with 8KB of RAM, but the 800 could later be expanded to 48KB. Both included impressive sound and sprite-handling abilities for the time. Over the next ten years, Atari refreshed the 8-bit line with newer models like the 1200XL, 800XL, 130XE, and the XEGS.

Atarimax "Maxflash" Flash MultiCarts

www.atarimax.com/flashcart/documentation
Price: \$24.99 (1Mbit) or \$39.99 (8Mbit)

» The Atarimax flash multicarts are user-programmable cartridges that use built-in flash media to store multiple Atari 8-bit programs, usually games. Maxflash fans typically provide sets of Atari games that one can easily place on the cartridge, which is programmed via software executed on Atari computer itself. Even better, with the right tools (provided by Atarimax) and the proper know-how, one can also flash one's own homebrew Atari projects onto the cartridge.



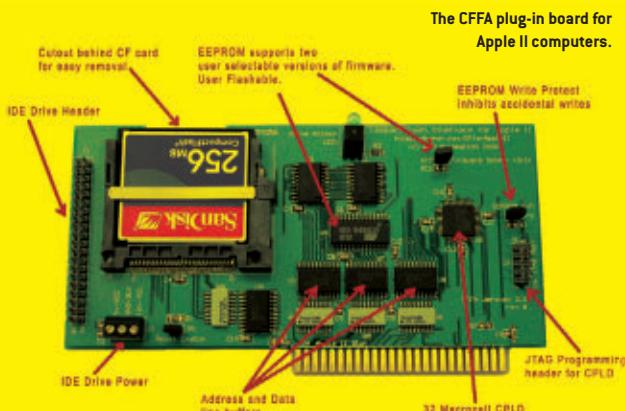
Atarimax SIO2PC Universal Interface.

C64prgGen

www.ajordison.co.uk

Price: Free

C64prgGen (short for "Commodore 64 Program Generator") provides a modern Windows environment in which programmers can type BASIC or machine language programs. The software will then convert the program



Atarimax SIO2PC Universal Interface

www.atarimax.com/usbcart/documentation

Price: \$59.99 (USB) or \$29.99 (Serial)

» An essential tool for any modern Atari 8-bit developer, the SIO2PC interface allows transfer of data between an Atari 8-bit computer and a modern Windows or Mac machine with a serial or USB port. ("SIO" was Atari's name for its 8-bit serial-based peripheral bus.) Better yet, it allows a PC to act as a disk drive emulator for an Atari 8-bit machine, making storing and loading software a snap.

COMMODORE VIC-20

Commodore conceived the VIC-20 [released in 1980] as a low cost home computer, and it shows. The VIC-20 included a mere 5KB of RAM and could only generate a 22-column by 23-line text display. Despite this, the VIC-20 sold by the millions due to its low retail price and is fondly recalled by many who had their first programming experiences on the endearingly primitive machine.



Mega-Cart Multi-Cartridge

www.mega-cart.com

Price: \$99.95

» Aside from containing every VIC-20 cartridge game ever released (and more) in a single cartridge, the Mega-Cart also includes built-in 32K RAM expansion and many software tools for VIC-20 programmers. The only downside is its very high price.

MULTI-SYSTEM

The following tools work with multiple computer platforms.

cc65

www.cc65.org

Price: Free

» cc65 is a freeware C cross compiler for computers that use the MOS 6502 microprocessor and its derivatives.



The cc65 compiler/assembler itself resides on a host system running Windows or Linux. From a C program written in a text editor of your choice, it produces binaries executable on 6502-based target systems like the Apple II, Atari 800, Commodore 64, and even the Nintendo Entertainment System.

8-Bit Baby

www.protovision-online.de/catalog

Price: 8 Euros

» The 8-Bit Baby is a hardware development breadboard for multiple 8-bit computer systems. It's unique in that it includes four different built-in card edge connectors that plug into slots on the Commodore 64, VIC-20, Plus/4, and the Apple II. If you've ever wanted to build a novel C64-to-toaster interface, here's your chance.

GAME DEVELOPER'S CAREER GUIDE AVAILABLE

GAME DEVELOPER'S ANNUAL CAREER GUIDE

issue, which is oriented toward students and prospective game developers, does not ship to subscribers, who should already be familiar with most of the introductory content.

However, for those entry-level readers who would like to get ahead in the industry, or those who would simply like to see what horrible lies we're trying to propagate among today's youth, the editors are pleased to announce that the entire Game Career Guide can be viewed online in interactive PDF form, at no cost.

The special supplement this year focused on simply going ahead and making games, perhaps the single most important thing employers look for in potential employees.

Read the special issue here: <http://gamedeveloper.texterity.com/gamedeveloper/2009fall>



GAME DEVELOPER ANNOUNCES REVAMPED ADVISORY BOARD

IN A CONTINUED DRIVE TOWARD

supreme excellence above all things, *Game Developer* is pleased to announce its newly-refreshed advisory board. Some of these folks have been on the board for a while now, but with this lineup, we're confident we have our board set for the foreseeable future.

Now working with us will be:

HAL BARWOOD Independent designer known for his work on many venerable LucasArts titles and for co-writing the *Dragonslayer* screenplay.

BRAD BULKLEY A tech director at Neversoft, and frequent contributor to *Game Developer*.

CAREY CHICO Executive art director for Pandemic Studios, and regular *Game Developer* Tool Box reviewer.

MARK DELOURA Independent tech consultant, previous Sony and Ubisoft exec, and former *Game Developer* editor-in-chief.

BIJAN FORUTANPOUR Senior graphics and tools engineer at Sony Online Entertainment and software tools author.

CLINTON KEITH Agile and SCRUM trainer, and former tech director for High Moon Studios.

MICK WEST Independent programmer, co-founder of Neversoft, and ex-code columnist for *Game Developer*.

CALENDAR

IEEE SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE AND GAMES

- Politecnico di Milano
- Milan
- September 7–10
- Price: \$300–\$580

www.ieee-cig.org

- Austin, TX
- September 15–18
- Price: \$895–\$995

www.gdaustin.com



GDC Austin.

GO3 ELECTRONIC & ENTERTAINMENT EXPO

- Perth Convention Exhibition Centre
- Perth, Australia
- October 1–4
- Price: See web site

www.go3.com.au

GAME DEVELOPERS CONFERENCE AUSTIN

- Austin Convention Center

KOREA GAMES CONFERENCE

- COEX
- Seoul
- October 7–9
- Price: See web site

www.kgconf.com



EVEN IF YOU
GOT YOUR
OFFICE CHAIRS
FROM THAT
DUMPSTER
BEHIND THE
BUILDING,

your game can be Unreal.

No matter what size your budget. No matter what type of game.
Unreal can be your game engine. Email Mark Rein at getunreal@epicgames.com



alternative geometry

THINKING OUTSIDE THE TRIANGLE

Thanks to science fiction-like advances in technology, games have taken a large step forward in image quality and photorealism during the past ten years. Games are the killer app of graphics hardware, and are a huge part of the raison d'être of that industry. In turn, if video cards did nothing more than drive the monitor, we'd all still be playing PONG. Because graphics hardware development and game development are so intertwined, many of the decisions made by one side affects the other. A primary example is the use of the triangle as the core rendering primitive. GPUs have been designed to process hundreds of millions of triangles per second, and so game engines are forced to express the game as triangles, the art tools have to output artistic vision as triangles, and finally, artists have to think in triangles.

Even though rasterization of triangles clearly has a stronghold on today's rendering pipelines, there are other methods and technologies that are available today. They may not be appropriate for rendering the entire game, but they may be appropriate for rendering parts of a 3D scene at interactive speeds. Alternatively, they may be useful in other parts of the game development process, such as tools for creating art assets. In this article, we will review technologies that are alternative to the triangle, but still make sense for use today.

EXPLICIT GEOMETRY

» First, let's begin with a brief summary of the drawbacks of using triangles. Triangles are very explicit; a lot of data is required to describe a single triangle. The location, normal, and texture coordinate of each vertex add up to a lot of data very quickly, especially for models with curved surfaces. This produces a bottleneck for data transfer between CPU and GPU, not to mention a lack of storage space in video memory for all that data.

This leads to the need for a series of programming optimizations to help manage the data efficiently. Examples include triangle stripping, creating data-aligned vertex buffers for position, normal, and texture coordinates, followed by index buffers to access them correctly, and sorting vertices such that they are used in sequential order to minimize missed hits to the hardware vertex cache. It's not really what one would consider part of the pleasures of graphics programming, unlike creative lighting, materials, and effects algorithms.

Sorted triangles with per-vertex and connectivity information are at the lower levels of the language of expressing geometry. One could think of it as akin to programming in assembly. You can express anything in assembly, and if used correctly it is faster than any other language, but it is way too detailed and explicit a language to be fun or practical for daily coding of large systems. Yes, given current hardware architectures, any form of geometry supplied to the system will eventually be converted to triangles before final display anyway,

alternative geometry

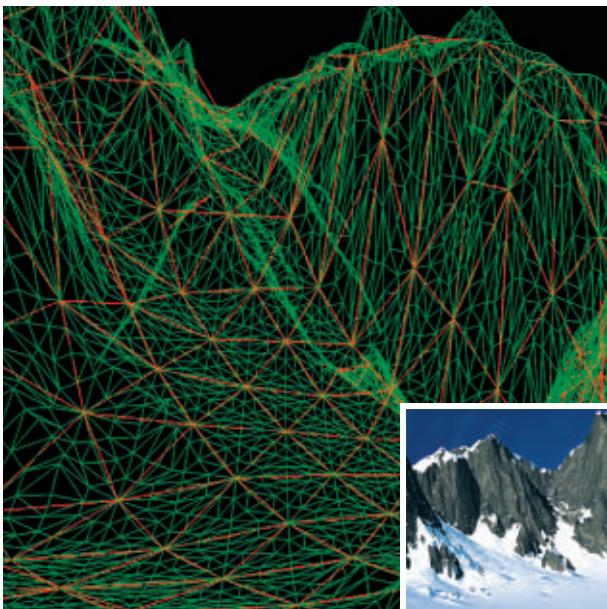


FIGURE 1 An example of real-time tessellation on a GPU from AMD's ATI Radeon HD 2000 series demo.

but that shouldn't necessarily dictate the need to push triangles all the way up to the engine and art pipeline.

PARAMETERIZED GEOMETRY

► At the other end of the geometry definition spectrum comes the class of parameterized geometry, described by mathematical equations. Curves and curved surfaces fit into this category, as do subdivision surfaces, Bezier patches, NURBS (Non Uniform Rational B-Splines), and quadric surfaces. There are at least four advantages to using parameterized curves and surfaces over triangle meshes. First, they require much less data than explicit meshes do. That means less memory storage, fewer transformation calculations, and less bus bandwidth. Second, they are scalable, providing any amount of detail desired. For instance, a curved patch can be tessellated into just a few triangles, or many thousands of triangles, depending on the distance to camera. The

silhouettes of flat triangles by generating a curved surface to replace each triangle. The curved surface is created by generating new triangles which approximate the curved surface on the fly. This actually can be accelerated by hardware because it can generate the surface using only the normals and positions of the original underlying triangle. No information from neighboring triangles is needed.

ATI in fact has implemented hardware acceleration of N-patches in some of its chips. Beginning with the 8000 series of chips, this hardware feature was called TRUFORM, and was available in many games—QUAKE, QUAKE 2, UNREAL TOURNAMENT 2003 and 2004, MADDEN NFL 2004 and WOLFENSTEIN: ENEMY TERRITORY, among others.

But beginning with the Radeon X1000, the feature was no longer advertised, probably because of a lack of wider acceptance among developers. It also didn't help that Nvidia didn't have a similar feature in its hardware at the time. Nvidia had a competing tessellation solution based on quintic-RT patches, which did not receive much attention from developers. QuinticRT patches are based on quintic equations—polynomial equations of degree five. Quintic equations have some interesting properties, although I can't help but be reminded of the scene in *Spinal Tap* where Nigel declares, "This one goes to 11," while describing his amp. Usually third and fourth degree polynomials are more than sufficient to represent the needed curvature.

Beginning with the R600 GPU found in the ATI Radeon HD2000–4000 series, new hardware tessellation units were added. The new hardware is placed before the vertex shader stage in the GPU pipeline. It does not calculate the final vertex positions, nor store them in video memory. Instead the tessellator generates new vertices and their positions as parametric coordinates, which are then passed to the vertex shader, along with the vertex indices of the base mesh. It is then the vertex shader's responsibility to convert the parametric coordinates to world coordinates, as well as to apply the desired surface evaluation function. This may be any algorithm for evaluating any higher order surface, such as Bezier, NURBS, or Catmull-Clark subdivision surfaces.

It's important to note that the tessellation is done in one pass through the GPU pipeline, and no extra video memory is needed to store the generated data. The tessellation unit can generate up to 411 triangles from a single source triangle. In one of the technical demonstrations from ATI, realtime tessellation of terrain geometry is performed with a model of 4050 triangles being tessellated to over 1.6 million triangles, with a frame rate of 110 frames per second. See Figure 1 or visit www.gametrailers.com/video/tessellation-tech-atiradeon/20445 for a video of the demo.

Access to the new hardware is not part of OpenGL or DirectX 10, but is available as an extension to DirectX 9 and Direct X 10 from ATI's website, along with an excellent paper by Tatarchuk, Barczak, and Bilodeau, "Programming for Real-Time Tessellation on GPU" which also

discusses the upcoming DirectX 11 tessellation architecture [see <http://developer.amd.com/gpu/radeon/Tessellation>].

Clearly, there is a vision and desire for graphics hardware handling curved surfaces at interactive frame rates. They haven't become ubiquitous yet, but it may simply be a matter of time. Working with curved patches in a production pipeline does have its drawbacks though. One problem to overcome is the need to maintain continuity of the curved surface between patches. Manipulating the control hull (and thus the surface) of one patch may create tears and seams with neighboring patches. Subdivision surfaces address this problem nicely, as the control hull mesh can have any arbitrary topology and is smoothly evaluated between neighboring faces. Subdivision surfaces have already made

Even though today's GPUs process triangles faster than curved primitives, the advantages of surfaces are compelling enough to encourage researchers and hardware companies to keep trying.

third advantage is that they simply look better, and represent rounded characters, terrain, or terrain objects such as trees or rocks better than triangles do. Lastly, for per-frame calculations such as animation or collision, they require fewer calculations. The geometry hull which contains fewer vertices can be used for any calculations, and then the smoother and higher resolution mesh can be generated and rendered afterward directly by the graphics hardware.

Even though today's GPUs process triangles faster than curved primitives, the advantages of surfaces are compelling enough to encourage researchers and hardware companies to keep trying. One relatively recent idea is N-patches, short for "normal patches," also called PN triangles. The goal of PN triangles is to improve the shading and

their way into current art pipelines with mainstream art tools such as ZBrush and Mudbox.

Even though there currently doesn't seem to be any effort focused on creating subdivision-based game engines, it clearly is possible. One option is hardware tessellation, mentioned above, but the other is a method known as valence subdivision, by David Brickhill. The technique is a high-speed rendering algorithm for subdivision surfaces that was implemented in a game engine running on a PlayStation 2. The algorithm achieves interactive speeds by avoiding recursion and its disadvantages, namely very inefficiently accessing memory across areas too large to fit into a scratch pad or data cache.

IMPLICIT, VOLUMETRIC GEOMETRY

» One of the more interesting methodologies for 3D modeling is voxels. The term voxel is short for volume element, akin to the pixel's meaning of picture element. It follows then that three dimensional space can be thought of as a three dimensional grid of voxels, in the same way that a digital image is a two dimensional grid of pixels. Looking at past usage of voxels in games, aside from the basic definition just mentioned, different people speak of voxels in different terms and contexts. For instance, the assumptions on a voxel set's uniformity of size, adjacency, or alignment requirements are all left to the individual algorithm and context. Even the data contained in a voxel and how it is rendered is open to interpretation.

As an example, NovaLogic Inc, creator of the COMANCHE and DELTA FORCE series, was awarded a patent in 2000 for a voxel-based terrain rendering algorithm used in the VoxelSpace game engine. However, the terrain was actually a 2D heightfield array, and each grid in a two dimensional horizontal grid is extruded to a given height, yielding a tall volume, thought of as a voxel. This algorithm for terrain generation and rendering was very innovative for its time. It used the CPU for the bulk of the work, and produced results that were impossible to achieve with the underpowered GPUs of its time.

Fast forward to present day. CRYYSIS' art toolset, Sandbox, includes a terrain creation and editing system, which includes voxel editing capabilities. The particular use of voxels in this context is to allow for freedom from a fixed single-layer topology, and allows for creation of caves, overhangs, and other complicated configurations. The definition



NovaLogic's COMANCHE 4 employs voxel-based terrain rendering.

of voxels in this context is more in line with the scientific community's definition of a voxel.

Another great example of the use of voxels in current games is Maxis' SPORE. The creature building part of the game involves constructing an arbitrarily-shaped body, followed by attaching any number of limbs, of varying types. The limbs can be placed anywhere along the body, using any topology the user wishes. This is done using metaballs, also known as blobbies, which are part of a voxel-based algorithm. The difference between metaballs and regular voxels is that instead of each voxel containing only one of two values [inside or outside the model], the voxel defines a weighting factor, or probability that a surface exists. This allows for the creation of a smooth surface attaching and blending two different models together.

True voxels began with the medical visualization community and are used in Magnetic Resonance Imaging to uniformly sample 3D space. Therefore all voxel are adjacent, same-sized cubes, defined by a single number specifying whether they lie inside the model volume or outside.

Voxels have several advantages over polygonal geometry. The first is smaller data size. A voxel is a single element in space, a single number determining if the voxel is inside or outside of the model's volume. Unfortunately, depending on the application, a lot of voxels may be

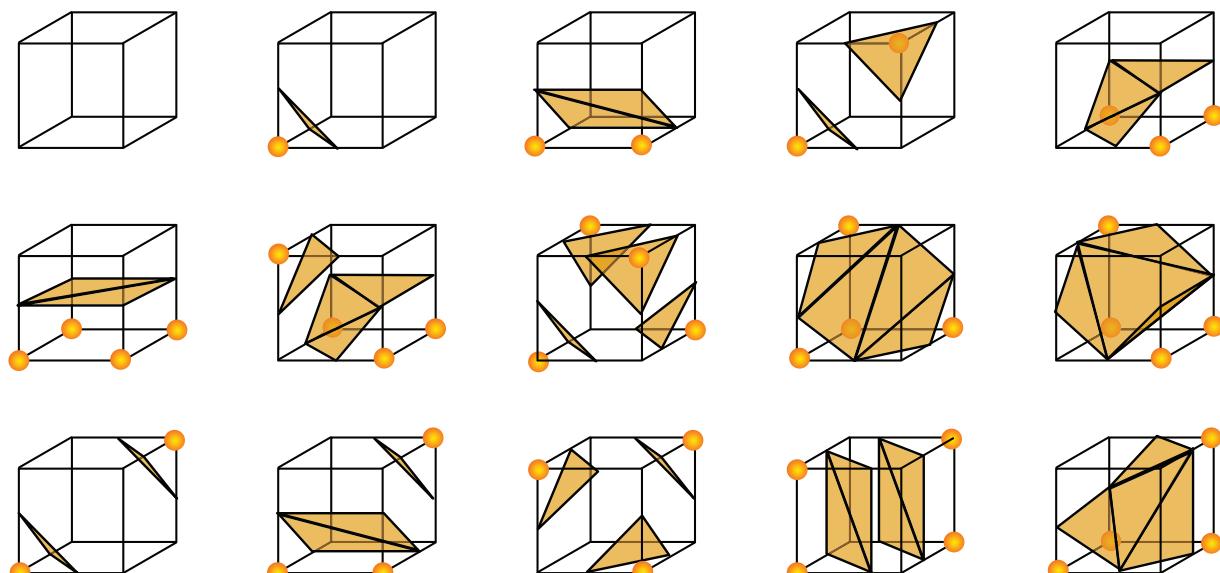


FIGURE 2 The "Marching Cubes" algorithm creates a triangulated isosurface from voxel data.

alternative geometry



FIGURE 3 SPORE CREATURE
CREATOR uses metaballs, a variation of voxels.

required to render an object or scene, so smart solutions for handling this problem are still required.

The second is its uniform sampling of space, as compared to a polygon. In a polygon-based artist tool for sculpting 3D models, if a vertex is placed too far relative to its neighboring vertices, stretching will occur, which may result in texturing artifacts, and reduced geometry resolution for placing more details, and a possible loss of smoothed curvature. In a voxel-based modeling program, there are no such limitations, as the volume of space is evenly sampled, and new vertices are automatically added to the model surface when the voxels are used to generate the polygonal mesh for rendering.

The third advantage voxels have is that they don't have any topological constraints relative to other parts of the model geometry. Again, this can be very useful when sculpting a character model where limbs and appendages can freely be removed or attached.

There are very few commercial voxel-based 3D sculpting applications available today. One that comes to mind is 3DCoat, which clearly demonstrates the advantages mentioned above.

How to render true voxel data, and whether it can be done at interactive speeds for games, has been the million dollar question. There is also the question of the interaction of voxel data with the remaining elements of a game—for instance, physics calculations of rigid body dynamics, interaction with particles, game characters, and so forth.

One of the problems with voxels is that even though a single voxel may itself may be lightweight, when converted to triangles they can create an incredible amount of data—triangle bloat.

There are a few approaches to rendering voxel data. The first is to create an isosurface, essentially connecting the dots and creating a triangle mesh to be rendered by the GPU. The most well-known method is the “marching cubes” algorithm, which until recently was under patent and not available for wide, unlimited use. However, the patent has expired, and there has been renewed interest by developers. The algorithm basically visits each voxel cube (thus the name “marching cubes”), and generates the needed triangles. It does so by looking at the eight corners of the voxel, which is a single point in space with a number associated with it specifying if it is inside the 3D model (one) or outside (zero). For each of the voxels, if an edge has one vertex with a value of zero and one vertex with a value of one, then a triangle vertex is created at the midpoint of the voxel edge.

Adjacent triangle vertices are connected to create a triangle. The triangle creation step is sped up by creating an array of all 256 possible triangle configurations, and indexing into the array. Figure 2 shows the 15 unique cases of the 256 configurations, the rest are rotated or mirrored variations. For instance a cube with vertices valued 0, 1, 0, 1, 1, 0, 0, 0 will produce 01011000 in binary format, or 56 in decimal.

One of the problems with voxels is that even though a single voxel may itself may be lightweight, when converted to triangles they can create an incredible amount of data—triangle bloat. Each voxel can produce up to four triangles, which means twelve vertices. The number of voxels needed to cleanly represent a 3D scene can be quite large, quickly leading to gigabytes' worth of triangle vertex data.

Marching cubes is not the only method for creating triangulated isosurfaces from voxel data. Another algorithm, found in *Graphics Gems III*, “Compact Isocontours from Sampled Data,” was used in SPORE’s creature builder. Other algorithms exist as well, using other shapes, such as “marching tetrahedrons,” originally intended to circumvent the patent on marching cubes.

