Postmortem: Treyarch's *Tony Hawk's Pro Skater* (Dreamcast Version)

By Jamie Fristrom

When Activision was looking to sell off the publishing rights for a Dreamcast version of Neversoft's *Tony Hawk's Pro Skater*, Crave Entertainment snapped them up and offered the development contract to Treyarch, who was developing another Dreamcast title for them, *Draconus: Cult of the Wyrm*. Naturally, we were overjoyed to be doing the port of one of the most successful Playstation titles of 1999. The project had all the ingredients for a sure-fire winner: we were moving a title from a low-memory, low-power machine to a high-memory, high-power machine, we had enough time and money, and only a small amount of their code base would need to be rewritten.

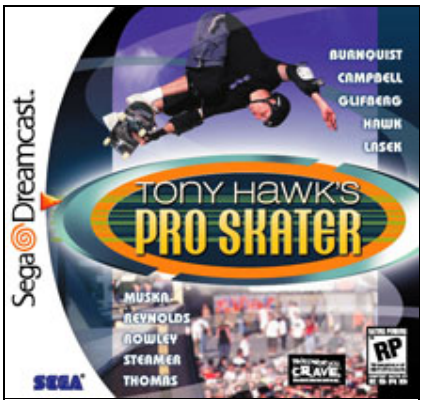
Staffing Up

As cutting-edge as *Draconus* is, we knew it wouldn't sell as well as a blockbuster like *Tony Hawk*. Since *Draconus* was supposedly nearing the end of its development cycle, we decided to move two of the *Draconus* coders (Wade Brainerd and myself) over to the new project. We also selected *Draconus* artist Christian Busic to lead the project and enlisted producer Gregory John, who was busy finishing up *Triple Play 2001*, to manage us part-time. (He spent a lot of time in the office.)

In addition to the two artists we added to the project, we also hired two programmers: Srini Lakshmanan, who had worked on the Playstation titles *Tommorow Never Dies* and *March Madness*, and Sean Palmer, who had worked on *Demise* for the PC and the *Oddworld* port from the Playstation to Windows. My criteria for new programming hires were that they must have finished at least one title, write correct and readable code, and have some Playstation experience. Although Treyarch had plenty of Playstation programmers, Wade and I had no Playstation experience, so we needed to address that gap.

Unfortunately, it takes time to find new staff. It looks good on paper to say, "We can do all this with four programmers, no problem." But you have to remember that those programmers aren't going to join you right away, and even when they do finally come aboard, some ramp-up time is always necessary. Our schedules took that into account to some extent, but not enough. The new programmers and artists weren't there when we scheduled them -- yet more padding lost.

The Issue of Internet Play



Originally, *Tony Hawk* for Dreamcast was envisioned as an Internet game. We planned on a staff of five or six programmers, two for the Internet end and the rest for the port itself. To get an Internet game completed in five months, we wanted to begin the networking support right away on the Playstation before the game was even up and running on the Dreamcast. The Playstation development systems had a serial port that we could send messages through, and that was all we thought we would need to take our first steps toward an Internet game.

Coincidentally, Sega held its first network game conference during the