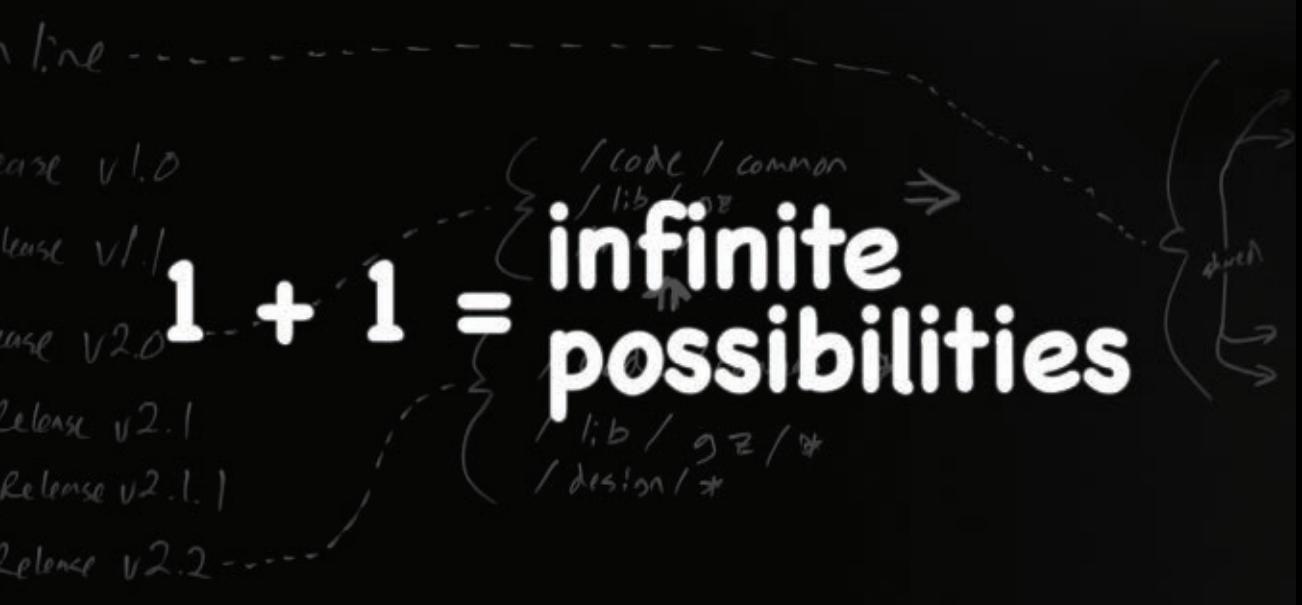
Much research has gone into fast rendering of voxel data sets on current graphics hardware. The research and implementation challenge comes down to creating a real-time implementation of marching cubes, and there has been significant work done in this area. Interestingly, NVidia released a geometry shader in its GeForce 8800 series of cards, and

created a very compelling demonstration in the Cascades demo, discussed at GDC 2007. The demo consists of generating voxel data to create a rock formation by rendering voxel density values into a 3D texture. Then, using the geometry shader, the marching cubes algorithm is used to generate a triangle mesh. Afterward, water particles are

dynamically generated and physics calculations are used to determine collision between particles and the rock geometry. Additionally, swarms of dragonflies fly around, avoiding the rock as well. Although a very compelling demonstration of the new geometry shader hardware units, it remains to be seen how effectively it will be used in a 3D game. As a first step, the hardware shaders may be better suited for use in an art tool as part of the source art creation pipeline.

Another interesting project in voxel rendering is “High Speed Marching Cubes Using Histogram Pyramids” by Dyken and Ziegler, and can be found at www.mpi-inf.mpg.de/~gziegler. The HistoPyramid algorithm, which is usually used in data compaction for GPUs, was used here for data expansion of voxels into triangles. The algorithm does not use

www.seapine.com/gamescm
Satisfy your quality obsession.



© 2009 Seapine Software, Inc. All rights reserved.

Get More Change Management with Seapine Integrated SCCM

TestTrack Pro + Surround SCM = infinite SCCM possibilities. Seapine's integrated software change and configuration management (SCCM) tools do much more than competing tools, and at a much lower price point. Start with TestTrack Pro for change management and add Surround SCM for configuration management—two award-winning tools that together give you the best integrated SCCM solution on the market.

- Link issues, change requests, and other work items with source code changes.
- Manage simple or complex change processes with flexible branching and labeling.
- Coordinate distributed development with RSS feeds, email conversation tracking, caching proxy servers, change notifications, 3-way diff/merge, and other collaboration features.
- Enforce and automate processes with incredibly flexible work item and file-level workflows.

Built on industry-standard RDBMSs, Seapine's SCCM tools are more scalable, give you more workflow options, and provide more security and traceability than competing solutions.

Get more, do more with Seapine tools. Visit www.seapine.com/gamescm.

 Seapine Software™



alternative geometry

a geometry shader, but runs three to four times faster than running marching cubes with hardware acceleration on the geometry shader.

Finally, as mentioned earlier, creation of a triangle mesh isn't the only method of visualizing voxels. The other option is volume raycasting. Like raytracing, rays are cast from the camera, through each pixel on the screen, into the voxel data, sampling it and making the final calculations. The advantage of ray casting is that it inherently removes hidden surfaces as well. Voxels that are not seen are not used, and no computation time is wasted on them. Another benefit is that partially transparent surfaces can be taken into account when calculating the final pixel color. One disadvantage of ray casting is that it does not produce any geometry, which may be desirable for physics or collision calculations.

SPARSE VOXEL OCTREES

» During Siggraph 2008, id Software demonstrated the concepts behind the company's latest game engine, id Tech 6. The goal of the new engine is to achieve an unlimited amount of unique texturing and unique geometry. This is accomplished by ray casting into an octree structure of voxels. Octrees are spatial data structures used to partition three-dimensional space by subdividing a scene into eight octants of identical size, and recursively subdividing only each octant that contains geometry to the smallest desired level of detail. The sparse voxel octree algorithm is conceptually easy to understand, but nontrivial to implement. As part of the art processing pipeline, polygonal source art must first be broken down into voxels, called voxelization. There are different methods of voxelization, such as 3D scan conversion, volume projection, and subdivision.

The created voxels are then hierarchically stored in an octree. Each voxel contains the color of the surface model at that position in space. The octree is now considered to be the object that is rendered, there are no polygons so to speak. During runtime, rays are cast from the camera position through each pixel location on screen, and the octree is sampled. If the voxel in the octree that intersects the ray is larger than the size of a pixel, the subvoxels stored in the lower level of the octree are retrieved and the ray stepping is continued. When the ray stepping intersects a voxel that is less than the size of a pixel, its color is used to set the value of the pixel on screen.

One way to think of sparse voxel octrees is as mip-mapping for geometry. As mentioned earlier, the details of implementation are nontrivial. The final result of creating octree data for a typical 3D scene for a game will be very large, on the order of tens of gigabytes. This is way beyond the memory capacity of graphics hardware of many PC systems. So, the next required system is a streaming paging system designed and optimized for hierarchical storage and retrieval of octree data.

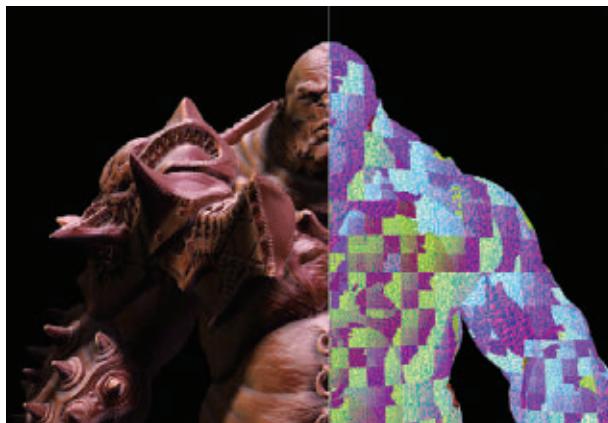
Sparse voxel octrees are intended mostly for creation of large static objects with unique detail, namely terrain. For dynamic, animated objects, more work is required and may not be worth the effort. The animation data would be used to deform the octree voxels, and as rays are cast through the screen pixel into the deformed octree, rays would have to be bent to follow the same deformations. Although possible in theory, in practice it still makes sense to follow the traditional triangle rasterization approach.

Further interesting research similar to sparse voxel octrees has been done in a paper called "Interactive GigaVoxels" by Crassin, Neyret, and Lefebvre and can be found at <http://artis.imag.fr/Publications/2008/CNL08>. The work is based on a dynamic generalized octree with mip-mapped 3D textures at the lowest level leaves of the octrees.

CONCLUSION

» In reviewing NURBS, subdivision surfaces, and voxels, it's clear that they are viable alternatives to triangles. Although, like triangles, they have their weaknesses, and each method provides a good solution to a specific problem.

FIGURE 4 An example of 3D modelling that uses sparse voxel octrees from Jon Olick's SIGGRAPH 2008 presentation.



NURBS are ubiquitous in the CAD industry, and are still the preferred method for hard surface modeling of objects with smooth curves. For a car racing game, using NURBS at least in the art pipeline makes sense, both for their ease of use and flexibility when it comes to final tessellation.

Subdivision surfaces avoid the seaming problems of NURBS surfaces, and are a great approach to modeling organic surfaces such as characters and creatures. Subdivision surfaces have been used in at least one game engine on the PlayStation 2, and already have a strong foothold in the art creation pipeline in tools such as Zbrush and Mudbox.

Voxels have traditionally been used for terrain rendering in game engines, as a fast method to avoid using the GPU. Currently voxels are still ideal for authoring terrain because they allow more complex terrain type creation such as caves, arches, and overhangs. They also are ideal for modeling sheer vertical cliffs, since voxels provide uniform sampling of 3D space and avoid the stretched triangle problem.

Voxels can also be used as the basis for a game engine, as they can provide a stylized look to the game and are perfect for implementing completely destructible environments. Additionally, voxels can be used in limited scale in game engines where flexible character geometry topology is needed, as seen in SPORE CREATURE CREATOR.

As graphics hardware advances are made, whether in speed and memory capacity, or new features such as hardware tessellation, these methods may very well become as ubiquitous as the triangle. ☺

BIJAN FORUTANPOUR is a senior graphics programmer at Sony Online Entertainment in San Diego with 16 years experience in the visual effects and games industries. He is also the author of Enzo 3D paint for Photoshop (www.enzo3d.com). Email him at bforutanpour@gdmag.com.

Look Who's #1



Top Products for the Top Games



Award Winning Products & Services.

Over 250 Shipped Games.

Cross Platform Optimized.

The Proven Solution.

havok™



PHYSICS



ANIMATION



BEHAVIOR



CLOTH



DESTRUCTION



AI

©2008 THQ International GmbH. THQ, de Blob and their respective logos are trademarks and/or registered trademarks of THQ Inc.
All Rights Reserved. All other trademarks, logos and copyrights are the property of their respective owners.



THE MAKERS OF EVE ONLINE ARE ACTIVELY RECRUITING LEADING TALENT

SENIOR PROGRAMMERS • 3D ARTISTS • TECHNICAL DIRECTOR • ANIMATORS
QA ENGINEERS • SENIOR ANIMATION PROGRAMMERS • UI DEVELOPERS

» ATLANTA • REYKJAVIK • SHANGHAI «

VISIT US ONLINE AT WWW.CCPGAMES.COM/JOB
TO LEARN MORE AND APPLY TODAY

© 2008 CCP hf. All rights reserved. Reproduction without the written permission of the publisher is expressly forbidden. EVE, EVE Online, CCP logo are registered trademarks of CCP hf.

demystifying matrix layout considerations for modern architectures

MATRICES ARE OFTEN USED IN THE MOST TIME—CRITICAL PARTS OF A game. For example, most shaders, scene graphs, and physics systems make heavy use of matrices. Often special hardware is available to run matrix code efficiently. Ideally, matrices should make optimal use of this hardware. Designing an efficient matrix type is mostly about choosing the right matrix layout. But choosing the matrix layout is not as straightforward as might be expected. The optimal matrix layout depends on multiple factors like target hardware, storage type, vector type, and the particular matrix type that you need to interface with. All of this can be quite confusing. This sometimes results in misconceptions: We found that programmers sometimes wrongly assume that the choice between row- and column-vectors affects performance. In this article, we will explain how to design an efficient matrix type for modern CPU and GPU architectures. Hopefully, we will demystify matrix layouts for the reader.

One important aspect when designing a matrix type is deciding how to store the matrix in memory. There are a few things to consider when choosing storage. One is how this layout is compatible with other matrix types. DCC tools, platform APIs, physics engines, and shader compilers all have their own way of storing matrices. Understanding how your matrix implementation should communicate with other matrix implementations can help you decide what matrix format to use. It can even improve performance. Another thing to consider is how the layout in memory affects matrix multiplication performance. Different implementations may have different performance characteristics, depending on the target hardware.

We begin by reviewing the basics of matrix layouts, and will later ramp up and show if and how different matrix layouts can affect performance using SSE, SSE4, and AltiVec SIMD instruction sets and the Cg shading language.

CONTINUED ON PAGE 16

MATRIX MEMORY LAYOUT

» Matrix layout is determined by two concepts: the vector notation type used [row or column vectors] and storage type used [row-major or column-major storage].

Vectors can be written in two ways: either as a row [a 1x4 matrix] or a column [a 4x1 matrix]. Using one vector form or another has implications on the order of multiplication with matrices. Let us define a row vector v_{row} , a column vector v_{col} and a square matrix M :

$$v_{row} = [x \ y \ z \ w]$$

$$v_{col} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}$$

Multiplication of v_{row} with M is only defined if v_{row} is pre-multiplied with M , and it yields another row vector:

$$v'_{row} = v_{row} M$$

Multiplying v_{col} and M is done by post-multiplying v_{col} , and it yields a column vector:

$$v'_{col} = M v_{col}$$

However, matrix multiplication is not commutative. In other words, we cannot just switch the multiplication order around. To see the implications of switching multiplication order, let us look at the following definition:

$$vM = [M^T v^T]^T$$

This means multiplying a vector with a matrix is equal to: 1) transposing the matrix, 2) transposing the vector [this converts the vector from row to column or the other way around], 3) reversing the order of multiplication, and 4) transposing the resulting vector [this converts the resulting vector back to a row or column vector]. Generally, C/C++ APIs do not expose specific types for row and column vectors. Instead, the use of a vector determines whether it is interpreted as a row or column vector. Often C/C++ APIs do so by offering a pre-multiply and post-multiply operation. For instance, in Cg and HLSL the mul operation has multiple overloads, one with the vector as the first argument and the matrix as the second argument. In this case the vector is interpreted as a row vector. When the arguments are switched, the vector is interpreted as a column vector. Transposing the vector is not done by explicitly transposing the vector, but by using it differently.

This leaves us with the transposed matrix. And this is exactly what is important. When using row vectors, the matrices are the transposed versions of what would have been used when column vectors were used, and vice versa.

The second factor in understanding matrix layouts is the way matrices are stored in memory.

There are two ways matrices can be stored in memory: in row-major and in column-major format. This means that either a matrix's row or column is stored consecutively in memory. It is easy to see that transposing a matrix has the same effect as switching rows and columns. So, when you need to convert a matrix from storage format, transposing the matrix will do just that.

A factor that seems related to matrix layouts is the way matrix types are named and how they are indexed. In mathematics, matrix dimensions are written as rows x columns. Matrix types are generally named in the same way in C/C++ APIs. For instance, in Cg and HLSL, the float 4x3 type is a matrix with four rows and three columns. This definition is completely independent from how a matrix is actually stored in memory. Likewise, indexing elements of a matrix is performed in the same order, with row number preceding column number. So, APIs often have operator overloads that index in the same way. For instance, matrix[2, 3] will generally mean: "row two, column three," regardless of the matrix's storage. It's important to remember that naming and indexing have actually nothing to do with matrix layout.

CONVERTING FROM ONE FORMAT TO ANOTHER

» So, there are two factors to consider when converting matrices: the type of vector that is used [row or column vectors] and the type of matrix storage [row or column major]. We have seen that both the vector type and the matrix storage can be converted to their counterparts by transposition. There are three possible scenarios when converting:

- Both the vector- and storage type are equal. This means that no transposition is required before converting.
- Either the vector- or the storage type differ. This means that the matrix needs to be transposed, either with a vector type change or a storage type change.
- Both the vector type and storage types differ. The matrix needs to be transposed twice, resulting in the original matrix. Hence, the matrix does not need to be transposed.

For example, imagine a DirectX matrix that needs to be converted to an OpenGL matrix. DirectX uses row vectors and stores its matrices in row-major format. OpenGL uses column vectors and stores its matrices in column-major format. So we need to switch from row to column vectors and from row-major storage to column-major storage. This falls into category three, meaning no changes are required.

BINARY COMPATIBILITY

» If you want to design a matrix class in such a way that it encapsulates the matrix contents properly, there is no alternative but to make a copy of the matrix upon conversion. Although encapsulation is a very good software engineering practice, a matrix implementation should often be designed for performance. We can gain performance by exploiting the fact that the memory contents of the objects are interchangeable. We call these matrices binary

listing 1

```
const Vector4 Transform(const Matrix4x4& a_Matrix, const Vector4& a_Vector)
{
    return Vector4(a_Matrix(0, 0) * a_Vector.X() + a_Matrix(0, 1) * a_Vector.Y() +
        a_Matrix(0, 2) * a_Vector.Z() + a_Matrix(0, 3) * a_Vector.W(),
        a_Matrix(1, 0) * a_Vector.X() + a_Matrix(1, 1) * a_Vector.Y() +
        a_Matrix(1, 2) * a_Vector.Z() + a_Matrix(1, 3) * a_Vector.W(),
        a_Matrix(2, 0) * a_Vector.X() + a_Matrix(2, 1) * a_Vector.Y() +
        a_Matrix(2, 2) * a_Vector.Z() + a_Matrix(2, 3) * a_Vector.W(),
        a_Matrix(3, 0) * a_Vector.X() + a_Matrix(3, 1) * a_Vector.Y() +
        a_Matrix(3, 2) * a_Vector.Z() + a_Matrix(3, 3) * a_Vector.W());
}
```

compatible. You can exploit binary compatibility by copying the contents of the matrix object into another object. Often it is possible to just pass the address of a matrix so that it is interpreted as another matrix. A more exotic (and dangerous) form of exploiting binary compatibility is to test the contents of two matrices for equality using a fast memory compare.

But be advised, compatible vector- and storage types between matrix classes do not necessarily guarantee equal memory layouts. The matrix's data is assumed to be stored consecutively in memory and no additional data members should precede or follow the matrix data. In C++, a matrix class also should have no vtable, as this affects the memory layout as well. This means that there should be no virtual methods in your matrix class. Finally, compilers may decide to pad data to data members because of data member alignment. The contents of the padded data are undefined and will most probably differ per object. This is why comparing the contents of C++ objects by performing a memory compare is dangerous. Dedicated compare code is much safer, more portable and, unlike the conversion tricks, does not impose performance penalty.

Exploiting binary compatibility between C++ objects is generally considered bad software engineering practice. It breaks encapsulation completely and it is often hardly portable. Matrices could be an exception to the rule, but understanding the dangers such as the ones described above is important when designing your own matrix class. For more information about this subject refer to item 96 in C++ Coding Standards (see Resources, pg. 20).

EFFICIENCY CONSIDERATIONS

» Single Instruction Multiple Data (SIMD) is a form of data-parallelism in which a single instruction is executed on multiple data elements at a time.

Nowadays SIMD hardware is available in the form of graphics cards, PCs, and gaming consoles.

Matrix multiplication lends itself particularly well to optimization using SIMD techniques. By using SIMD hardware it's often possible to double or even quadruple the performance of a matrix-vector multiplication compared to a completely sequential implementation.

Most current SIMD hardware is extremely picky about data layout. We can only achieve optimal performance by organizing our matrices in memory in such a way that they can be handled efficiently by SIMD hardware. Of course, which memory layout is the most efficient depends on the particular SIMD hardware used. In the remainder of this article we will discuss how to optimize matrix multiplication using GPU shaders, SSE, and AltiVec. We assume the use of column-vectors from here on after. If the reader understands the preceding part of this article, he or she should have no problem adapting the same principles to a system that uses row-vectors instead.

GENERIC IMPLEMENTATION

» Before we move on to SIMD implementations of multiplying a matrix with a column-vector, here is a generic C++ implementation (see Listing 1).

This code contains little information about the order in which the multiplications and additions should be executed. In principle, a compiler should be able to reorder the operations and generate efficient vectorized code. In practice, however, most current compilers do a bad job at vectorizing

listing 2

```
const Vector4 Transform(const Matrix4x4& a_Matrix, const Vector4& a_Vector)
{
    __m128 rhs = GetSSEVector(a_Vector);

    __m128 broadCastX = _mm_shuffle_ps(rhs, rhs, 0x00);
    __m128 result = _mm_mul_ps(broadCastX, GetSSEColumn(a_Matrix, 0));

    __m128 broadCastY = _mm_shuffle_ps(rhs, rhs, 0x55);
    __m128 rowResult = _mm_mul_ps(broadCastY, GetSSEColumn(a_Matrix, 1));
    result = _mm_add_ps(result, rowResult);

    __m128 broadCastZ = _mm_shuffle_ps(rhs, rhs, 0xAA);
    rowResult = _mm_mul_ps(broadCastZ, GetSSEColumn(a_Matrix, 2));
    result = _mm_add_ps(result, rowResult);

    __m128 broadCastW = _mm_shuffle_ps(rhs, rhs, 0xFF);
    rowResult = _mm_mul_ps(broadCastW, GetSSEColumn(a_Matrix, 3));
    result = _mm_add_ps(result, rowResult);

    return Vector4(result);
}
```

listing 3

```
const Vector4 Transform(const Matrix4x4& a_Matrix, const Vector4& a_Vector)
{
    __m128 sseVector = GetSSEVector(a_Vector);
    __m128 xComponent = _mm_dp_ps(GetSSERow(a_Matrix, 0),
                                    sseVector, 0xF1);
    __m128 yComponent = _mm_dp_ps(GetSSERow(a_Matrix, 1),
                                    sseVector, 0xF2);
    __m128 zComponent = _mm_dp_ps(GetSSERow(a_Matrix, 2),
                                    sseVector, 0xF4);
    __m128 wComponent = _mm_dp_ps(GetSSERow(a_Matrix, 3),
                                    sseVector, 0xF8);

    __m128 result = xComponent;
    __mm_add_ps(result, yComponent);
    __mm_add_ps(result, zComponent);
    __mm_add_ps(result, wComponent);
    return CreateVector4FromSSEVector(result);
}
```

code. This means, unfortunately, that we have to assist the compiler by manually reordering instructions and using compiler intrinsics.

MATRIX-VECTOR MULTIPLICATIONS USING GPU SHADERS

» Matrices are passed to shaders as uniform parameters. Uniform parameters are stored in shader registers. These shader registers are four floats wide, so an entire 4x4 matrix cannot be stored in a single register. The shader compiler therefore allocates multiple registers for a single matrix. The way these registers are allocated depends on the shader compiler. In HLSL, matrices are stored in column-major format by default. This means that one column is stored per register. At first glance this seems peculiar since DirectX stores its matrices in row major format. If you are using row vectors in your shader, a transposition of the matrix is required before passing it to the shader. DirectX hides these details in most cases, but not in all cases. When you pass a uniform parameter by using direct register allocation, you have to know for sure how the compiler allocates the registers and whether you need to transpose the matrix before passing it, or not. The reason why the HLSL compiler prefers column-major format will become clear after we have seen how vectors and matrices are multiplied using shader assembly.

GPUs are equipped with a large number of instructions that can be used for computing matrix-vector multiplications. All GPUs that support shader model 1.1 and upward are equipped with dot product, multiply, add,

and fused multiply-add instructions. These instructions are all extremely efficient and can often execute in a single cycle. This gives the programmer a lot of flexibility in choosing an appropriate matrix layout.

We can use the dot-product instruction to multiply a matrix with a vector. For each component of the resulting vector we have to compute a dot-product between a row of the matrix and the vector itself. If matrix-vector multiplication is computed in this manner, it is most efficient to store the matrix in row-order. This way a constant register can hold a single row of the matrix, and the dot product can be computed directly between the constant register and the register holding the input vector. The next code sample shows the code generated by the Microsoft HLSL compiler for a multiplication between a 4x4 matrix and a 4D vector.

```
vs_3_0
dcl_position v0
dcl_position o0
dp4 o0.x, v0, c0
dp4 o0.y, v0, c1
dp4 o0.z, v0, c2
dp4 o0.w, v0, c3
```

Alternatively, we can use the multiply and fused multiply-add instruction to multiply a matrix with a vector. Instead of computing the result component-wise, we compute the entire result vector by taking a linear combination of matrix columns. We multiply each component of the input vector with the corresponding column of the matrix. The results of each component of the input vector are separately accumulated to get the final result. This can be done most efficiently if the matrix is stored in column-order. Each component of the input vector can be replicated in all components of a register and multiplied with the corresponding column in the matrix. The final result vector can be efficiently computed using a number of fused multiply-add instructions.

The following shader code sample shows the generated shader assembly for a shader that is compiled with column-order first matrices.

```
vs_3_0
dcl_position v0
dcl_position o0
mul r0, c1, v0.y
mad r0, v0.x, c0, r0
mad r0, v0.z, c2, r0
mad o0, v0.w, c3, r0
```

In terms of efficiency, there is probably not much difference between the two code samples. The generated shader assembly uses the exact same number of instruction slots, and all instructions execute in a single cycle.

The difference becomes more apparent when we are dealing with non-square matrices. For example, suppose we need to transform a 3D vector by a 3x4 matrix. In this case row-order first storage requires one constant register more than column-order first storage. Also, an extra dot-product instruction is necessary to compute the result.

By now you should understand why HLSL prefers column-major storage in shader registers even while DirectX uses row-major storage in system memory. HLSL expects users to use row vectors in vertex shader code. This can be done most efficiently by using column-major storage. Likewise, the Cg compiler uses row-major storage, while OpenGL uses column-major storage because it is expected that users will use column vectors in shader code.

Another interesting case is the use of square 4x4 matrices where the last row of the matrix is ultimately not used in the shader. This can be the case when the w component of a vector that was the result of a matrix-vector multiplication is ultimately not used in the shader. This is sometimes easier to spot for a compiler than it is to the human eye, so this may require a careful look at your vertex shader code. The fact that the last row does not result in any significant output means that it could have been stored in a 3x4 matrix.

This is often the case since the last row is generally only used for projections. Shader compilers can detect whether the last row of a 4x4 matrix is used. Some compilers, like the scc-cgc compiler on the PlayStation 3, require a special flag to use this optimization. Other compilers, like the HLSL compiler, will perform this optimization by default. In these situations, the HLSL compiler will only allocate three registers instead of four. This optimization is very dangerous when performing direct register allocation. If you assume that four registers are allocated and assign values to the register that was optimized out, the results are undefined. So if you are not using the last row of the matrix, it is good practice to use 3x4 matrices instead of 4x4 matrices.

Some shader compilers, such as the Microsoft HLSL compiler, give the programmer the ability to specify the matrix layout. This is very convenient because it enables the programmer to experiment with both options. The programmer can select the option that is compatible with the CPU matrix layout and avoid transposing, or evaluate the generated shader code and select the most efficient shader assembly. If a shader compiler has no packing option, like the Cg compiler, the same effect can be obtained in a more cumbersome way by switching between row- and column-vectors in the shader code.

MATRIX-VECTOR MULTIPLICATIONS ON THE CPU

» The latest trend is to move more and more operations onto the GPU in order to achieve more data parallelism. However, current CPUs are much more flexible than GPUs, and therefore most current games still have to perform a substantial number of matrix multiplications on the CPU. Transforming a vector by a matrix can eat away a significant portion of the total CPU time required for running a game. Hence, it often makes perfect sense to optimize the performance of these operations, for example by using SIMD techniques.

Almost all modern CPUs are equipped with SIMD multiplication and add instructions. Some CPUs even have a fused multiply-add instruction, which can make matrix-vector multiplication substantially faster. Certain CPUs are equipped with a SIMD instruction set that includes a dot product operation. For these CPUs, the programmer has the same flexibility in choosing matrix layout in memory as on a GPU. The dot product instruction is available on recent Intel processors with SSE4 support and on PowerPC processors with the AltiVec instruction set.

MATRIX-VECTOR MULTIPLICATIONS USING SSE

» The dot-product instruction available in SSE4 is powerful, but unfortunately it is only available on the latest Intel processors. Most PC games still have to run on older systems that lack SSE4 support. Intel processors starting with the Pentium 3 are equipped with SSE, which stands for Streaming SIMD Extensions. The SSE instruction set works on 128-bit registers that hold four single-precision floating point values at a time. SSE does support unaligned loads. To achieve optimal performance, vectors should be aligned on 16-byte boundaries. The original SSE instruction set architecture (ISA) lacks both a fused multiply-add and a dot-product instruction. However, we can still achieve pretty good SIMD parallelism by executing the vector-matrix multiplication in a different order as we have shown in the previous section. We can multiply one component of the input vector at a time by replicating, sometimes called *broadcasting* or *splatting*, a component of the input vector into all four slots of an SSE register. Then we multiply this register with the appropriate column of the matrix and add the results to the register storing the output vector. The code sample in Listing 2 shows how to transform a vector by a matrix using SSE intrinsics. Note the similarity to the GPU assembly using column-order first storage. We can even further optimize this code if we know that the vector is a position vector ($w=1$), or a direction vector ($w=0$). In the first case, we can avoid the last multiplication and just add the fourth matrix column to the result. In the latter case, we can avoid computing for the fourth matrix column entirely.

Because the GPU and CPU have very different SIMD instruction sets, it can be necessary to use different memory layouts on the CPU and GPU to achieve good performance on both systems. This should not be a problem, because we



Perforce Technical Support Fast Turnarounds. Precise Answers.

Our global support teams are always available to share their expertise in person – no scripted responses, answering services, or dispatch centers. At Perforce Software, our highly experienced technical support engineers take pride in providing fast turnarounds with precise answers.

Keeping your projects on track requires a support team that is ready to help right when you need it. You can count on Perforce's Fast SCM System and legendary technical support to give you the winning advantage.

PERFORCE
S O F T W A R E

Download a free copy of Perforce, no questions asked, from www.perforce.com. Free technical support is available throughout your evaluation.

Listing 4

```

const Vector4 operator*(const Matrix4x4& a_Matrix, const Vector4& a_Vector)
{
    Vector4 result(Utility::NoInit);

    __vector float zero = (vector float)(0.0f, 0.0f, 0.0f, 0.0f);
    __vector float *resultVector = reinterpret_cast<__vector float*>(&result.x);

    __vector float column0 =
        vec_ld(0, reinterpret_cast<__vector float*>(&a_Matrix[0].x));
    __vector float column1 =
        vec_ld(0, reinterpret_cast<__vector float*>(&a_Matrix[1].x));
    __vector float column2 =
        vec_ld(0, reinterpret_cast<__vector float*>(&a_Matrix[2].x));
    __vector float column3 =
        vec_ld(0, reinterpret_cast<__vector float*>(&a_Matrix[3].x));
    __vector float simdVector =
        vec_ld(0, reinterpret_cast<__vector float*>(&a_Vector.x));

    __vector float simdResultVector;
    simdResultVector = vec_madd(vec_splat(simdVector, 0), column0, zero);
    simdResultVector = vec_madd(vec_splat(simdVector, 1), column1, simdResultVector);
    simdResultVector = vec_madd(vec_splat(simdVector, 2), column2, simdResultVector);
    simdResultVector = vec_madd(vec_splat(simdVector, 3), column3, simdResultVector);
    vec_st(simdResultVector, 0, resultVector);
    return result;
}

```

can transpose matrices in memory during GPU/CPU communication. Having to transpose matrices during communication hardly affects the performance of the game, as GPU/CPU communication is often inefficient by itself and a matrix transpose is a relatively inexpensive operation.

MATRIX-VECTOR MULTIPLICATIONS USING SSE4

» The availability of a dot-product instruction in SSE4 gives the programmer more freedom in choosing memory layout. Intel processors with the Intel Core microarchitecture and AMD K10 processors come with SSE4 support. The first processors with a subset of the SSE4 instruction set are the Intel processors with the 45nm Penryn core. Using the dot-product instruction, the programmer can write code that looks similar to GPU code. This eliminates the need for matrix transpositions in shader bindings.

In SSE4, dot-products can be computed using the DPPS [Dot-Product of Packed Single-Precision Floating-Point Value] instruction. The dot-product instruction is very flexible. Using a bit-mask the programmer can specify for which scalars in the SIMD registers the dot-product should be computed. This way 2D, 3D, and 4D dot-products can be computed. The programmer can specify to which scalars in the destination register the result should be written. Scalars that are not written are set to zero. This is unfortunate because it forces us to use an extra add instruction to accumulate the results of all dot-products. Modern GPUs can handle this more efficiently using write-masks (see Listing 3).

A total number of seven instructions are required for the matrix multiplication. This is slightly better than the SSE implementation. Depending on the particular processor architecture, this implementation of a matrix-vector multiplication might run at increased speeds. Possibly even more performance can be gained because this alternative implementation gives the programmer more flexibility in choosing matrix layouts and avoiding data conversions.

MATRIX-VECTOR MULTIPLICATIONS USING ALTIVec

» AltiVec is a SIMD instruction set owned by IBM, Freescale Semiconductor, and Apple. AltiVec is also referred to as VMX (by IBM) or Velocity Engine (by Apple). Interestingly, every major game console from the current generation is equipped with a PowerPC processor with AltiVec support. Microsoft's Xbox 360 has an IBM PowerPC processor with its own version of AltiVec named VMX128. It is not fully compatible with AltiVec, because a number of integer instructions were removed. The code we provide in this section relies solely

on floating-point instructions and should work on Xbox 360. The VMX128 instruction set contains several new instructions specifically designed for accelerating games. The Cell processor used in Sony's PlayStation 3 and the IBM Broadway processor used in Nintendo's Wii also come with AltiVec support.

AltiVec registers are 128 bits wide and can hold several types of values such as characters, short integers, and single precision floating-point values. AltiVec supports unaligned loads, but data should be 16 byte aligned to achieve optimal performance.

As we have shown in the section about SSE, a matrix-vector multiplication can be computed efficiently by computing the linear combination of columns using vector multiply and add instructions. We have seen that this can be computed most efficiently by storing

matrices in column order. Using AltiVec, matrix multiplication can be performed efficiently using the fused multiply-add instruction vec_madd. The following example shows a 4D matrix-vector multiplication using AltiVec (see Listing 4).

The code assumes that both the matrix and the vector are aligned to 16 byte boundaries. Note that the vector is loaded into an AltiVec register. Then, each vector component is splatted and multiplied with a column of the matrix. This result is added to the result vector.

AltiVec also has two horizontal instructions vec_msum, and vec_sums. These intrinsics can be used in conjunction to compute a dot-product efficiently. Unfortunately, these instructions cannot be used for accelerating the typical floating-point matrix-vector multiplications in games, because they support only integer vector operands. For games, we are mostly interested in floating-point matrix multiplications. Therefore, we will not further consider AltiVec matrix-vector multiplication using vec_msum and vec_sums intrinsics. For optimal performance on AltiVec machines, floating-point matrices should be stored column order first. The VMX128 instruction set supported by the Xbox 360 includes a dot product instruction, so row order matrix storage should be efficient on Xbox 360. Refer to the Xbox 360 SDK documentation for more information.

CONCLUSION

» We have described a way to build a matrix implementation that can interface properly with other types of matrices, as well as the details of shader register allocation for matrices: how to save precious register space and how to avoid some of the common pitfalls. You should now be aware of the design trade-offs necessary to achieve optimal performance on modern CPU SIMD architectures such as AltiVec and SSE. We hope we have stuffed your backpack with enough information so you can make the right decisions for your particular codebase and target platforms. ☺

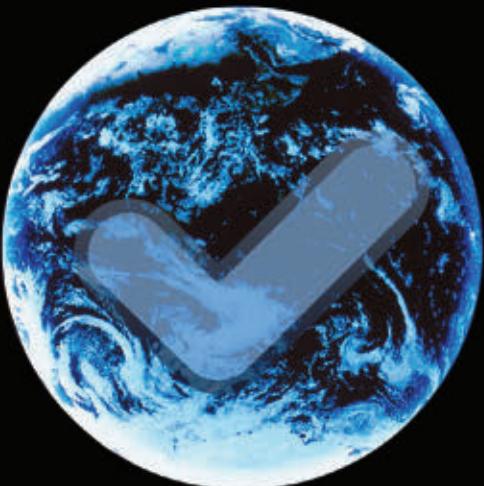
resources

Sutter, H. and Alexandrescu, A., *C++ Coding Standards*, Addison-Wesley (2005)

We would like to thank Willem H. de Boer.

JELLE VAN DER BEEK and ARJAN JANSEN like to make each others lives miserable competing for the best GEOMETRY WARS 2 score. When they're not playing they are probably working on WI Games's proprietary multiplatform game engine. Email them at jvanderbeek@gdmag.com.

**The de facto standard for quality management
for game developers around the globe.**



Don't you think it's time you found out why?



DevTest Studio

The integrated solution for defect tracking, test management and automated testing

DevTest

Use DevTest to manage your testing

- Create a central repository for your test cases, knowledge items and automation scripts
- Schedule releases and test cycles using a wizard-driven interface
- Execute test assignments and submit defects from the same interface
- Track results with real-time dashboards and reports

DevTrack

Use DevTrack to track defects/issues

- Track each issue through a definable workflow
- SCM integration – track fixes against their source code deliverables
- Deploy a resolution across multiple releases, versions and products
- Reporting and metrics to illustrate the entire defect lifecycle

TestLink

Use TestLink to automate your testing

- Add automated tests to the DevTest test library
- Schedule automated tests along with manual tests
- Launch automated tests from the DevTest interface
- Track automation results with real-time dashboards and reports

TechExcel

www.techexcel.com | 1-800-439-7782

WIZARD101

THE IDEA WAS SIMPLE: TAKE A SMALL TEAM AND QUICKLY CREATE AN MMO THAT WOULD APPEAL TO KIDS AND FAMILIES WHO LOVE WORLD OF WARCRAFT, COLLECTIBLE CARD GAMES (CCGS) LIKE POKEMON AND YUGIOH, AND CERTAIN POPULAR WIZARDS. J. TODD COLEMAN, JAMES NANCE, AND JOSEF HALL'S VISION OF THE GAME AND ITS LOVABLE CHARACTERS WAS THE RIGHT IDEA AT THE RIGHT TIME: THEY IDENTIFIED AN UNDERSERVED MARKET, AND DEVELOPED A GAME FOR IT.

Our core team was made up of the usual suspects—promising aspirants and industry veterans from Origin, Midway, and beyond. Josef Hall (senior director of software engineering), Todd Coleman (creative director), and I were founders of Wolfpack Studios, and still proudly wear the scars from SHADOWBANE.

At the time of writing this article, we're almost one year after launch and the response has been phenomenal. A fun game with interesting differences to what's currently available has made for a compelling combo, but it's still gratifying to hear from parents and kids who love the game and educational to hear from those who don't. I've personally learned a lot from this project, and I'm pleased to have the opportunity to share some of that.

WHAT WENT RIGHT

1] RIGHT IDEA AT THE RIGHT TIME. At the time we started the project, Disney's TOONTOWN was the only massively multiplayer online game on the market that

catered to the kids' demographic. Our goal was to address this audience with a product that had more depth.

The market was ripe for a new game, and Todd Coleman's idea of wizardry and CCGs based in a fantasy world was a wonderful framework for a game. Josef Hall, proud parent, made the connection that the kids' and tween market was wide open.

It's always fun to work on an original IP, and we honed the vision through countless brainstorming sessions. The decision was also made to make the game very story driven, with an emphasis on the player being the central character and hero in an epic magical adventure.

Players enter the game as new students recruited by the headmaster to combat a deadly magical threat to all of creation. The idea quickly grew beyond the scope of a school for wizards, and encompasses many different magical worlds. The concept of our different schools of magic mapped perfectly onto types of cards for the CCG.

CONTINUED ON PAGE 24



CONTINUED FROM PAGE 22

We tried to keep the story simple and easy to understand by incorporating a fun cast of characters with an old-school hero's journey. We also wanted to make a game that would appeal to families; Pixar's ability to appeal to parents and kids alike with movies like *Cars*, *Toy Story*, and *The Incredibles* was our shining example.

2] SCOPE, SCOPE, SCOPE. We had a team of fewer than fifty developers at our peak, and an aggressive schedule. We wanted to deliver a top-quality game in under three years. The veterans among us were skeptical, but determined. The only way it worked was to keep the game design tightly scoped; we knew feature creep would kill the project.

We had some leeway in that our target market was fairly free of competitors, so we were able to initially cut features that, in another market, we wouldn't be able to ship without. Anything that wasn't deemed critical to the core game experience was deferred or cut. Guilds, crafting, mounts, player housing, auction houses, grouping, player-versus-player combat—all these things could wait until after launch. We've added most of them in the year since the game opened, but we never would have shipped on schedule if we'd tried to do it all.

Our internal milestones were built around core features taken from the overall game design. We'd identify one or two major features to finish and polish, estimate them, and let those estimates determine the milestone duration. Then we'd fill out the milestone task list with smaller features as time

and resources allowed. Repeating this process in bursts of roughly eight to twelve weeks allowed us to focus on a few features at a time. We drew only from the master design, which kept the scope of the game from growing too much over time.

We didn't launch with as much content as we'd have liked, but you never do. We elected to ship with four major adventure areas and quickly added a fifth area three months after launch. This was a small enough amount of content to allow us to manage the work, but still enough to provide several hundred hours of game play. We recognize that content is vital in MMOs, but you still have to launch the game!

3] PROTOTYPE AND ITERATIVE DESIGN. The idea of a turn-based MMO collectible card game for kids was a bit risky, to say the least. We knew that the card game combat was our core unit of gameplay, so we had to get it right.

Our initial prototype of the combat system consisted of hand-drawn cards (art courtesy of game visionary Todd Coleman), some ten-sided dice, and colored glass beads (for power points and health). We spent hours playing the game against each other [there were no monsters initially], changing card values as we went with a quick erase and pencil scratch iterative approach.

The second prototype was on the computer, with a client and independent server—a multiplayer version with 2D cards and data stored in tables for easy iterations and balancing. Limited A.I. for computer controlled opponents

came later, and served as the basis for our full monster A.I. system.

The critical part of this early work was to see if the basic core gameplay was fun, and to refine the combat rules. Those rules evolved into our current combat resolver. Prototyping was critical to our later success; locking down core gameplay early allowed us to focus on other elements of the game instead of going through multiple project restarts we couldn't afford.

This iterative approach to development was applied to all new systems, though not to the same degree. Each time a new system was brought online, we'd get it functional as quickly as possible and try it out. Feedback was gathered from anyone and everyone in the company, and incorporated.

As the game's development progressed, we also took the opportunity to focus test. Art direction, pricing model, story elements, characters, combat—almost everything was put in front of kids and parents at some point during production. We listened to the customer, and reaped the benefits.

4] DIGITAL DOWNLOAD AND FREE TRIAL.

There was great debate about whether to go retail or direct download, adopt a free-to-play model or give the standard free 30 days. Those of us with shipped MMO game experience were more comfortable with a traditional approach, but our company founder Elie Akillian maintained that digital download was the best way to get our game into the hands of the casual masses. He was right.



GAME DATA	
PUBLISHER	KingsIsle
DEVELOPER	KingsIsle
NUMBER OF FULL-TIME DEVELOPERS	Approximately 40 at peak
NUMBER OF CONTRACTORS	Roughly 30 including QA
LENGTH OF DEVELOPMENT	Six months pre-production, 36 months of full development
RELEASE DATE	September 2, 2008
SOFTWARE	Microsoft Developer's Studio, G++, 3D Studio MAX, Adobe Photoshop
TECHNOLOGY	Gamebryo, Miles, Open Dynamics Engine
PLATFORM	PC/Online



As a new, independent studio, we didn't have the pull of a big studio that is able to demand shelf space and end caps. Our game was fun, but no one had heard of us. The obvious answer was to let the game sell itself, and the best way to do that was to let people try it for free.

Going with direct download had many challenges, however. Even now, we're constantly concerned with download size, since it's a major barrier to getting into the game. Each game update is scrutinized and pared down so that we aren't increasing the download to a new user.

One of our better features is our ability to stream the game to the user. This was a huge technical win for us, and basically means that we can deliver game content to the player just before they need it. We have a small initial download that allows the player to create a new character. While character creation is taking place, the game is downloading the tutorial—while the player is in the tutorial, we are downloading the starting area. Although most players will never notice, it means they don't have to incur a giant download to start playing the game.

Finally, our server architecture needed to be scalable and robust. A free-to-play or free trial game with millions of players coming through needs to be able to handle the load without turning away potential players!

Digital download is a hard road, but considering the millions of players who have given WIZARD101 a try, it was the right choice.

5] MIN SPEC AND SMART TECH. Who still has a GeForce2 in their rig? Who actually uses the integrated video chipset that comes with the motherboard? Who still has less than a gig of RAM? Millions and millions of casual users and kids on hand-me-down machines, that's who.

We did exhaustive research early in the project to try and determine what min spec would allow our target market to play the game, and it was pretty scary. Our research indicated a much higher min spec than what we chose, because all data at the time came from gamers, who typically have much more powerful machines than casual users. We took a gamble and went with a much lower min spec, and it really paid off.

We set a tight budget on polygons and texture sizes for every piece of art in the game, and created our areas to support a fixed number of players so that we could limit the load on the graphics card. Clever use of portalling and other tricks in world building allowed us to hide high-polygon pieces of art, and restrict how many concurrent combats a player could see on screen (another big poly hit).

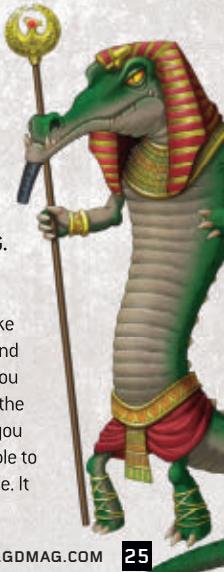
The programming team was very careful about how much data is kept in memory, and spent a lot of time optimizing the code to use

minimal RAM and processing power. Additionally, whenever driver problems arose during compatibility testing we wrote code workarounds so that our casual users would not be faced with the daunting prospect of having to update drivers to play the game. We also knew most casual users and kids would be on older operating systems with outdated service packs and drivers, so we went out of our way to support all that.

Finally, we made the decision to go with a very stylized look to the game. The art style is funny, approachable, and casual, but more importantly, the game looks good on low end machines and will age well, since we aren't competing in the realm of hyper-realistic, bleeding edge graphics.

WHAT WENT WRONG

1] MODULAR WORLD BUILDING. With a small team and aggressive schedule, we made a decision early on that we should take a very modular approach to world and level creation. The idea was that if you use generic building blocks and let the level builders snap them together, you can get a lot of re-use and will be able to create more content for less art time. It



didn't really work—custom areas are better and take just about as much time to create.

We came up with a list of "snappable" pieces (L-shape, T-intersection, end caps, boss rooms, etc.) and had the artists create them for each world with texture variations and decorate them with props appropriate to the different areas. The world designers knew the size of each piece and would create vast adventure area maps on grid paper while the art was being created. In this way, we hoped the designers and artists would be able to work concurrently on the tasks. After the art was created, the designers would snap the pieces together, export them, and the artists would go back for a decoration, polish, and lighting pass. Even though the individual pieces of art were excellent, the end result was fairly generic levels that all looked the same and were boring. The more we re-used pieces, the worse the problem became.

The solution seemed obvious—we would create custom pieces that we could drop in among the generic pieces to provide points of reference to the player in the area and break things up. Examples were gardens with statues, hedge mazes, camps with pavilions, and the like. That really didn't work that well either. Even though the custom pieces looked great, we weren't able to create enough of them to make a difference.

Another approach we tried was to make adjacent areas appear very different by changing the decoration and textures between areas. For example, on Wizard City (our starting world) we themed the adventures areas by element (nature,

fire, ice, etc.) and added icicles, snow, pools of lava, and burning trees to the different zones. That helped some, but it served to hide the problem rather than solve it. The areas still felt very much the same.

For our latest world, Grizzleheim, we finally made it work. We took a totally custom approach to level building; each area was individually conceptualized and designed, then hand crafted by an artist. The result was a much-improved visual appeal, and all the areas combined took about as long to make as it takes to create a set of snappable pieces.

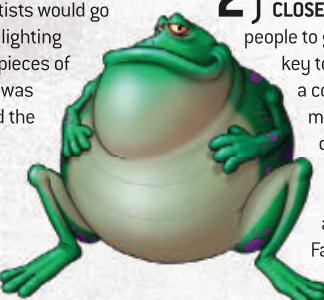
2] WE CHANGED THE BUSINESS MODEL

CLOSE TO LAUNCH. Naturally, getting people to give us money for the game was key to our longevity and success as a company, and so the business model was a hotly debated topic during early production. We finally settled on a subscription model that was family-friendly and had a good price point.

Fairly close to launch, however, we re-opened the subject for discussion and decided to

take a more hybrid approach—we'd allow for both subscribers and micropayment customers. At the same time, we also decided to allow users to play the first part of the game for free. By adding a free trial, we increased the number of players the architecture had to support by an order of magnitude. It's a testament to the scalability of what our engineers built that it was even possible that late in development.

We've seen promising results from catering to users that want to pay us in different ways, but because we chose to offer micropayments fairly



late in the development cycle our implementation was less than ideal. For example, rather than having a micro-payment shop available to the users at the touch of a button, we had to use an automated in-game character as our micro-payment shopkeeper. Players have to find him in-game to be able to make micro-transactions. Additionally, the types and variety of items available for micropayments are limited and not altogether compelling.

Another challenge to using the hybrid approach has been the fine line we have to walk with our users; we want to entice our subscribers to make microtransactions, but we don't want to make them feel like they are getting less value for their monthly payments or being forced to use microtransactions. The approach we've taken is that for every item available in the game for a micropayment, that item is also available in the game by other means—for gold, as a rare monster drop, or as a PvP reward. By doing this we have an answer to our subscribers' concerns about value, but it makes for a lot more data work, is error-prone, and can create game balance issues.

If we had the chance to do it all over again, we would pursue a hybrid business model earlier in development. That way we could have created a much smoother experience and more compelling micro-transaction offerings to the users.

3] USER INTERFACE MISTAKES. To be frank, our graphic user interface is kind of a mess. The GUI is in many ways the face of the game, and supposed to be the user's best friend. It's one of the pieces of the game that speaks to the overall quality of the product, and ours isn't great.

We failed, at the start, to come up with our user interface language—a bible of rules that should make your GUI elegant and intuitive, if followed. As a result, our GUI is often clunky, crowded, and inconsistent. Sometimes buttons are round with icons; sometimes they are square with words. Sometimes we navigate menus with side tabs; other times it's with circular icons at the top of the page. It is critical that you think through how your players will use the interface, and iterate and polish it until it shines—we didn't do that. Although some of our HUDs (deck configuration, for example) did go through dozens of revisions, without a set of established guidelines the result was inconsistent and unpolished.

On the technical side, for reasons beyond our control, we were forced to make the decision to build our own graphical interface system and have struggled with it ever since. We used a homegrown tool for interface layout that was difficult to use and hard to learn. This means that designers and artists had trouble making (and fixing) the GUI, so it fell to programmers to implement HUDs and fix bugs. As a result, programmers spent more time than they should have fixing interface issues, and



our screens lack the visual polish an artist would have provided.

Additionally, our interface elements don't scale and resize with the game's screen resolution. We support 800x600 at the low end, so you can imagine that the HUDs become so small they are almost unusable at high resolutions.

Lastly, our user interface screens are all static; we don't have the capability to animate them, so they seem to lack polish compared to other games made with Flash-based GUIs.

We're currently in the process of migrating our system and all interface elements to Flash, and will soon share the level of quality many of our competitors display.

4) STATS AND METRICS PROBLEMS. The collection, representation and mining of data related to player activities can provide the developer with the keys to tweaking their product to perfection. The trick is to collect enough of the right data, and to make that data accessible to the right audience. If you fail in any of these respects you're in for some headaches. We have headaches.

We weren't sure which facts were going to be important in understanding the success of our game and how we'd need to slice and dice those facts in order to make decisions. We also could have done a better job of making sure the metrics supported the different groups within our company. For example, marketing and operations may both be interested in unique logins, but may require different dimensions—demographically by week for marketing, peak activity hours for operations. We still struggle with asking the right questions and getting the right answers to the people who need them.

Our plan for growth underestimated the amount of data we would need to gather and the number of reports we'd need to run. The activities of millions of players add up quickly. We had some issues scaling our data warehouse with the increasing data set, and had to scramble to keep up. Beyond larger and faster disks, we needed a reporting, retention, and aggregation strategy that would keep our data warehouse manageable after a year of data and billions of facts.

It's not that we weren't warned—our unfortunate data expert told us we needed to make smart choices, but in the heat of making a fun game, we didn't listen. A year after launch, we have a mountain of data, and are having to work very hard to able to parse through it all to see valuable trends and statistics.



5) POOR PLAYER GROWTH STRATEGY. This was a rookie mistake, and we should have known better. MMOs grow over time, and have a lifespan of five to ten years. A smart designer would plan for plenty of room to grow the game and grow the characters along with it. We, however, chose to box ourselves in and make it difficult.

The first basic problem is that we chose to use a percent scale for many of our equipment and advancement modifiers. Accuracy, damage increase, damage resistance, and other attributes lie along on a scale of 1–100 percent [some with caps lower than 100 percent]. This means that we have a hard ceiling on how much power we can award the player through the course of the game.

Here's an example: players use power points (pips) to cast spells in combat, and more powerful spells require more pips. Players gain a pip at the beginning of each turn in combat. As players earn power and equipment, they gain the chance to get a double pip. Here's the sad part—the double pip chance is on a scale of 1–100 percent, and we launched the game with players able to achieve near 90 percent. Given that 100 percent is the max, we have very little room to make more powerful equipment or grow the player beyond what they could achieve when we launched. Additionally the equipment upgrades we designed ended up not being very compelling. With only 100 points to grant, we have upgrades that go from 2 percent Fire Resistance to 3 percent—not very exciting.

The second problem is that we didn't give characters very many attributes. We thought that because we were making a collectible card game, we wanted the majority of player

power to come from collecting cards and building decks. However, when you only have a few attributes on the character, you don't have many ways to create valuable

equipment, so your loot and advancement options become very limited.

The last major problem related to player advancement was that we didn't launch the game with any true boss fights. Really, the only way we had to make fights more difficult was to increase the health of the monsters, which just resulted in longer fights. Additionally, we added some scripted boss fights after launch, and there was a huge backlash from our player community.

Now that we're well past launch and it's quickly becoming time to increase the power scale, we're faced with some difficult challenges. The prospect of building and testing a new equipment and character development scheme is daunting, not to mention re-balancing thousands of pieces of gear. The anticipated community response alone is enough to make me cringe.

CLASS DISMISSED

» Some game projects are sprints, some are marathons. An MMO game project feel like sprinting a marathon. We learned just how much you can accomplish with a small, talented team. We learned there is no substitute for good planning, and that polish happens all the time, not just at the end.

The best thing about an MMO is that it doesn't go away after launch, so we can correct some of the mistakes we've made along the way and apply what we've learned in making the pre-launch product to the live product.

By anyone's standards, Wizard101 is a phenomenal success, and it's absolutely the best project I've ever worked on. There are a few things I'd do differently, and some good lessons learned, but overall it was an immense pleasure to work on such a great game. 

PHOTO BY TOM HALL



JAMES NANCE is the senior producer for *WIZARD101*. His career started in 1991 when Nance joined Origin Systems as a QA tester. He was the lead designer on *SHADOWBANE* and an executive producer at Wolfpack Studios prior to joining KingsIsle. Email him at jnance@gdmag.com.



Canadian-born Mark Rein is vice president and co-founder of Epic Games based in Cary, North Carolina.

Epic's Unreal Engine 3 won Game Developer magazine's Best Engine Front Line Award for three consecutive years, and it is also the current Hall of Fame inductee.

Epic's internally developed titles include the 2006 Game of the Year "Gears of War" for Xbox 360 and PC; "Unreal Tournament 3" for PC, PlayStation 3 and Xbox 360; and "Gears of War 2" for Xbox 360.

Upcoming Epic Attended Events:

CEDEC
Tokyo, Japan
September 1-3, 2009

Austin GDC
Austin, TX
September 15-18, 2009

Tokyo Game Show
Tokyo, Japan
September 24-27, 2009

KGC
Seoul, Korea
October 7-9, 2009

Please email:
mrein@epicgames.com
for appointments.



Unreal Technology News

by Mark Rein, Epic Games, Inc.

MOD TEAMS ON THE RISE IN \$1,000,000 INTEL MAKE SOMETHING UNREAL CONTEST

Epic Games and Intel announced Phase 3 winners of the \$1 Million Intel Make Something Unreal Contest at SIGGRAPH 2009.

What is clearer now more than ever before is that all kinds of unique, high-quality game content can be created with the Unreal Engine 3 toolset – without even touching the engine's source code.

Thanks to the Unreal development community, hundreds of free mods for *Unreal Tournament 3* are available for PC, and many have made their way to PlayStation 3 as well.

The Haunted is a third-person survival horror total conversion that took home top honors for Best Non-FPS Mod, Best New Weapon Set, and Best Graphics in Map, plus first and fifth places in Best Level for a Game Mod.



The Haunted mod for *Unreal Tournament 3*

While anyone can play *The Haunted* in single-player mode, its multiplayer experience shines with four-player co-op, demons versus humans Battle mode, and Demonizer mode.

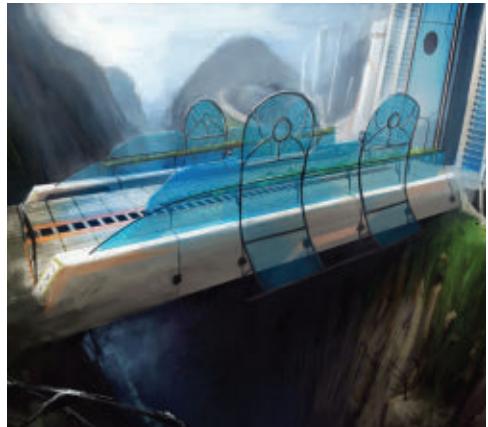
Another breakout mod is *The Ball*, an episodic action puzzle game that won first place for Best FPS Mod, second place for Best Level for a Game Mod, third place for Best Graphics in Map, and fifth place for Best Use of Physics.

The Ball is also a *UT3* total conversion, meaning it uses all original game assets, and like *The Haunted*, it has won recognition in previous phases of the contest.

Angels Fall First: Planetstorm, an assault-style game featuring massive ground and space battles, took home third place in the Best FPS category, fourth place for

Best New Weapon Set, and fifth place for Best Graphics in Map. *Angels Fall First: Planetstorm* is scheduled for completion in October 2009.

Prometheus, which won second place in the Best FPS category, is a community favorite thanks to its clever time travel mechanism in which players collaborate with ghosts of their past selves to solve time-based puzzles.



Sanctum mod for *Unreal Tournament 3*

Sanctum, a total conversion that won fourth place for Best FPS Mod, is a first-person tower defense game with a unique art style. The student developers of *Sanctum* plan to add co-op multiplayer, different game types and additional turret, enemy and weapon types.

UT2D Killing Time, an updated version of the popular *UT2D* side-scrolling mod for PC and PS3, won second place in the Best Non-FPS category.

There are many other mods, including characters, weapons, levels and more that have been recognized at www.makesomethingunreal.com.

Thanks to everyone in the Unreal community who has supported the contest with advice, critiques, content, code and creative genius.

We are extremely grateful to have so many talented developers working with Unreal Engine technology.



For UE3 licensing inquiries email:
licensing@epicgames.com

For Epic job information visit:
www.epicgames.com/epic_jobs.html

W W W . E P I C G A M E S . C O M

GDC

10

Game Developers Conference®

March 9-13, 2010 | Moscone Center, San Francisco

www.GDConf.com

Learn. Network. Inspire.

Updated Tools Spice Up New *Ghostbusters** Game

Terminal Reality used the popular Intel® Graphics Performance Analyzers to bring the newest *Ghostbusters** game to life.

Admit it. For the past 25 years, you ain't been afraid of no ghosts. Thanks to a certain movie about four intrepid heroes, we all know a well-aimed proton stream and a handy trap can bag any ghoul within range. The venerable *Ghostbusters** franchise has spun out at least eight different video games since 1984, each taking advantage of the movie's supernatural feel and sci-fi effects. The newest version *Ghostbusters: The Video Game* has received good reviews since its release earlier in 2009, thanks in no small part to its updated effects.

As Mark Randel, president and chief technology officer of Terminal Reality, Inc. described it in his blog at <http://software.intel.com/en-us/blogs/author/mark-randel/>, "The results of having a massively parallel game engine were stunning. When we finally got rendering and simulation of the game in parallel in the last weeks of *Ghostbusters*, the game became solely render-bound. Jobs were totally asynchronous, and we were able to fully utilize three to four cores. When there wasn't any action in the game, the game was waiting on the vertical blank. With a lot of action, the job model allowed the heavy lifting to be absorbed over as many processors as the system had."

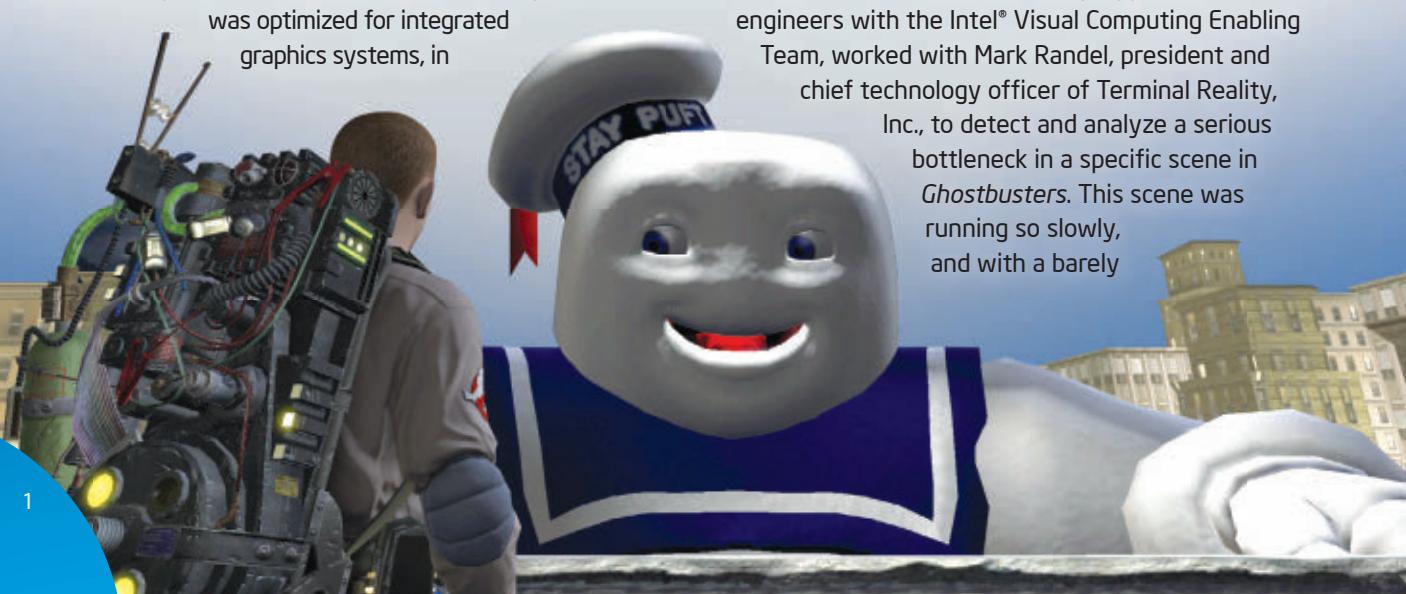
The game is published by Atari, who wanted a great mainstream game to reach the largest possible target market. Atari pushed the team to make sure the game was optimized for integrated graphics systems, in

order to maximize their investment and ensure good performance. The developers at Terminal Velocity took extensive advantage of Intel® Graphics Performance Analyzers (Intel® GPA) and their membership in the Intel® Software Partner Program to bring out the best special effects required to chase down vapors, slimers, and poltergeists. Intel's tools helped identify a performance bottleneck so the game could be optimized for desktops and laptops that use Intel® Graphics processors. And once performance problems are solved for the Intel® Graphics world, they are essentially solved for the rest of the graphics universe.

Thanks to fine-tuning for multi-core and extensive testing for bottlenecks, *Ghostbusters: The Video Game* really shines, especially on the newest Intel-based systems. What follows is a step-by-step analysis performed on an exceptionally low-performing scene in *Ghostbusters: The Video Game* by a team consisting of both Intel and Terminal Reality developers. The team's comprehensive work is a model for anyone who wants to troubleshoot similar game-performance issues.

Optimizing a Slow Game Scene

Jeff LaFlam and Shankar Swamy, application engineers with the Intel® Visual Computing Enabling Team, worked with Mark Randel, president and chief technology officer of Terminal Reality, Inc., to detect and analyze a serious bottleneck in a specific scene in *Ghostbusters*. This scene was running so slowly, and with a barely



acceptable frame rate, that the gameplay was visually stuttering. This scene had stymied progress in optimizing the game's overall performance.

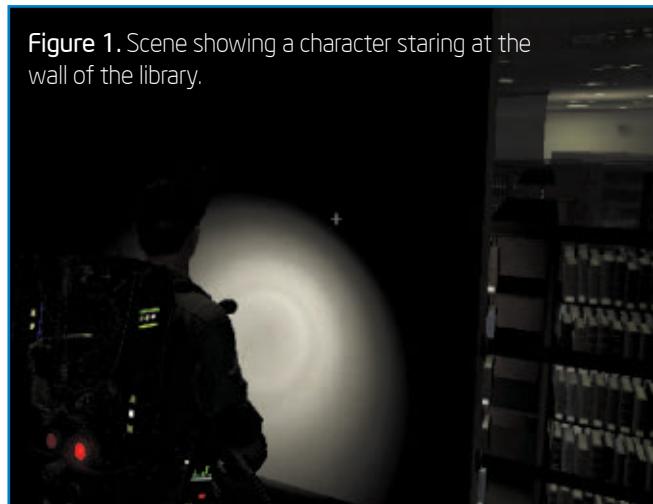
The troublesome scene contains about 200,000 books in a library where two human characters and a "ghost" character might interact. When the characters are fully outside the library they cannot see the books; hence, there is no need for the game to render the books. However, as a character enters the library, the books are gradually exposed to the viewer and displayed in the gameplay scene.

The team of LaFlam, Swamy, and Randel analyzed this scene to determine solutions for increasing the frame rate.

Step 1: Visually Analyze the Scene

The team began by visually analyzing the entire scene sequence in order to determine a direction for further investigation.

Figure 1. Scene showing a character staring at the wall of the library.



The team observed that when a character was staring at the wall and the books were partially exposed, the frame rate was very low and the scene stuttered (Fig. 1). When they then advanced the scene and moved a character closer to the wall but with no books visible in the scene, the frame rate did not change noticeably. This indicated to the team that the books were being rendered in the scene even when they were not visible.

Step 2: Render with Z-Test Disabled

The goal of the second step in the analysis was to determine how many occluded objects were being rendered in the library scene. This was done by rendering all the objects in the scene with the Z-test disabled.

Figure 2. Library scene rendered with the walls forced to render "transparently," showing that the books occluded by the wall are still being rendered.



In Figure 2, notice that the character is standing very close to the wall and staring directly at it. Prior to optimizing this scene, during normal gameplay (with the Z-test enabled), the books shown would not be visible because of the direction the character is looking. However, because the team disabled the Z-test for Figure 2, all the books being rendered by the game are also now visible.

This confirmed that books are being rendered all the time—even when they are completely occluded during normal gameplay. Of course, only the books that are visible to the characters at any point in the game play need to be rendered.

Step 3: Conduct a Single-Frame Analysis

The team wanted to investigate other possible hot spots in the scene by using the Intel® GPA Frame Analyzer.

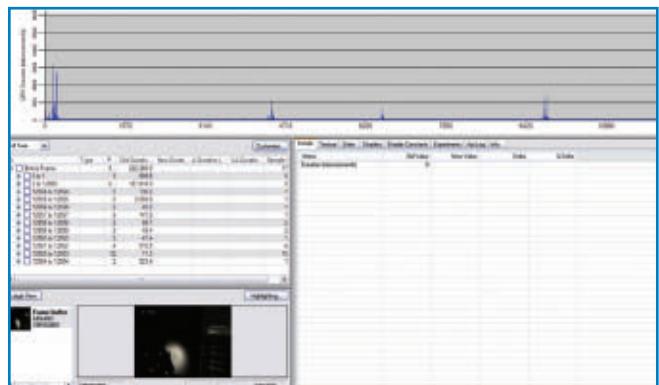


Figure 3. Output from the Intel® Graphics Performance Analyzers Frame Analyzer for the scene inside the library.

According to the Intel® GPA Frame Analyzer, the Library scene had 12,564 Draw() calls (Fig. 3). However, other scenes in the game typically had about 3,000

Draw() calls, and those scenes had higher frame rates. The conclusion was that there were too many Draw() calls in the Library scene, indicating to the team that further testing should be aimed at reducing the number of Draw() calls in the troublesome scene. The team also wanted to investigate how many of these calls were coming from the rendering of the books.

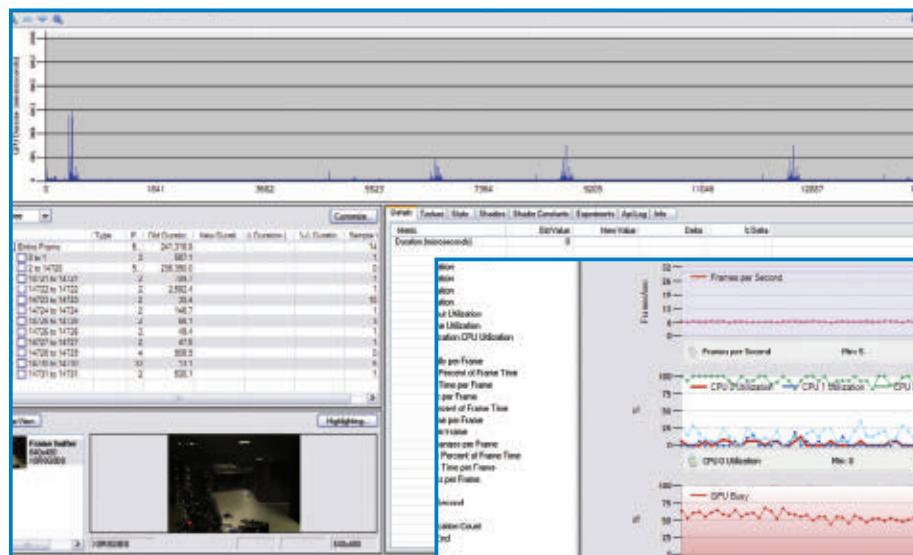


Figure 4. Output from the Intel® Graphics Performance Analyzers Frame Analyzer of the scene inside the library.

Step 4: Estimate the Cost of Rendering the Books

The team placed the camera in front of a wall that had no objects behind it. Because this is a third-person view game, the characters in the Library scene are still rendered—as they should be. However, the books, which are now behind the camera, are invisible and should not be submitted for rendering due to the game’s culling algorithm.

The team wanted a reliable estimate of the cost of rendering the books. By submitting the scene to the Intel GPA Frame Analyzer (Fig.4), the team discovered the scene had 14,731 Draw() calls, confirming that the books were quite expensive to render. In fact, the overhead of rendering the books is



significant enough that it negatively affected the frame rate when the books were occluded yet still rendered.

Step 5: Verify the Potential Gains

Next, the team included a software switch in the graphical user interface (GUI) that allowed them to completely turn off rendering for all the books (whether visible or occluded). They then rendered the scene by dynamically turning this switch on and off, allowing them to determine the change in frame rate when books were rendered versus when they weren’t.

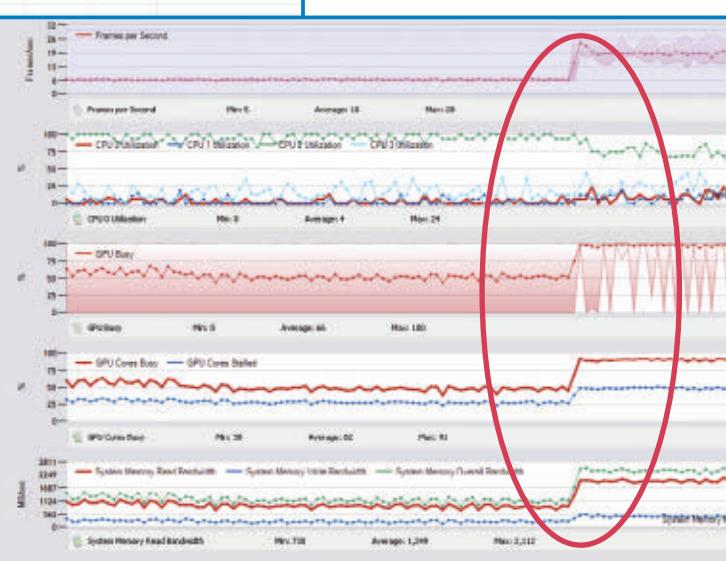


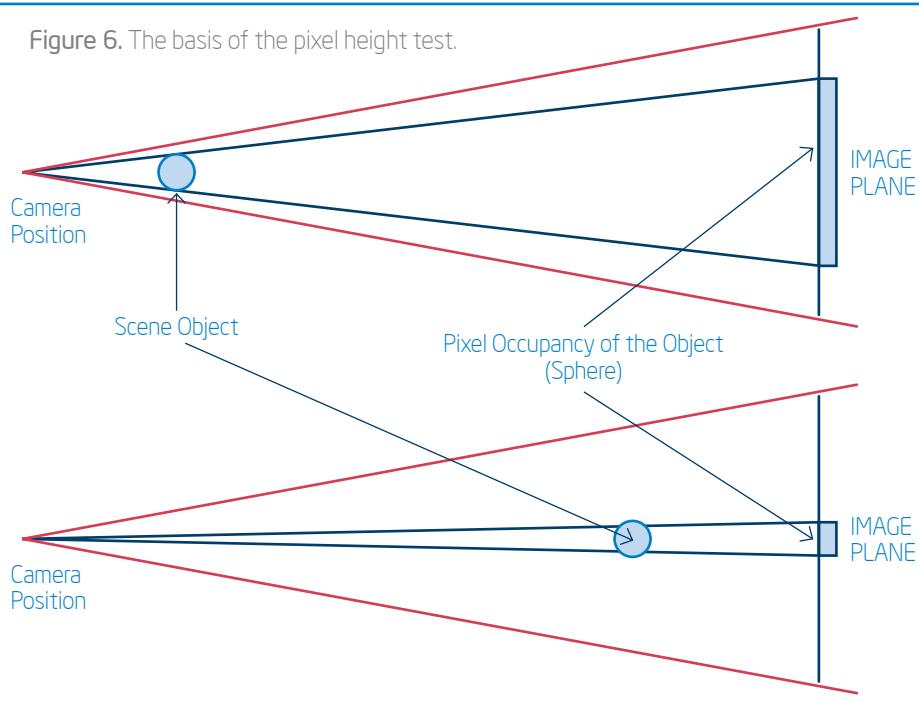
Figure 5. Output from the Intel® Graphics Performance Analyzers System Analyzer with all the books removed from the scene.

When book rendering was turned off, the frame rate increased by approximately 2.5 times, as shown by the data from the Intel® GPA System Analyzer within the red oval in Figure 5. This indicated that the cost of rendering the books in the scene was quite high.

At this point in the analysis, the obvious options for increasing the performance of this scene were either:

- Don’t render the books that aren’t visible in the scene, or
- Reduce the number of books in the scene.

Figure 6. The basis of the pixel height test.



screen space in pixels. This testing code is executed on the processor. As a result of the pixel height test, if the pixel height of an object is less than a pixel, the object is not submitted for rendering.

In the troublesome Library scene, the fact that the objects (books) all had identical dimensions—because they are instantiations of a single object—made the test easier and faster to run because the bounding spheres for all tested objects (books) were identical.

Step 7: The Results of the Pixel Height Test

Figure 7 shows the result of implementing the pixel height test on the Library scene in *Ghostbusters*. Using the software switch created by Randel, developers were able to turn the test on and off. When the pixel height test is running, objects (books) that are less than one pixel in height in the scene, are not rendered. As shown by the data in the green oval in Figure 7, where the test was turned on, the frame rate of this scene doubled when the books less than one pixel in height were not rendered.

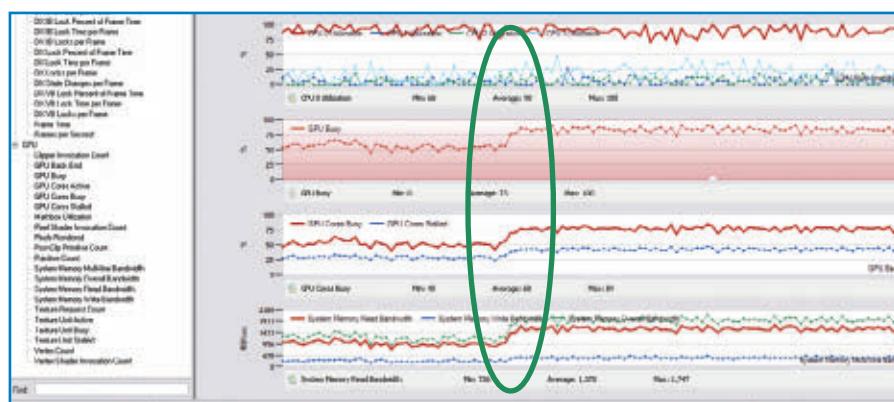


Figure 7. Frame rates with the pixel-height-test solution in place.

Step 6: A Third Solution is Created

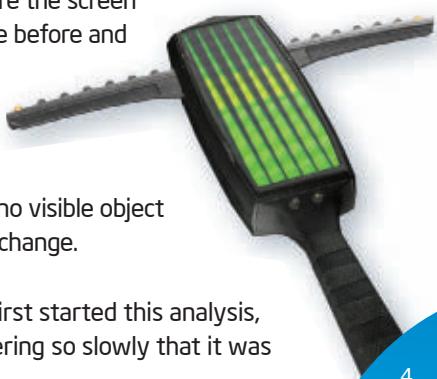
When the Intel team shared their findings with the developers at Terminal Reality, Mark Randel suggested—and implemented—a third solution: a “pixel height test.”

Figure 6 shows the idea behind the pixel height test. The bounding sphere of an object is shown as circles in Figure 6 and indicates the pixel coverage on the screen required for that object either when the object is close to the camera or when it is farther away.

Using the pixel height test on the objects in a scene, the test can determine which objects contribute less than one full pixel to the displayed frame. To approximate the pixel coverage, the test determines the object height in

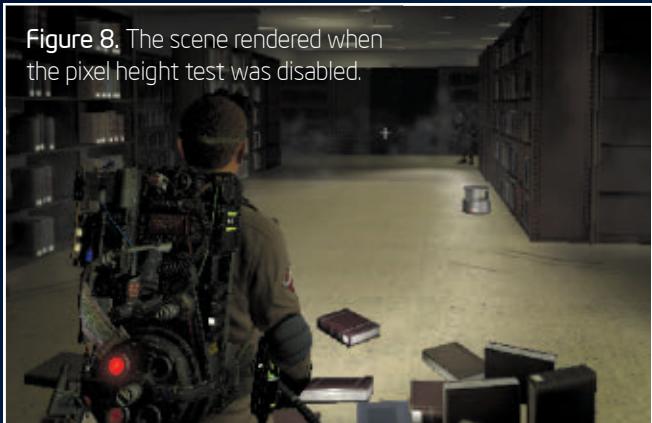
The data in Figure 7 also shows that the overall usage of the graphics resources went up, with the test indicating that the game was now using resources more optimally.

Figures 8 and 9 are the screen captures of the scene before and after the test was enabled. There is no visual difference between the two renderings, because no visible object was affected by the change.



When the team first started this analysis, the scene was rendering so slowly that it was

Figure 8. The scene rendered when the pixel height test was disabled.



considered the major issue preventing the game from being highly playable. Based on a thorough analysis and the implementation of the pixel height test that followed, the scene ended up rendering at double the original frame rate. Other scenes in the game enjoy even higher frame rates.

New Features for Intel® GPA, Version 2.1

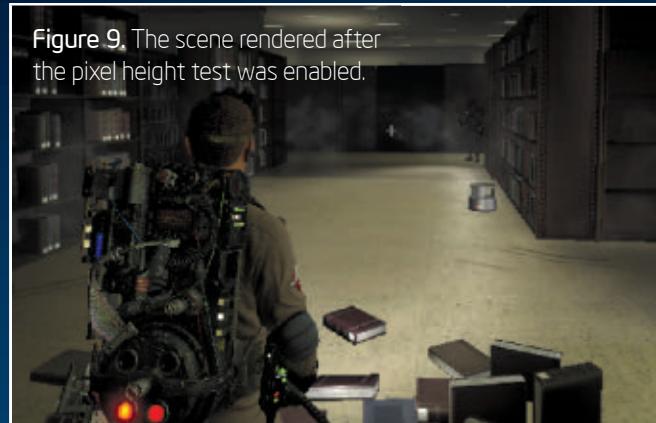
As good as the Intel® GPA tool was for the development of the latest *Ghostbusters* game, several new features have been added subsequent to that project. Randel reports that he is finally enjoying a little downtime after working since 2006 on *Ghostbusters*, but he's already looking forward to the next project. "It will be really nice to have the new Intel GPA tools," he said recently. "There are still a few more things we can do to add those key details to a highly believable, fully destructible environment."

Here are some of the key new features that have been added to the Intel GPA to make it even easier to find and quickly address performance issues in games, as well as debug rendering problems:

Pixel History

Pixel history is a great new feature in Intel GPA that provides a wealth of information on any pixel in any render target. A zoom feature (using the mouse wheel) was also added for a more exact selection of a particular pixel of interest. To select a pixel, simply left-click a pixel in any render target. After a pixel is selected, the history of all GPU operations (draw calls, clears, and so on) that affected that pixel is displayed in the pixel history tab, which is automatically opened. This lets you see exactly which draw calls affected that pixel location for the render target from which it was selected. For each draw call in the list, the number of times the pixel was touched and the final pixel

Figure 9. The scene rendered after the pixel height test was enabled.



color are also displayed. If the pixel was rejected, for example if Z-test was enabled, the reason for the rejection is noted as well.

Pixel history enables two key use cases: visual debug and overdraw analysis. The visual debug workflow allows you to diagnose why a pixel was rendered incorrectly. It also shows which draw call in the history caused the selected pixel to be the color that it is. The overdraw analysis workflow depicts how much overdraw exists at any pixel location and specifically which draw calls contribute to it.

Overdraw Visualization per Render Target

The Intel GPA render target viewer has a new overdraw visualization mode. When enabled, each render target is visualized in gray scale. Overdraw corresponds to lighter pixels in the gray-scale visualization. By enabling this mode, you can immediately see which portions of the render target are being written to most often.

Intel GPA also allows you to combine the usage of both pixel history and overdraw visualization. This allows you to seamlessly find overdraw hotspots with the visualization and then immediately select any of the hot pixels to understand which draw calls are contributing to overdraw at that location.

Vertex Shader and Pixel Shader Durations

Shader durations are now enabled as metrics for all DirectX* devices. These metrics are available in three places: the bar chart graph at the top of the user interface, the scene overview spreadsheet view on the left, and the details tab on the right.

With the bar chart, you can now select any metric in the x- and y-axis. For example, you can configure vertex shader duration in the x-axis and pixel shader duration in

the y-axis. By looking at the shape of each rectangle in the bar chart you now can compare two metrics at the same time. Within the scene overview, you can view these new metrics in spreadsheet form by clicking the Customize button, and then selecting any metrics of your choice. Finally, the details tab always lists all possible metrics and enables you to view their values summed across the current draw call selection set.

Single Step Frame

Intel GPA has a new single step feature that enables better control over the frame to be captured and analyzed. When using the System Analyzer, simply press the pause button to pause the game in real time, then press the single step button as many times as needed to reach a frame of interest. The capture button can be pressed at any time.

In-Game Hot Key

The new hot-key feature allows easy frame captures on a single computer while playing the game. Simply launch the game using Intel GPA, run it full screen, and then press CTRL+SHIFT+C (or configure any keys you want to use) for each frame you want to capture. When you are ready to analyze, close the game, and then open the Frame Analyzer on the same computer or a remote system for analysis.

Export Metrics to a CSV File

With CSV (comma separated value) file export, detailed frame performance data can be saved and later pulled into Microsoft Excel* or any other program that can process CSV files. This feature allows you to track game performance changes over time, compare game performance with various game options enabled, or even compare game performance on various graphics cards—all at a per-draw level of detail.

FOR MORE INFORMATION

To learn more about the Intel® Graphics Performance Analyzers, visit www.intel.com/software/gpa

To download a free version of the Intel® Graphics Performance Analyzers, join the Visual Adrenaline developers program by completing the form at:
<https://ssl.software.intel.com/en-us/register/visual-adrenaline/>

TO GET MORE GREAT ARTICLES LIKE THIS ONE,
 VISIT THE INTEL® VISUAL ADRENALINE WEB SITE:
<http://visualadrenaline.intel.com>

Because this feature is draw call selection set-based, you can select the draw calls you are interested in (or the whole frame) and export only those calls, so you don't have to wade through large amounts of data to find the details you want.

Conclusion

Intel GPA tools help game developers make sure that performance issues don't detract from a game's entertainment value. Developers can run code experiments that measure and report performance results in real time. Intel GPA provides open, accessible libraries that can both customize tools for specific needs and pull data for deeper analysis. Better use of screen real estate avoids the intrusive display overlay of other interfaces, and the ability to share captured frames with team members increases the efficiency of optimization.



Thanks to the Intel® GPA tools, developers can learn more about what's going on "behind the curtain" on their games. The new features take an already strong engineering toolset and turn it into a formidable asset manager. Thanks to interaction with game developers around the world, Intel continues to fine-tune these tools. Priced at USD 299, the Intel GPA tools are free to anyone willing to take the time to fully register. Go to www.intel.com/software/gpa and grab the tools and the documentation, read the case studies and white papers, and get involved in the developer forums. Your game's performance—and fun factor—are at stake. ■

For more details about the breakthrough performance of the Intel® Core™ i7 processor Extreme Edition, visit: www.intel.com/products/processor/corei7ee/

For more information about *Ghostbusters**:
The Video Game, visit <http://www.ghostbustersgame.com/>



Intel does not make any representations or warranties whatsoever regarding quality, reliability, functionality, or compatibility of third-party vendors and their devices. All products, dates, and plans are based on current expectations and subject to change without notice. Intel, Intel logo, and Intel Core are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others. | Copyright ©2009. Intel Corporation. All rights reserved. 08/09/RHM/SM/CS



PIXELACTIVE CITYSCAPE 1.7

REVIEW BY MARC-ANDRÉ GUINDON

WHEN IT COMES TO OPEN WORLD

game development, set design and set dressing are crucial milestones. They reflect the capabilities of your game engine and are instrumental in immersing players in your game's virtual space. Of course, coding entire cities from scratch carries extra time and cost, so for games situated in urban landscapes, why not use a specialized tool to build these essential sets? CityScape 1.7 from PixelActive is exactly that, and is intended to speed up your environment creation tasks.

CITYSCAPE IN A NUTSHELL

» CityScape is focused entirely on quickly creating urban environments as part of your production pipeline. That is to say, the main use for CityScape in the game industry would be for rapid prototyping and design iteration, not as a full-featured level editor.

From villages to full cities [as large as 100km in diameter], this software can manage huge sets assembled from library assets such as buildings, roads, vehicles,

trees, and anything else you might need. Big scenes are manageable thanks to a strong level of detail algorithm. The scene can then be exported to your preferred software and modified to suit your needs, or it can be integrated into commercial or custom real time rendering engines.

USING CITYSCAPE

» When you first open a CityScape scene, you will be presented with a flat world grid. This grid can be used as a canvas on which to paint a height map using different brush types and options. These brushes can either be kept in the scene for later interactive manipulation, or they can be baked into the terrain to make the edits permanent. Alternatively, you could also use a standard greyscale height map to automatically get the proper terrain geometry. It is important to note that any given point can only go up or down, so you cannot create tunnels or caverns. PixelActive informs us that it is

currently at work adding tunnel support for future versions.

You next build on the terrain geometry. Even though CityScape ships with basic assets, you will have to model entire libraries of assets in order to suit your productions. If your scenario is a medieval village or a futuristic metropolis, you will have to create these assets manually in a 3D modeling package, then import them individually or as entire directories into CityScape in order to lay them in the scene.

Importing 3D models in CityScape is done using COLLADA. For instance, if you would like to create a building [using Maya, Max, XSI, SketchUp, Modo, etc.], you would model it normally out of polygons and texture it using file textures and shaders. Once that is done, you can export the models with COLLADA and create a new asset in CityScape.

You can also export a CityScape scene through COLLADA. Doing so is probably imperative for most game studios since touch-ups will need to be done to clean the scene of any glitches before compiling the models for the game engine.

There are several options to export a scene since every studio has different ways of exporting files. PixelActive has been really good at listening and implementing features to make their files compatible with their client requirements. For instance, you can export files compatible with OpenFlight or Gamebryo. You can also export the terrain in square lots using automatically-

generated level of detail. The assets can be exported as full geometry or as locators defining the different attributes and metadata of each asset instances.

One of the bright feature of CityScape is certainly the interactive placement of the assets. At any point during the creation of an environment, you can decide to change the relief of the terrain, move streets, or create a water surface, and the assets will deform and follow accordingly. This is a very important feature while designing a city, since you never know when or where modifications will happen.

There is also a multi-user feature that allows splitting the city into sections so different artists can modify different portions of the city. This comes in very handy when terrain exceeds several kilometers in diameter and is simply too much for a single artist to handle.

SOME ROADBLOCKS

» Prototyping large urban environments in only a few hours is a plus for any production, but CityScape only allows you to generate a terrain and assemble assets together. Any modeling, texturing or animation will need to be done in your preferred 3D modeling package, and any game logic or features will still need to be created in your game engine. However, the software can export procedurally-generated traffic data in XML and the company tells us it is working to include features for adding game data standalone or attached to scene props.





PixelActive Inc. CITYSCAPE 1.7

STATS

PixelActive Inc.
2382 Faraday Ave. #150
Carlsbad, CA 92008
<http://pixelactive3d.com>

PRICE

Single Seat License is \$19,000 and includes 1 year of full support.

SYSTEM REQUIREMENTS

Windows XP [SP2 or higher] or Windows Vista, Pentium 4 2.0 GHz (Pentium 4 2.8 GHz recommended), 1 GB RAM (2 GB or more recommended), 150 MB hard disk space, 128 MB video memory (256 MB or more recommended), video card with support for Shader Model 3.0 or better.

PROS

- 1 Fast learning curve.
- 2 Simple set dressing using libraries.
- 3 Great interactive placement logic.

CONS

- 1 High price (but includes 1 year support).
- 2 Creating unique designs requires more resources.
- 3 Uneven terrain needs manual tweaking.

A recurring problem I experienced while using CityScape involved building cities on uneven terrain, which resulted in a building partially floating above the ground. If your production requires a city on an uneven surface then these situations will require special attention.

Though the basic library offers some good customizable models, the models you will import in CityScape will most likely be static. I was really impressed when I tried building my first demo city since every default asset model had variables that you can tweak to customize the look of each object. Later on, when I created my own assets, I noticed there was no explanation for how to change the model's look, such as per-object textures, procedural blend shapes or optional model visibility (such as using a variety of antenna or door styles for houses).

I would have also appreciated other import and export formats. The PixelActive staff is currently working on FBX file format support, which would greatly simplify the process going in and out of popular 3D platforms.

As a solution to some of my problems, I wanted to turn to the CityScape SDK, but there is no such thing at this time. Any custom solutions will need to be implemented by the PixelActive team and the features will eventually make their way into a future release. I also tried searching for more information or client testimonials on the web, but found that the CityScape community had very limited shared information. This is probably due to the fact that the software is relatively young and the price is limiting the users.

MASS MARKET DEVELOPMENT TOOL? NOT JUST YET ...

► CityScape is pleasingly fast for dressing up environments, but from a gaming point of view it is not a straight "out of the box" solution. While it uses industry standard control schemes and can import and export to a variety of formats, the tool will need to be completely undertaken by the modeling and texturing department and every bit must be customized, including game engine integration.

The CityScape software is relatively new, and is not yet considered a mass market tool. The pricing is steep, but PixelActive includes one full year of support and is committed to custom engineering to support its clients, and will implement any features that could make the product more suitable for a wider range of situations. Buying a license can be thought of as getting the PixelActive team on board rather than only buying a technology.

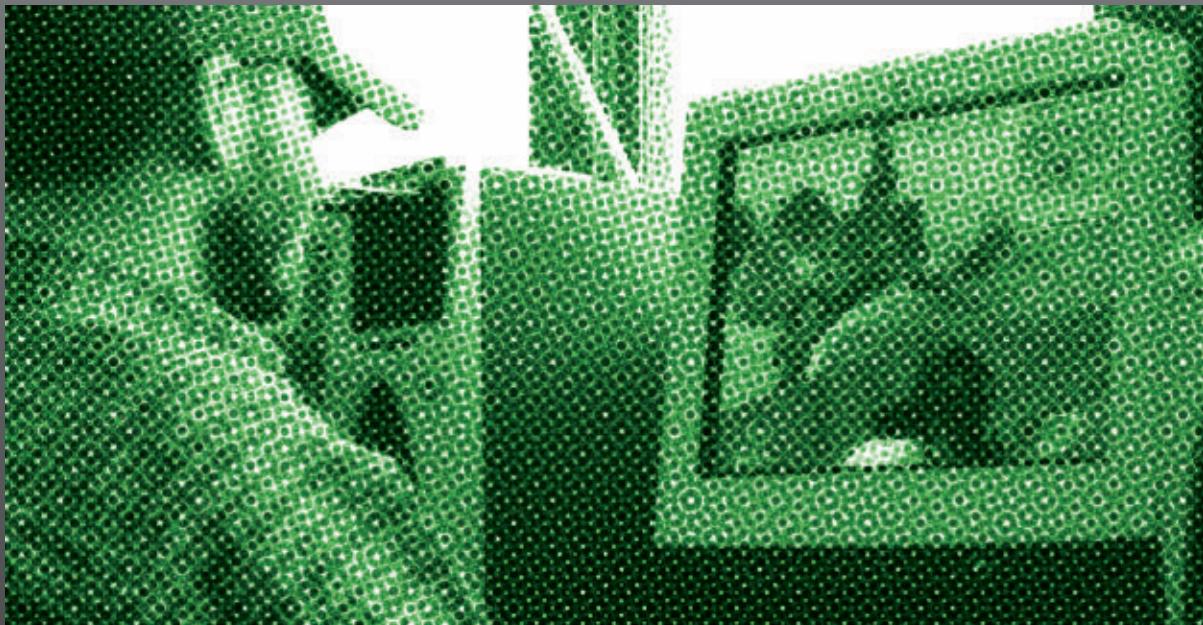
With all the different game studios out there, each with a different way of working, the tool might not do exactly what you would expect. But it's worth considering that the cost involved in building a similar tool for your own usage might be much higher than the price of a CityScape license.

MARC-ANDRÉ GUINDON has been in the visual effects and games industries for 10 years, specializing in pipelines using MotionBuilder and Maya. He is the founder of NeoReel inc., a production and consultation studio located in Montreal. Email him at mguindon@gdmag.com.



GAMASUTRA

The Art & Business of Making Games



**THE LEADING PROFESSIONAL
GAME SOURCE FOR JOB SEEKERS
AND EMPLOYERS**

Access the Web's Largest Game Industry
Résumé Database and Job Board

Take Control of Your Future Today!
Log onto www.gamasutra.com/jobs

THINK
SERVICESTM
A DIVISION OF UNITED BUSINESS MEDIA LLC



JUMPING TO OCCLUSIONS

THE AMBIENCE OF AMBIENCE

WHAT SUMS UP ALL THE

indefinable qualities of a place or time, the mysterious element that binds it all together into a distinct experience with a character of its own? There's a word for it in French. They call it "*ambience*."

Take away the italics and put a nice, nasal American A up front, and it becomes ambience. For those of us in the game business, that means all the light bouncing around in a scene that is not emitted directly from a single light source.

Two fonts, and two completely different words ... but maybe not quite so different as you think. For artists, ambience and *ambience* are almost interchangeable. Ambience plays three key visual roles. First, it is the balance between ambient and direct light that convinces our eyes that a humble LDR monitor can depict everything from a dingy dungeon to a brilliantly sunlit desert. Ambience is also an index of emotional intensity—strong diffuse lighting with little ambience is the classic trick for creating tension [as in a film noir movie], while bright ambience is a stereotypical symbol of relaxation and peace. Finally, and particularly important for games, is the fact that ambient lighting ties the disparate forms in a scene together. Without ambient light bouncing between them, the planes of a 3D scene can look like they were cut and pasted in Photoshop, rather than creating a believable 3D space.

Far too often though, our tools don't capture all this subtlety. Common shading tactics, such as the Phong and Blinn shaders, try to cram it down into a single ambient color parameter. Almost as bad, many games do create beautifully lightmapped environments with nice bounced lighting effects—and then plop characters in front of

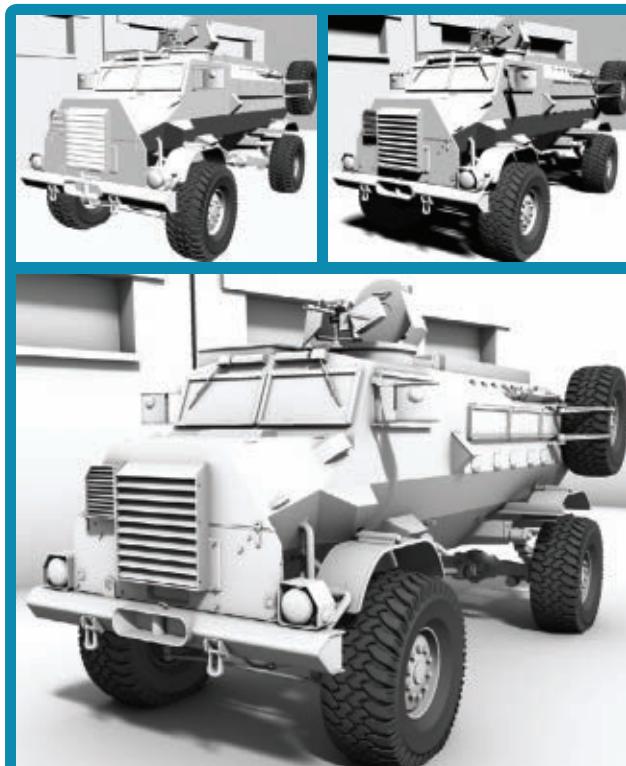


FIGURE 1 This series of pictures illustrates clearly how ambient occlusion helps the eye to pick out detail and understand the 3D space of a scene. The first image at the top left uses only a conventional directional light, and seems visually disjointed. Shadows in the second image [top right] help tie the scene together, but still don't reveal many of the details. Ambient occlusion, in the larger image, clearly picks out important bits of modeled detail and spatial relationships the other images lose.

them with only direct lighting, which makes them to stand out from the background as if they were in a different game. Despite the incredible power of our fancy shaders, it's embarrassingly common to see worlds with neither brand of ambience.

TROMPE L'ŒIL

» To set things right, we need to make deliberate use of the limited tools we do have. Most shaders have some kind of single-value ambience control—this might not be the greatest tool, but far too many

artists simply ignore it. Don't! Even a boring ambient color or ambience coefficient can be very useful.

Ambience control is important for contrasting materials. Typically, we categorize surfaces along a continuum between "shiny" and "matte," but this distinction leaves out differences in ambient response. For example wool and wood are both matte, since neither shows a strong specular highlight. But cloth reacts much more strongly to ambient light. At the microscopic level wool is extremely rough and hairy, so

it catches incoming light from every angle. Wood, being more structured and less chaotic on the microscopic level, picks up ambient light less aggressively. In general softer, rougher and more textured surfaces [for example fabric, concrete, unfinished stone] will have a higher ambient response than smooth, hard, or shiny surfaces. Convincingly capturing these differences makes materials seem rich and physical.

Annoyingly, the tool most of us have to embody these subtleties is a single coefficient value between 0 and 1. This makes sense if you're a graphics card but it's hardly user-friendly. Besides the whole arbitrary-number / technobabble aspect, nothing in the real world would ever have an ambient coefficient of 1—that would mean that the surface only picked up ambient and completely ignored direct lighting. In reality even something like chalk, which is extremely sensitive to ambient light, would still have an ambient coefficient of only about .5. Don't let the slider range fool you! Also, remember that diffuse and ambient lighting work in tandem. Turning up the ambience should mean turning down the diffuse reflectance, or vice-versa—otherwise you'll wash objects out or make them seem to glow.

OOH LA LA!

» The real drawback to the one-slider version of ambience is that it flattens out 3D forms by removing the shading our eyes expect. The effect is inescapably clear in Figure 1. Even though ambient light is coming from all directions at once, it's still light—and thus, it still can be shadowed. Deep folds and crevices don't receive as much ambient light as exposed ones, so they should be darker. The fancy

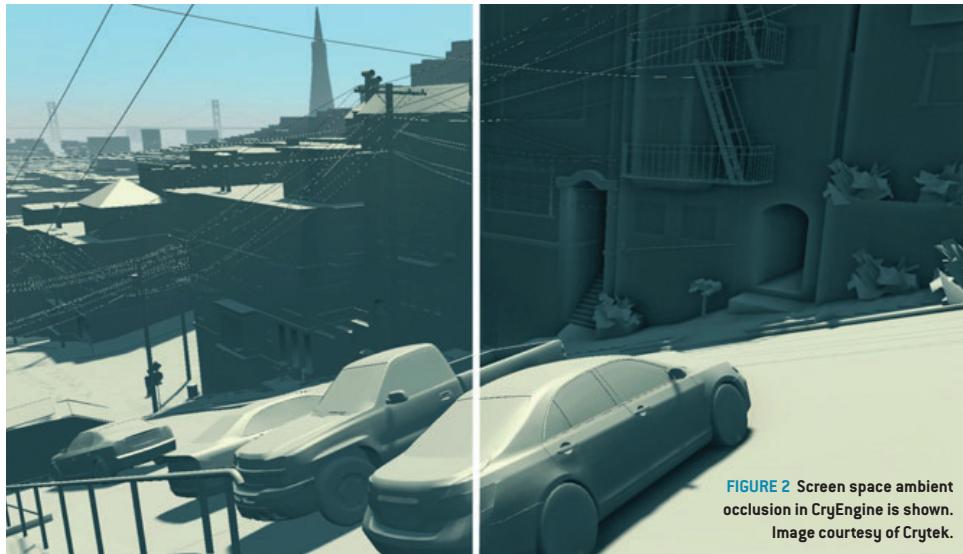


FIGURE 2 Screen space ambient occlusion in CryEngine is shown.
Image courtesy of Crytek.

name for this diminution of ambient light is "Ambient Occlusion."

The results speak for themselves. Ambient occlusion brings out the shading we need, enhancing subtle details in normal maps and clarifying the relationships between forms. There is, as it happens, serious academic research to back up the claim (often heard from artists) that AO makes it easier to pick out detail and understand forms. It should hardly be necessary, though; the results in Figure 2 should be enough to convince any skeptic.

Though the name sounds like something you'd hear on the bridge of the Enterprise during a Klingon attack ("Captain, ambient occlusion is approaching critical! She can't take much more of this!"), the actual concept is something even we artists can grasp. Occlusion simply means "blocking," or "hiding." Ambient occlusion is just a measure of how much (or how little) a given point on an object will be affected by ambient light. In the days before shaders, this was the sort of thing we'd paint into a texture by hand.

Nowadays it's easier to use a render-to-texture function and bake out an ambient occlusion texture. Max and Maya both offer their own versions, though the most popular flavor is probably Mental Ray's. Regardless of the

package, the basic strategy is the same. Imagine that each texel shoots out a hemisphere of rays, and then counts how many of them hit something and how many zip off to infinity. The occlusion of that point is basically the percentage of the rays that were blocked (See Figure 3). Naturally deep folds show lots of occlusion, while exposed surfaces are unblocked. The resulting value is stored as a grayscale map which looks a lot like a soft shadow map.

You can tell that ambient occlusion is a neat idea, simply because it looks so nice. Raw ambient occlusion maps have an ethereal quality like the carefully lit statues in a coffee table book. It's not surprising that AO renders instantly became the standard mode for modelers' demo reels; the broad range of tones and the nicely defined details sell themselves. Even so, to get the most out of AO you need to pay careful attention to your setup.

The most crucial element in any AO render is the dropoff distance. Its job is to control how far a ray goes before it is assumed to be unblocked. Large dropoffs will produce very soft, faint, smooth shadows. Shorter dropoffs make for tighter, clearly-defined areas of dark shadows alternating with large blank areas.

can make an object seem too large and may suppress details you want to emphasize. It's critical, though, that whatever settings you pick are aligned between different characters, objects, and environments. The whole point of ambient occlusion is bringing different elements together into a coherent environment, so getting too individualistic with the settings undercuts the whole enterprise.

CUL-DE-SAC?

➤ The smooth soft tones of a large dropoff setting have an appealing painterly quality. Unfortunately, they also emphasize the biggest weakness of current-generation AO: since it's pre-calculated, it's not dynamic. For environments that's not a big deal, but for moving or deforming objects, particularly characters, it's a serious limitation.

In the short term, you should be careful to avoid creating ambient maps that will emphasize the fact that the occlusion isn't static. For example if you're casting occlusion maps for a tank, you might want to omit or paint out the occlusion from the gun barrel, since that would leave a visible shadow on the deck of

Setting dropoffs isn't just a matter of taste. Dropoffs also influence your audience's perception of scale. If you set your dropoffs too large so that everything in your model occludes everything else, you get a very attractive set of soft shadows, but your model tends to read as if it were physically very small, rather like a close up photo of a tabletop miniature figure. Conversely, very short dropoffs

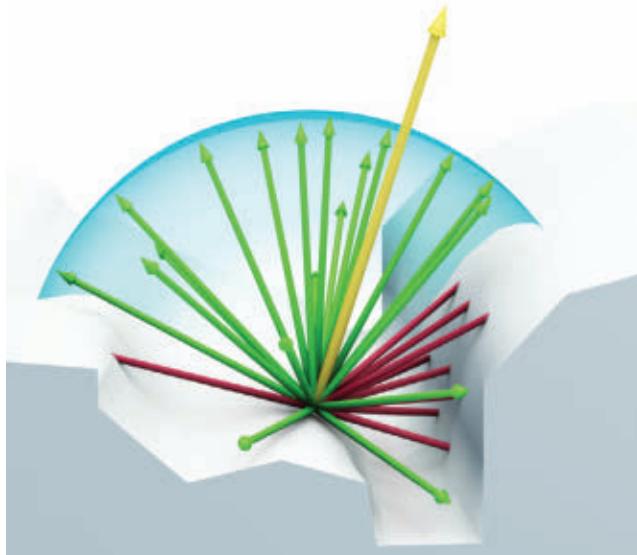


FIGURE 3 A simple example of how ambient occlusion is calculated: From a given point on the surface shoot a bunch of rays out in a hemisphere around the normal (yellow) and see how many hit geometry (red) within a specified cutoff (blue). In this case the point will be shaded about 20%, since about 4/5ths of the rays don't hit anything.

FIGURE 4 This series of screenshots illustrates the potential of real-time radiosity—a step up from ambient occlusion that we should be seeing in games soon. Images courtesy of Geomerics.



the tank even when the turret was traversed. Shorter dropoffs will also help limit potential embarrassments.

For characters, casting in a spread eagle pose will help prevent arms and legs from casting too strong a shadow. When working with paper-doll characters that can add or remove equipment, you should consider adding shadow cards to objects to make sure they blend in properly when added to a character that is otherwise nicely tied together by AO. And of course, don't be afraid to hand edit an AO map to get rid of an annoying artifact—unlike a normal map, AO values are simple grayscales that require no special processing.

ENFANTS TERRIBLES

» For now, good-looking ambient light requires a lot of attention and tweaking. In the near future though, these tricks may become obsolete. CRYYSIS has already shipped with a clever new technique that applies ambient occlusion in real time to an entire image. Screen Space Ambient Occlusion, or SSAO for short, allows deforming characters and moving objects to occlude correctly in ways that pre-rendered AO maps cannot. For the time being the detail can't compete with the polished results of an offline mental ray occlusion bake. But that's beside the point: the immersive power of having everything on screen showing live occlusion is enormous. Lots of other engines are scrambling to add support for SSAO, so expect this to be a common feature in the near future.

Another extremely interesting technology that bears on ambience is a new middleware lighting engine from Geomerics, which advertises radiosity in real time. The possibility of doing a full radiosity simulation in real time sets the heart of any lighting enthusiast on fire. The screenshots in Figure 4 show how nicely the pervasive ambient lighting ties scenes together. Most importantly, there's no question that lighting will be better when lighting artists count their iteration times in seconds rather than hours.

It's nice to think that this under-appreciated but supremely important aspect of graphics is finally getting some serious love from the whiz-bang brigade. With a little attention to detail, and some help from our friends in engineering, we can finally wring some serious *ambience* out of our ambience. In the meantime, au revoir. 

[For more on the relationship between direct and ambient light, check out "Let There Be Light," in the March-April 2005 editions of *Game Developer*, Megan Fox and Stuart Crompton's "Ambient Occlusive Crease Shading" in the March 2008 *Game Developer*, or Jeremy Birn's *Texturing and Lighting*.]

STEVE THEODORE has been pushing pixels for more than a dozen years. His credits include *MECH COMMANDER*, *HALF-LIFE*, *TEAM FORTRESS*, and *COUNTER-STRIKE*. He's been a modeler, animator, and technical artist, as well as a frequent speaker at industry conferences. He's currently content-side technical director at Bungie Studios. Email him at stheodore@gdmag.com.



MONTRÉAL INTERNATIONAL
GAME SUMMIT 09

MOOSE SCHMOOZE & DO'S

> NOVEMBER 16 & 17
AT THE HILTON MONTREAL BONAVENTURE HOTEL

DON'T MISS OUR KEYNOTES:

YOICHI WADA ■ PRESIDENT & CEO · SQUARE ENIX

PAUL HOLDEN ■ LEAD ARCHITECTMEDIA · MOLECULE

JASON HOLTMAN ■ DIRECTOR OF BUSINESS DEVELOPMENT · VALVE

HEATHER CHAPLIN ■ AUTHOR

CHRIS HECKER ■ TECHNOLOGY FELLOW · MAXIS/EA



REGISTER NOW!
SIJM.CA

EARLY BIRD ENDS
SEPT 25TH

STAY TUNED!
**DETAILED PROGRAM
IS AVAILABLE ONLINE**

FOR MORE INFORMATION,
VISIT SIJM.CA OR CONNECT
WITH US ON FACEBOOK
& TWITTER!

ORGANIZED BY:



PARTNERS:





DATA-ORIENTED DESIGN

OR WHY YOU MIGHT BE SHOOTING YOURSELF IN THE FOOT WITH OBJECT-ORIENTED PROGRAMMING

PICTURE THIS: TOWARD THE END OF THE DEVELOPMENT CYCLE, YOUR GAME crawls, but you don't see any obvious hotspots in the profiler. The culprit? Random memory access patterns and constant cache misses. In an attempt to improve performance, you try to parallelize parts of the code, but it takes heroic efforts, and, in the end, you barely get much of a speed-up due to all the synchronization you had to add. To top it off, the code is so complex that fixing bugs creates more problems, and the thought of adding new features is discarded right away. Sound familiar?

That scenario pretty accurately describes almost every game I've been involved with for the last 10 years. The reasons aren't the programming languages we're using, nor the development tools, nor even a lack of discipline. In my experience, it's object-oriented programming (OOP) and the culture that surrounds it that is in large part to blame for those problems. OOP could be hindering your project rather than helping it!

IT'S ALL ABOUT DATA

» OOP is so ingrained in the current game development culture that it's hard to think beyond objects when thinking about a game. After all, we've been creating classes representing vehicles, players, and state machines for many years. What are the alternatives? Procedural programming? Functional languages? Exotic programming languages?

Data-oriented design is a different way to approach program design that addresses all these problems. Procedural programming focuses on procedure calls as its main element, and OOP deals primarily with objects. Notice that the main focus of both approaches is code: plain procedures (or functions) in one case, and grouped code associated with some internal state in the other. Data-oriented design shifts the perspective of programming from objects to the data itself: The type of the data, how it is laid out in memory, and how it will be read and processed in the game.

Programming, by definition, is about transforming data: It's the act of creating a sequence of machine instructions describing how to process the input data and create some specific output data. A game is nothing more than a program that works at interactive rates, so wouldn't it make sense for us to concentrate primarily on that data instead of on the code that manipulates it?

I'd like to clear up potential confusion and stress that data-oriented design does not imply that something is data-driven. A data-driven game is usually a game that exposes a large amount of functionality outside of code and lets the data determine the behavior of the game. That is an orthogonal concept to data-oriented design, and can be used with any type of programming approach.

IDEAL DATA

» If we look at a program from the data point of view, what does the ideal data look like? It depends on the data and how it's used. In general, the ideal data is in a format that we can use with the least amount of effort. In the best case, the format will be the same we expect as an output, so the processing is limited to just copying that data. Very often, our ideal data layout will be large blocks of contiguous, homogeneous data that we can process sequentially. In any case, the goal is to minimize the amount of transformations, and whenever possible, you should bake your data into this ideal format offline, during your asset-building process.



Because data-oriented design puts data first and foremost, we can architect our whole program around the ideal data format. We won't always be able to make it exactly ideal (the same way that code is hardly ever by-the-book OOP), but it's the primary goal to keep in mind. Once we achieve that, most of the problems I mentioned at the beginning of the column tend to melt away (more about that in the next section).

When we think about objects, we immediately think of trees—inheritance trees, containment trees, or message-passing trees, and our data is naturally arranged that way. As a result, when we perform an operation on an object, it will usually result in that object in turn accessing other objects further down in the tree. Iterating over a set of objects performing the same operation generates cascading, totally different operations at each object (see Figure 1a).

To achieve the best possible data layout, it's helpful to break down each object into the different components, and group components of the same type together in memory, regardless of what object they came from. This organization results in large blocks of homogeneous data, which allow us to process the data sequentially (see Figure 1b).

A key reason why data-oriented design is so powerful is because it works very well on large groups of objects. OOP, by definition, works on a single object. Step back for a minute and think of the last game you worked on: How many places in the code did you have only one of something? One enemy? One vehicle? One pathfinding node? One bullet? One particle? Never! Where there's one, there are many. OOP ignores that and deals with each object in isolation. Instead, we can make things easy

for us and for the hardware and organize our data to deal with the common case of having many items of the same type.

Does this sound like a strange approach? Guess what? You're probably already doing this in some parts of your code: The particle system! Data-oriented design is turning our whole codebase into a gigantic particle system. Perhaps a name for this approach that would be more familiar to game programmers would have been particle-driven programming.

ADVANTAGES OF DATA-ORIENTED DESIGN

» Thinking about data first and architecting the program based on that brings along lots of advantages.

Parallelization. These days, there's no way around the fact that we need to deal with multiple cores. Anyone who has tried taking some OOP code and parallelizing it can attest how difficult, error prone, and possibly not very efficient that is. Often you end up adding lots of synchronization primitives to prevent concurrent access to data from multiple threads, and usually a lot of the threads end up idling for quite a while waiting for other threads to complete. As a result, the performance improvement can be quite underwhelming.

When we apply data-oriented design, parallelization becomes a lot simpler: We have the input data, a small function to process it, and some output data. We can easily take something like that and split it among multiple threads with minimal synchronization between them. We can even take it further and run that code on processors with local memory (like the SPU's on the Cell processor) without having to do anything differently.

Cache utilization. In addition to using multiple cores, one of the keys to achieving great performance in modern hardware, with its deep instruction pipelines and slow memory systems with multiple levels of caches, is having cache-friendly memory access. Data-oriented design results in very efficient use of the instruction cache because the same code is executed over and over. Also, if we lay out the data in large, contiguous blocks, we can process the data sequentially, getting nearly perfect data cache usage and great performance.

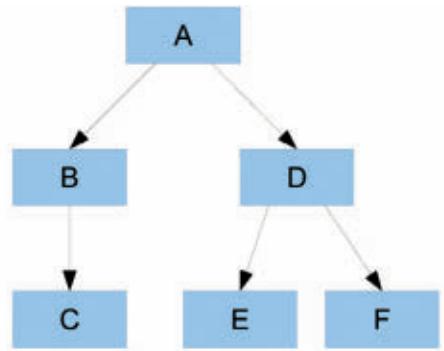
Possible optimizations. When we think of objects or functions, we tend to get stuck optimizing at the function or even the algorithm level; Reordering some function calls, changing the sort method, or even re-writing some C code with assembly.

That kind of optimization is certainly beneficial, but by thinking about the data first we can step further back and make larger, more important optimizations. Remember that all a game does is transform some data (assets, inputs, state) into some other data (graphics commands, new game states). By keeping in mind that flow of data, we can make higher-level, more intelligent decisions based on how the data is transformed, and how it is used. That kind of optimization can be extremely difficult and time-consuming to implement with more traditional OOP methods.

Modularity. So far, all the advantages of data-oriented design have been based around performance: cache utilization, optimizations, and parallelization. There is no doubt that as game programmers, performance is an extremely important goal for us. There is often a conflict between techniques that improve performance and techniques that help readability and ease of development. For example, re-writing some code in assembly language can result in a performance boost, but usually makes the code harder to read and maintain.

Fortunately, data-oriented design is beneficial to both performance and ease of development. When you write code specifically to transform data, you end up with small functions, with very few dependencies on other parts of the code. The codebase ends up being very "flat," with lots of leaf functions without many dependencies. This level of modularity and lack of dependences makes understanding, replacing, and updating the code much easier.

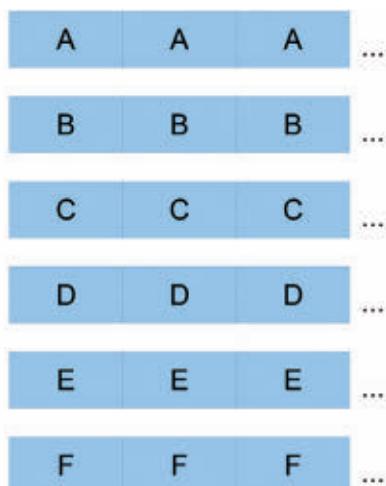
Testing. The last major advantage of data-oriented design is ease of testing. As we saw in the June and August Inner Product columns, writing unit tests to check object interactions is not trivial. You need to set up mocks and test things indirectly. Frankly, it's a bit of a pain. On the other hand, when dealing directly with data, it couldn't be easier to write unit tests: Create some input data, call the transform function, and check that the output data is what we expect. There's nothing else to it. This is actually a huge advantage and makes code extremely easy to test, whether you're doing test-driven development or just writing unit tests after the code.



Call sequence:

A, B, C, D, E, F, A, B, C, D, E, F,
A, B, C, D, E, F, ...

FIGURE 1A Call sequence with an object-oriented approach



Call sequence:

A, A, A, ..., B, B, B, ..., C, C, C, ...,
D, D, D, ..., E, E, E, ..., F, F, F, ...

FIGURE 1B Call sequence with a data-oriented approach

DRAWBACKS OF DATA-ORIENTED DESIGN

» Data-oriented design is not the silver bullet to all the problems in game development. It does help tremendously writing high-performance code and making programs more readable and easier to maintain, but it does come with a few drawbacks of its own.

The main problem with data-oriented design is that it's different from what most programmers are used to or learned in school. It requires turning our mental model of the program ninety degrees and changing how we think about it. It takes some practice before it becomes second-nature.

Also, because it's a different approach, it can be challenging to interface with existing code, written in a more OOP or procedural way. It's hard to write a single function in isolation, but as long as you can apply data-oriented design to a whole subsystem you should be able to reap a lot of the benefits.

APPLYING DATA-ORIENTED DESIGN

» Enough of the theory and overview. How do you actually get started with data-oriented design? To start with, just pick a specific area in your code: navigation, animations, collisions, or something else. Later on, when most of your game engine is centered around the data, you can worry about data flow all the way from the start of a frame until the end.

The next step is to clearly identify the data inputs required by the system, and what kind of data it needs to generate. It's OK to think about it in OOP terms for now, just to help us identify the data. For example, in an animation system, some of the input data is skeletons, base poses, animation data, and current state. The result is not "the code plays animations," but the data generated by the animations that are currently playing. In this case, our outputs would be a new set of poses and an updated state.

It's important to take a step further and classify the input data based on how it is used. Is it read-only, read-write, or write-only? That classification will help guide design decisions about where to store it, and when to process it depending on dependencies with other parts of the program.

At this point, stop thinking of the data required for a single operation, and think in terms of applying it to dozens or hundreds of entries. We no longer have one skeleton, one base pose, and a current state, and instead we have a block of each of those types with many instances in each of the blocks.

Think very carefully how the data is used during the transformation process from input to output. You might realize that you need to scan a particular field in a structure to perform a pass on the data, and then you need to use the results

to do another pass. In that case, it might make more sense to split that initial field into a separate block of memory that can be processed independently, allowing for better cache utilization and potential parallelization. Or maybe you need to vectorize some part of the code, which requires fetching data from different locations to put it in the same vector register. In that case, that data can be stored contiguously so vector operations can be applied directly, without any extra transformations.

Now you should have a very good understanding of your data. Writing the code to transform it is going to be much simpler. It's like writing code by filling in the blanks. You'll even be pleasantly surprised to realize that the code is much simpler and smaller than you thought in the first place, compared to what the equivalent OOP code would have been.

If you think back about most of the topics we've covered in this column over the last year, you'll see that they were all leading toward this type of design. Now it's the time to be careful about how the data is aligned [Dec 2008 and Jan 2009], to bake data directly into an input format that you can use efficiently [Oct and Nov 2008], or to use non-pointer references between data blocks so they can be easily relocated [Sept 2009].

IS THERE ROOM FOR OOP?

» Does this mean that OOP is useless and you should never apply it in your programs? I'm not quite ready to say that. Thinking in terms of objects is not detrimental when there is only one of each object (a graphics device, a log manager, etc) although in that case you might as well write it with simpler

C-style functions and file-level static data. Even in that situation, it's still important that those objects are designed around transforming data.

Another situation where I still find myself using OOP is GUI systems. Maybe it's because you're working with a system that is already designed in an object-oriented way, or maybe it's because performance and complexity are not crucial factors with GUI code. In any case, I much prefer GUI APIs that are light on inheritance and use containment as much as possible (Cocoa and CocoaTouch are good examples of this). It's very possible that a data-oriented GUI system could be written for games that would be a pleasure to work with, but I haven't seen one yet.

Finally, there's nothing stopping you from still having a mental picture of objects if that's the way you like to think about the game. It's just that the enemy entity won't be all in the same physical location in memory. Instead, it will be split up into smaller subcomponents, each one forming part of a larger data table of similar components.

Data-oriented design is a bit of a departure from traditional programming approaches, but by always thinking about the data and how it needs to be transformed, you'll be able to reap huge benefits both in terms of performance and ease of development. ☺

Thanks for Mike Acton and Jim Tilander challenging my ideas over the years and their feedback on this article.

NOEL LLOPIS has been making games for just about every major platform in the last ten years. He's now going retro and spends his days doing iPhone development from local coffee shops. Email him at nllopis@gdmag.com.

resources

Performance and Good Data Design by Mike Acton (www.cellperformance.com/mike_acton/2006/04/basic_principles_of_good_data.html)

SPIEL™ studios

GAME DEVELOPMENT PARTNERS
COST EFFECTIVE | HIGH QUALITY | TIMELY DELIVERY

SPECIALIST IPHONE DEVELOPERS

- 2D Game Development
- 3D Game Development (Unity 3D)
- Application Development
- Published Several Titles

Approved Developers for:

- iPhone / iPod Touch
- Android G1
- PSP

DEVELOPMENT / PORTING SOLUTIONS FOR MOBILE & ONLINE GAMES

2D/3D Art Development | Handheld & Flash Game Development

Special Attractive & Economical Rates Partner with Us, Today!
Developers of Award-winning games.

E-mail: info@spiel-s.com Website: www.spiel-s.com

ACTIVISION®

GREAT GAMES

— START WITH —

GREAT PEOPLE



ARE YOU READY TO EXPLORE THE MOST SOUGHT AFTER JOBS IN ENTERTAINMENT?

ACTIVISION
MINNEAPOLIS

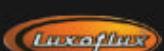
SHABA GAMES



Vicarious Visions®

RADICAL ENTERTAINMENT

FreeStyleGames



Visit our site: activision.com



RESONANCE

DOES YOUR GAME HAVE LASTING APPEAL?

THERE ARE SONGS THAT ARE KIND of catchy. Others you just can't get out of your head. And then, ever so rarely, there are those songs which are so memorable, you could swear you had heard them before. Song taste is highly personal—different people react to different songs in different ways—but the breakout hits are the ones that resonate on this level with a large number of people.

Creating a breakout hit is no easy task, in part because the songwriter's instincts can often be wrong. Steven Tyler of Aerosmith was reportedly surprised that "St. John" from *Permanent Vacation* was met by collective yawns from concert-goers. He thought he knew what made a hit—the song had an interesting riff, topical lyrics, was meaty to play live—but somehow just didn't reach the fans. Today, the song is a footnote in the band's music catalog.

All creative fields are like this. Sometimes films and books just catch fire from nothing. Sometimes, sure-handed directors stumble. Pop radio is full of songs like "The Macarena" and "I'm Too Sexy," all done by bands that later proved to be one-hit wonders unable to repeat their success. This is true of games as well.

BASIC RESONANCE

» Resonance—the idea that some art is simply more immediately arresting and intriguing than other art—exists in games as it does in film and music. But how much of a black art is it, really? Is resonance something that can be willfully added, shaped, and controlled? Or is the concept that some games just stick better than others best left to luck and fate?

Most people reading this magazine are probably pretty comfortable with the idea that

games are fun and sell well for quantifiable reasons having to do with good design, technique, and craftsmanship. The idea that part of game design is left to fate can be somewhat unnerving. But this can be a trap—the idea that there is a formula to good art is seductive, but it also ignores the subjective nature of art. Sometimes, a book, a movie, or a game just feels good the first time the players get their hands on it.

In his book *Blink*, author Malcolm Gladwell cites several studies that point out most people decide whether they like or dislike something in seconds, if not nanoseconds. Whether a song lingers is often decided by the opening bars. Whether a film resonates can often be decided before the opening credits finish. Whether you're going home with the girl at the bar can often be decided before you blurt out your pickup line.

THE FIRST 15 SECONDS

» It's no secret among game designers and executives that we have a very short period of time in which to earn a player's trust and attention. What may be underestimated is how short that time period is. Producers love to press other developers about the "first five minutes of gameplay," when in actuality, a customer may only give the game designer 15 seconds. The player may play beyond that, but by then his initial impressions are set and must be overcome. If Gladwell is right, then it's useful for the game developer to obsess over whether those first 15 seconds are resonating with test audiences. Think that's too short? Consider the fact that the viral hit YouTube video of a prairie dog turning his head to the camera is eight seconds long.

I once heard of a programmer who, in an interview asking what he was most proud of, said he crunched and worked overtime for a month in order to get the jumping exactly perfect in his last game. His interviewers were not impressed by his answer, but I am. What is a player more likely to do in his first 15 seconds but to run, and then jump?

Good, smooth movement is the cornerstone of many games, especially platform games.

Having smooth jumping increases

comfort—feeling comfortable with the setting and mechanics and his role as quickly as possible. If getting into the game feels to the player like he is slipping into a comfortable pair of shoes, the game designer has probably successfully built resonance. This is one reason why licenses are so attractive to game designers, but even those working with original designs can leverage this.

ALPHA CENTAURI was a solid and needed evolution to the

Pop radio is full of songs like "The Macarena" and "I'm Too Sexy," all done by bands that later proved to be one-hit wonders unable to repeat their success. This is true of games as well.

the chance of resonance. More importantly, bad jumping is incredibly dissonant—if players feel the movement controls in the first 15 seconds are clumsy or sluggish, they are likely to extend this prejudice to the game as a whole. The designer then has to work harder to overcome these initial judgments.

A game designer's job can be thought of as trying to build resonance, and whenever possible, to remove game aspects that are dissonant for the player. There are undoubtedly elements that are impossible to predict or ascertain, but some aspects are certainly within the control of the game developer and should not be left to pure chance.

RESONANCE OF FAMILIARITY

» So what is resonance composed of? One cornerstone is a certain level of player

gameplay found in its predecessor, CIVILIZATION II. It offered more depth and strategy than previous CIV games, while still streamlining it in ways the classic CIV design needed streamlining. And yet, when I played it, I mostly felt an urge to find my old CIV disks.

ALPHA CENTAURI was more polished, more streamlined, prettier, and more atmospheric as a whole, but I just couldn't get into the game the way I could into CIV. I found it's easier to get excited over discovering writing, building catapults, and crushing the Greek Empire than discovering applied gravitronics, building super tensile solids and crushing the human hive. The former mean something to me and my life. These ideas have resonance, and they grant that to the game.

A similar example in gaming can be found in EVERQUEST vs. ASHERON'S CALL. EVERQUEST's



Sega's 1991 hit SONIC THE HEDGEHOG continues to resonate in the minds of players almost two decades after its release.

developers chose to populate their worlds with standard D&D fare—trolls, orcs, gnomes, and dragons. By contrast, ASHERON'S CALL went to great pains to create an entirely invented bestiary—no orcs and trolls here, instead players fought creatures with names like Mattekars, Lugians, and Mosswarts. AC may have won points for originality, but for EVERQUEST players, most of whom were playing an online RPG for the first time, the familiar setting and enemies undoubtedly made the game feel like a sort of homecoming.

A HINT OF NEW

» But familiarity can be (and often is) taken to a fault. Right now, there are musicologists diligently working on algorithms to detect whether any given pop song will be a megahit—and trying to define algorithms to write the next hit song. Some musicians are concerned about this—I'm not. Could the result be anything other than formulaic?

The games industry is sequel-heavy—it is one of the few creative fields where sequels frequently outearn the original—but at the same time, the market demands novelty.

Players want new ways to interact with their old classics, and games with little new to offer are viewed as glorified expansion packs.

At the same time, if the player loves a game or a genre, they don't look kindly on adjustments to the classic game design they see as a step backwards. Players, essentially, want new features that feel like they should have been there the whole time. Studios that trade in sequels, such as EA's MADDEN team, are acutely aware of this delicate balance, and take great pains to try to find the franchise's logical extensions. Sometimes, they stumble—the Quarterback Vision feature from MADDEN '06 was received by many fans as making their beloved game more difficult and wonky to play. But sometimes, they score—when they announced the ability to manage your MADDEN team from the web this E3, the first words out of my mouth were "that's so obvious!" If you find yourself saying "that's so obvious!", it's very likely you've found a new feature with resonance.

CHANGING TIDES

» Tastes are always changing as well. What resonates today may

not have that sort of impact in the future. "Spirit in the Sky" was a monster hit in 1969, selling more than 2 million copies, and it was named one of the top 500 songs of all-time by *Rolling Stone*. Would the song have nearly the impact if it had been released last year? Doubtful.

Tastes change just as quickly in the game space as well—ask any adventure game fan. What's more, the skills of game players tend to graduate as well. Competitors to Blizzard hoping to make the next great WoW-killer have a tricky balancing act to achieve. On one hand, you want the gameplay to be familiar and inviting to the WoW population to maximize resonance. On the other hand, though, WoW is now four years old, and even devoted fans are now eagerly looking ahead to the next logical evolution of the genre.

This has some unexpected side effects. Modern FPS players who go back and play CASTLE WOLFENSTEIN are often shocked at how far the genre has come—since then, the genre has added full 3D environments, multiplayer, jumping, crouching, rolling, cover, alternative fire modes, and full physics simulations. What the

designer has to be wary of is the opposite—that the player who has never played a first-person shooter now must learn all at once the skills other FPS players have learned over 20 years. Any time a game has a steep learning curve, the barrier to resonance is all that much higher.

DISSECTING RESONANCE

» Still, resonance has a large intangible component to it. Personally, I find it intriguing to consider the one-hit wonders and unusual hits. What was it about "Song 2" that Blur could never replicate? Why did KATAMARI DAMACY catch on? Even more intriguing is the internet meme. Did "All Your Base" really merit exploding into the public consciousness? Lessons of resonance abound.

Building games that stick is a black art, but not unteachable. Designers striving for resonance should learn to balance the familiar with the new, be obsessive about the first 15 seconds of gameplay, and do everything to remove dissonant gameplay elements, especially early in the game experience. And don't just trust your gut—developers are too close and familiar with the game to be objective about it. Run playtests as early and often as you can.

I remember a designer on GUITAR HERO saying, shortly after the game shipped, that the studio had no idea if the game would succeed. The series has since sold more than 25 million units and Activision claims it to be the third largest franchise in video game history. It's easy to see why—the game just resonates. ☀

DAMION SCHUBERT is the lead combat designer of STAR WARS: THE OLD REPUBLIC at BioWare Austin. He has spent nearly a decade working on the design of games, with experience on MERIDIANS9 and SHADOWBANE as well as other virtual worlds. Damion also is responsible for Zen of Design, a blog devoted to game design issues. Email him at dschubert@gdmag.com.

Register with code:
GDCCHINA09
and receive 10% off
Attendee and VIP Passes*

GDC China

Game Developers Conference® China

October 11–13, 2009

Shanghai International Convention Center
Shanghai, China

Reach a unique vertical market of Chinese industry professionals in online games, game outsourcing, and mobile games at the Game Developers Conference® China's second developer event.

Visit www.GDCChina.com
for more information.

*Discounts cannot be combined with other GDC China discounts or other promotions including group registrations and alumni discounts. Discount may be applied to Early Bird rates. GDC China reserves the right to review and end this promotion before the end of online registration.

Supported
by



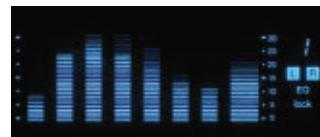
www.GDCChina.com

THINK
SERVICESTM
A DIVISION OF UNITED BUSINESS MEDIA LLC



STEREO DOWNMIXING

HOW DOES YOUR GAME SOUND ON ENTRY-LEVEL SETS?



CATERING A GAME MIX FOR THE WIDE

variety of audio systems available to consumers is a huge challenge. There is a plethora of surround configurations these days, such as Dolby Digital, Dolby Pro Logic II, DTS (and now DTS Neural 7.1), and the PS3's many discreet PCM surround streams. This vast array of choices ensures that the audio team mixing a game has some difficult decisions to make. The sheer number of settings and listening configurations that an end user is capable of using severely complicates the mix decisions that need to be made at the end of production for the game. At best, a game will be mixed in a 5.1 calibrated mix suite, but with significant attention paid to what happens when the game is heard through TV speakers or a stereo receiver and speaker system. But as the "HD world" evolves, attention to such detail is becoming an exception rather than a rule. Many development studios are not well-enough equipped to mix high-end content, and some gamers with stereo televisions are missing out on a lot of really great audio, much of which has high gameplay value.

There has been a lot of independent research commissioned to get to the bottom of exactly what system gamers use for audio playback, to better understand the priorities for mixing decisions. All the research that I have seen (but sadly am not at liberty to divulge) shows differing results based on the slightly different nature of the questions asked. I can however give ballpark numbers, and roughly 40 percent of console owners connect their consoles to home theatre systems [5.1 or surround-enabled], while around 10 percent own a home theatre sound system but don't actually have their console hooked up. The remaining majority, 50 percent, are listening to stereo only. This does not mean that support for the higher end systems should be in any way de-prioritized, but at the very

least more consideration needs to be afforded to the stereo mix.

Having recently experienced the high end of surround game mixing while mixing *PROTOTYPE* for DTS Neural 7.1 on the Xbox 360 and PS3's discreet PCM 7.1, it became critical to check and tweak the mix on all the different "lower" supported configurations, testing both stereo fold-down as well as 5.1 fold-down mixes regularly in order to maintain the general balance of sounds on all possible systems. In hindsight, it occurs to me now that rather than compromise the amazing potential and immersion of a 7.1 mix, or even the 5.1, the best thing to do is to provide unique mixes for all three configurations and play to the strengths of each mix. As far as I am aware, full separate in-game mixes have not yet been attempted in video games, but are certainly just around the corner, at least in terms of both separate 5.1 and stereo mixes. Providing richer options for the gamer seems to be the way to go, and *GTA IV* recently demonstrated how easy it is to offer differing surround set-ups in its audio options screen, allowing for choice of rear or side surround positioning.

One approach to catering for both surround and stereo consumers has been to provide two versions of any offline surround content—for example, cutscenes or music tracks—so that both a surround version and a specially remixed stereo version of those assets sit on the disc simultaneously. Extra assets mean added disc footprint, so instead of just a stereo music track, you now have a separate six channel surround music track taking up space, even more so if you factor in multiple localized languages on the disc. Several games have already pushed the boundaries in this regard by providing both stereo and 5.1 surround assets that get

switched depending on the outputs used by the console. Most recently *HEAVENLY SWORD*, *SACRED 2*, and *WIPEOUT HD* have gone down this route and make a clear statement of how important supporting a unique stereo mix is to certain developers.

However, one could certainly argue, in light of the fact that the majority of gamers listen only in stereo, that if extra effort is being put into the cut-scenes and music, then even more effort should be put into a completely different mix of all the in-game sounds that are authored in 3D surround in real-time by the game engine. This is because

GAMES THAT USE SEPARATE SURROUND AND STEREO ASSETS

BURNOUT 2, 3, and 4
PS2 Stereo, PS2 Dolby PL2,
Xbox Stereo and Xbox Dolby
Digital

HEAVENLY SWORD
separate 5.1 and stereo
mixes for cut scenes,
ambiences etc

SACRED 2
separate 5.1 and stereo
music mixes

WIPEOUT HD
separate 5.1 music mixes

in-game 3D sounds are considerably more problematic and unpredictable when downmixed than pre-authored music or cinematics.

While downmix algorithms are designed to represent surround audio as accurately as possible through only two speakers, they are mainly tailored to movie mixes, particularly in their modest use of surround channels. In games, certain sounds, particularly in-game 3D sources, can behave unpredictably when summed. Games use surrounds extensively, often mixing in 3D source sounds at the same output levels of the front

lefts and rights to further maintain immersion in the 3D world. While this approach works well when heard in surround and makes the most of surround channels to immerse the player in the 3D world, in stereo those "rear" sounds are down-mixed back into the front left and rights, and often end up summing far louder than any movie mix, drowning out important dialogue or even more important events in front of the player. This approach then, of supporting two separate in-game mixes, would have the benefit of bringing consistency to all the content (both cinematics, music and in-game sound) in a video game, and ensure that the levels are heard the way the sound designers intended, whether the game is enjoyed on a high-end surround sound system or a stereo television.

Theatrical movie soundtracks are often given exactly this kind of careful attention when presented in the home environment via a specialized DVD remix. With games, the switching needs to be similarly dynamic and intelligently implemented, fully catering to high end surround systems as well as the most basic stereo TV speakers.

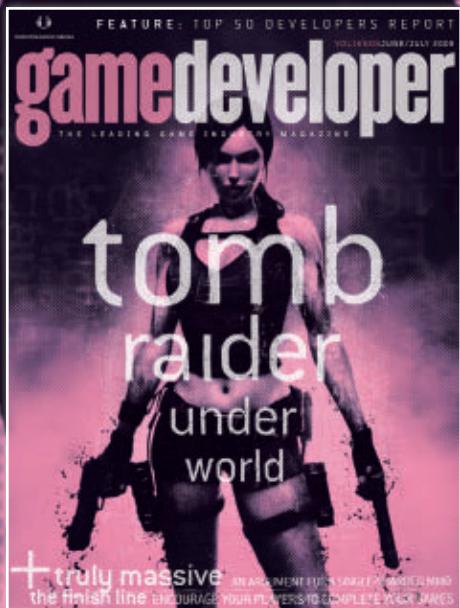
It's true that providing separate mixes requires more time and attention at the end of production, time that is set aside specifically to mix and balance the sounds in the game. That time is already at a high premium, and rarely available the closer the developer gets to the gold master date. This places more emphasis and pressure on having meticulously well-planned and well-executed post-production audio phase at the end of development, but the gains in quality, for all gamers, regardless of their set ups, can simply no longer be ignored. 

ROB BRIDGETT is senior audio director at Radical Entertainment in Vancouver. Recent titles include *50CENT: BLOOD ON THE SAND*, *PROTOTYPE*, and *SCARFACE: THE WORLD IS YOURS*. You can email him at rbridgett@gdmag.com.

ACCESS IT IN REAL TIME

game developer

DIGITAL EDITION



Game Developer magazine's 6 month and 1 year Digital Edition subscriptions give you new issues delivered promptly, full searchable access to more than 50 back issues, plus downloadable PDF versions, easy access from any web browser, and more!
Subscribe today!

[HTTP://WWW.GDMAG.COM/DIGITAL](http://WWW.GDMAG.COM/DIGITAL)

THINK
SERVICES



Dave Bolton – BFA Production Animation.



2009 IGF Best Student Game award goes to TAG: The Power of Paint.

→ **DIGIPEN INSTITUTE OF TECHNOLOGY**

5001 150th Ave. NE
Redmond, WA 98052
Toll-free: 866.478.5236
Email: admissions@digipen.edu

WWW.DIGIPEN.EDU

"Over the past decade, Valve has continuously hired individuals and teams from DigiPen who are ready and equipped to make material contributions to professional projects immediately out of school."

Kathy Gehrig
HR Director
Valve

"Because of DigiPen's focus on game design/development and real-world approach to the game development cycle, graduating students have a clear leg up on other college students seeking jobs in the games industry."

James Pfeiffer
Test Manager
Microsoft Games



Education, Location, Imagination

Since 1989 DigiPen has been helping students prepare for careers in the electronic gaming and animation industry with a formula of world-class faculty, an ongoing tradition of academic excellence, hands-on experience and strong industry connections.

Top Five Reasons to Choose DigiPen

RECOGNITION To date our students have been recognized 25 times at the Independent Game Festival – more than any other school in the world. We are also the only school to have finalists in the IGF's Main professional category. Over the years, DigiPen has also been covered by numerous news sources from *Wall Street Journal* to *Rolling Stone*. *Electronic Gaming Monthly* also ranked DigiPen as the #1 school in the world for Game Development.

LOCATION The Redmond/Seattle area is home to over 150 game-related companies. For many of these companies, DigiPen is the preferred choice for recruiting new employees. This allows DigiPen students and graduates unparalleled access to networking, internships and job opportunities. Class of 2008 DigiPen programmers had a 91% employment rate in fields utilizing their degrees. In 2007 DigiPen accepted the invitation of the Singapore government to open a second campus in Singapore to facilitate that country's growth as a leader in the interactive digital media industry in Asia.

CONNECTED DigiPen has the world's first game development degree program. In the last two decades, DigiPen has garnered a strong reputation as an established, quality institution. We place our graduates and interns at Microsoft, Nintendo, ArenaNet, Valve, Bungie, Sony, Monolith, Big Fish Games, LucasArts, Rhythm & Hues, Intel, Activision and many others.

HANDS-ON DigiPen education goes beyond theory to applied. It's not enough that our students learn about games – they have to make them. Every year at DigiPen, students must form into groups for a project class. Whether they are developing a game for competition at the IGF or an animated short film, the students must create everything themselves, from complete game engines and AI to storyboarding and sound development.

SPECIALIZED Specializing your education should never mean compromising your education. Our faculty are 92% full-time and are comprised of PhDs and experienced game and animation industry leaders. Our students spend a full four years in their chosen program. This allows a depth of knowledge far beyond that of a traditional school. As a result, undergraduate interns from DigiPen have worked on published AAA title games, assisted in Microsoft research projects and they have helped to develop the device drivers for the Wii.

We invite all prospective students to see first hand why DigiPen is considered among the very best of game development degrees. Prospective students are welcome to shadow our students, sit in on classes and attend a monthly information session in person or online.

EPIC LOOT FOR GAME DEVELOPERS THAT WANT TO REACH HIGHER LEVELS

INCENTIVES that reduce expenses and increase profitability are available in the Baton Rouge area.

WE OFFER CASHABLE TAX CREDITS AT THE FIXED RATE OF:

► **25% OF PRODUCTION COSTS**
expended in Louisiana

► **35% OF LABOR COSTS**
in Louisiana, paid to Louisiana residents



BRADIC

Baton Rouge Area Digital Industries Consortium

To find out more about the **HIGHEST DIGITAL MEDIA INCENTIVES IN THE COUNTRY**, visit us at GDC Booth #713.
CONTACT: Chad Cornett at 225-381-7136 • BRADIC.ORG



THINK! SERVICES

GET YOUR GAME ON! EVERYTHING YOU NEED TO KNOW TO GET INTO THE GAME INDUSTRY!

- News and features for students and educators
- Getting Started section – an invaluable how-to guide
- Message Boards



DIGITALCOUNSELOR

I'LL MATCH YOUR INTERESTS and GOALS WITH THE RIGHT GAME RELATED PROGRAMS AND SCHOOLS FROM AROUND THE WORLD.

www.gamecareerguide.com

Download your FREE digital edition of the 2009 gamedeveloper Game Career Guide online at:
www.gamecareerguide.com



Create the Game

Gaming Degree Programs for the Next Generation

Game Art

Bachelor's Degree Program

Game Development

Bachelor's Degree Program

Game Design

Master's Degree Program



©2008 Full Sail Inc.

ANIMATION | DESIGN | ENTERTAINMENT BUSINESS | FILM | RECORDING ARTS | SHOW PRODUCTION | VIDEO GAMES | WEB

Master's | Bachelor's | Associate's Degrees

800.226.7625 • 3300 University Boulevard • Winter Park, FL 32792
Financial aid available to those who qualify • Career development assistance • Accredited University, ACCSCT

fullsail.edu

Online Programs Available



FULL SAIL
UNIVERSITY

MAKE MORE ENEMIES

Game Design at Vancouver Film School shows students how to make more enemies, better heroes, cooler levels, and tighter connections to the industry.

In just one year, you'll learn every aspect of game design. Your portfolio project is a playable video game.

VFS grads get snapped up by top companies like BioWare, Radical, Relic, and Ubisoft, and the *LA Times* named VFS a top 10 school "most favored by video game industry recruiters".



vfs.com/enemies

VFS student work by
Moo Won Kim



[GEEKED AT BIRTH.]



You can talk the talk.
Can you walk the walk?
Here's your chance to prove it.
Please geek responsibly.

LEARN:

ADVANCING COMPUTER SCIENCE	NETWORK ENGINEERING
ARTIFICIAL LIFE PROGRAMMING	NETWORK SECURITY
DIGITAL MEDIA	OPEN SOURCE TECHNOLOGIES
DIGITAL VIDEO	ROBOTICS & EMBEDDED SYSTEMS
ENTERPRISE SOFTWARE DEVELOPMENT	SERIOUS GAME & SIMULATION
GAME ART & ANIMATION	TECHNOLOGY FORENSICS
GAME DESIGN	VIRTUAL MODELING & DESIGN
GAME PROGRAMMING	WEB & SOCIAL MEDIA TECHNOLOGIES

www.uat.edu > 877.UAT.GEEK

CENTER FOR DIGITAL IMAGING ARTS AT BOSTON UNIVERSITY

imagination capture

3D ANIMATION + INTERACTIVE MEDIA
GAME ART + CHARACTER ANIMATION
certificate programs

TWO CAMPUS LOCATIONS
WALTHAM, MA & WASHINGTON, DC

Financial assistance and career services available.
Now accepting applications. *Apply today!*

cdia
BOSTON UNIVERSITY
CDIABU.COM

ADVERTISER INDEX

COMPANY NAME	PAGE	COMPANY NAME	PAGE
Activision46	Intel	30–35
AutodeskC3	Perforce Software	19
BRADIC53	RAD Game ToolsC4
CCP14	Seapine Software11
Center for Digital Imaging Arts55	Spiel Studios45
Digipen52	TechExcel21
Epic Games6, .28	University of Advancing Technology ..	.55
EyetronicsC2	Vancouver Film School54
Full Sail University54	Xaitment3
Havok13		

Game Developer (ISSN 1073-922X) is published monthly by United Business Media LLC, 600 Harrison St., 6th Fl., San Francisco, CA 94107, (415) 947-6000. Please direct advertising and editorial inquiries to this address. Canadian Registered for GST as United Business Media LLC, GST No. R13288078, Customer No. 2116057, Agreement No. 40011901. **SUBSCRIPTION RATES:** Subscription rate for the U.S. is \$49.95 for twelve issues. Countries outside the U.S. must be prepaid in U.S. funds drawn on a U.S. bank or via credit card. Canada/Mexico: \$69.95; all other countries: \$99.95 (issues shipped via air delivery). Periodical postage paid at San Francisco, CA and additional mailing offices. **POSTMASTER:** Send address changes to *Game Developer*, P.O. Box 1274, Skokie, IL 60076-8274. **CUSTOMER SERVICE:** For subscription orders and changes of address, call toll-free in the U.S. (800) 250-2429 or fax (847) 647-5972. All other countries call (1) (847) 647-5928 or fax (1) (847) 647-5972. Send payments to *Game Developer*, P.O. Box 1274, Skokie, IL 60076-8274. For back issues write to *Game Developer*, 4601 W. 6th St. Suite B, Lawrence, KS 66049. Call toll-free in the U.S./Canada (800) 444-4881 or fax (785) 838-7566. All other countries call (1) (785) 841-1631 or fax (1) (785) 841-2624. Please remember to indicate *Game Developer* on any correspondence. All content, copyright *Game Developer* magazine/United Business Media LLC, unless otherwise indicated. Don't steal any of it.



MY STORY ISN'T DUMB

A NINJA MASTER TAKES ON HIS CRITICS

HELLO, I'M KEN OGAWA. YOU MAY REMEMBER ME FROM THE XBOX 360 smash hit, NINJA BLADE, in which I single-handedly saved Tokyo from an infestation of the virulent and seemingly unstoppable Alpha-worms, using only my bare hands and a little bit of ninja magic. (Please do not compare me to NINJA GAIDEN's Ryu Hayabusa. We are very different.)

I have decided to write this editorial because lately it seems like a lot of people have been talking about story in games. In particular, they are complaining about how stories in many games are, in their words, "stupid," or "dumb." To be honest, sometimes I have to struggle to keep myself from using my fully-upgraded Stonerender Sword's devastating Heavenly Slam attack against such people. Who are they to call these stories "stupid?" It makes my alien blood boil just thinking about it. Ops—spoiler alert.

The tale of NINJA BLADE, just to pick a random example—which, incidentally, is available in stores everywhere, like right now—is one of a life-or-death battle against dangerous foes, such as the colossal Plague Snail. I can say with confidence that for me, this was no laughing matter. When this fiendish monster was not belching flaming cars at me, his tongue, which had a ball of spikes on the end of it, had to be carefully dodged. The only thing that saved me was that he fortuitously decided to regurgitate an undamaged and perfectly functional motorcycle with a key already in the ignition, so that I could start it up, drive it down several buses that were floating in the air due to his continuous eructations, and detonate its gas tank inside his mouth, causing a large explosion.

Critics claim that the performance of such feats comes off as silly or absurd, and prevents the story from being "serious." But I ask you, what is really the absurd thing here? The idea that the Prime Minister of Japan would call in a squad of elite ninjas to fight an outbreak of parasitic worms? Or the critics' jealous reactions to the kinds of feats that my many long years of difficult ninja training have enabled me to perform?

That's right, I said it: jealousy. It doesn't take Ninja Vision to see that these critics are motivated by simple envy. And I have to admit, stopping a crashing jetliner by digging my heels into the ground and mashing some face buttons is pretty impressive. (It's a good thing my late, lamented father ordered me to constantly practice the secret airplane-stopping techniques that have been handed down in our ninja family for many generations. At the time, I wondered what the point was, but his wisdom turned out to be unquestionable. It really is a testament to my father's incredible strength of character that he allowed himself to be infected by Alpha-worms in order to become one of them, and therefore discern their headquarters. It turned out to be the gigantic meat-tower on the horizon. Who would have guessed?) Why else would they find fault with what I do? Especially given that I saved... well, I don't know what was on that plane, exactly. I threw all the cargo off. Maybe it was passengers? The plane was really beat up by the time it crash-landed. I don't think anyone could have survived. Anyway, the point is, I fought off the flying hydra monsters and got the plane out of the sky, just like they asked me to do. Or did they ask that? Whatever.



ILLUSTRATION BY JONATHAN KIM

And for the people who say they want more complexity and depth in their narratives, NINJA BLADE also features unprecedented subtle shades of meaning—with just as much of a hidden message as, say, BRAID has. You don't believe me? I'll give you some hints. Why did the guy say "we evacuated this area days ago," just before the camera panned down to show traffic-clogged city streets? Why did the driver of a tank I was riding persist in driving down the train tracks even though an infected train (called the Parasite Train) was pursuing us, instead of just driving off to the side? Think about it. See? Jonathan Blow's got nothing on this kind of stuff.

Of course, only time will tell which stories stand the test of the ages and which fade away into obscurity forever. Will one of the images that persist in the minds of generations to come be the way I, beautifully backlit by the full moon, cleverly utilized a giant wrecking ball, which was serendipitously perched for some reason at the very top of one of Tokyo City Hall's towers, to defeat a menacing arachnid beast by knocking the ball into the air with my sword and riding it down on top of the foul creature's head? Or that stupid part in NINJA GAIDEN II in the colosseum where Sonia shows up in a helicopter and shoots missiles at all the werewolves populating the stands? Because that was ridiculous. ☺

MATTHEW WASTELAND is a pseudonymous game developer who has a fairly common first name. Email him at mwasteland@gdmag.com.

Autodesk Games Insight

The latest scoop from Autodesk Media & Entertainment

Welcome to this issue of Autodesk® Games Insight, our monthly column on what's new with Autodesk in the games industry. In this issue, I'll cover Digital Entertainment Creation, the launch of our suites, and the new 2010 versions of our software.

Digital Entertainment Creation

This recession is putting the spotlight on the rising costs of entertainment production. At Autodesk, we believe that the key to improved efficiency and effectiveness in production is to empower artists with better and faster tools that communicate well together. That's our vision for Digital Entertainment Creation.

The way games are created – from complex environments to believable characters – is improved by using new real-time, immersive, interactive technologies. We are focusing our tools on creative exploration and collaboration. Our products now feature higher-quality interactive rendering in viewports to make better creative judgments on assets. Our real-time animation tools can be integrated more tightly with games engines, for a better understanding of how animation impacts game play.

Our solutions bring together game design, art creation and game programming. With faster turnarounds and more iterations, Digital Entertainment Creation helps you improve both quality and efficiency.



Gears of War 2, image courtesy of Epic Games, Inc.

New Autodesk Entertainment Creation and Real-Time Animation Suites with 2010 software

Implementing this vision involves multiple Autodesk products, and we want to offer you more value. We're introducing two new Suites – offering more than 35%* savings. The Autodesk® Entertainment Creation Suite combines the choice of Autodesk® Maya® or Autodesk® 3ds Max® software, plus Autodesk® Mudbox™ and Autodesk® MotionBuilder® software, in one package. The Autodesk® Real-Time Animation Suite includes Maya or 3ds Max, plus MotionBuilder. These suites include the new 2010 software versions, and are available for commercial and educational facilities.

New 2010 Versions and One Maya

We've just announced new versions of our software, including Maya 2010, Autodesk® Softimage® 2010 software, Mudbox 2010 and MotionBuilder 2010. I invite you to check out all the exciting new features on our website.

I'd like to highlight what we've done with Maya. In Maya 2010, we have combined Maya Unlimited and Maya Complete into one single product. We have also integrated compositing and match moving functionality, making Maya 2010 an ideal solution for the creation of games assets, and also for end-to-end movie-quality cinematic production.

*International savings may vary.

New Releases:

- Autodesk Entertainment Creation Suite
- Autodesk Real-Time Animation Suite
- Autodesk Maya 2010
- Autodesk Softimage 2010
- Autodesk Mudbox 2010
- Autodesk MotionBuilder 2010
- Autodesk 3ds Max 2010 Connection Extension

To learn more, visit:
www.autodesk.com/me2010 or
area.autodesk.com.

Area v3

We love to hear from you and see your work. Over a quarter million artists have joined the AREA, our Digital Entertainment and Visualization Community, and it keeps on growing. We've just revamped the site and new content is posted regularly. Join us online for dialogue, downloads, learning and fun.

Enjoy the ride,



Marc Petit
Senior Vice President
Autodesk
Media & Entertainment
marc.petit@autodesk.com

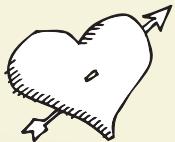
"Time was our biggest challenge with Gears of War 2. 3ds Max, Maya, and MotionBuilder allowed us to iterate quickly and create content in record time. It also enabled us to go beyond anything we've done before. Without Autodesk's software, combined with our own proprietary Unreal® Engine 3 toolset, we would have been hard-pressed to finish the game in such a short development cycle."

— Chris Perna, Art Director
Epic Games

IF YOU WANT TO BE AN EVEN MORE

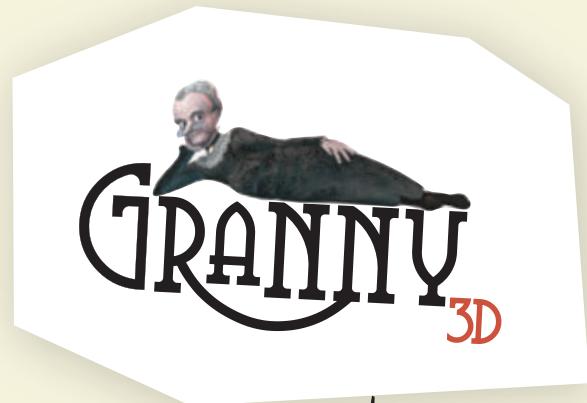
Awesome

GAME DEVELOPER



THEN YOU NEED TO BE USING

Granny 3D

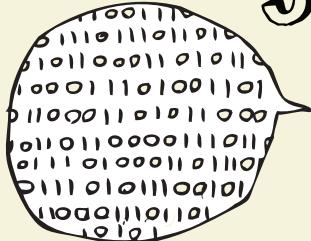


You get a run-time dynamic

3D ANIMATION SYSTEM with

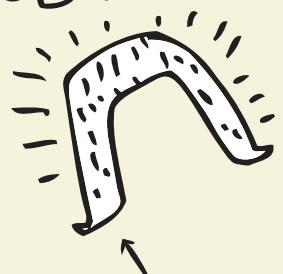
AMAZING content exporters

EASY data manipulation



and a brand NEW blend graph editor.

MUSTACHES



USING OUR TOOLS NOT ONLY MAKES YOU

MORE AWESOME, THEY MAKE YOU

rad!



www.radgametools.com

(425) 893-4300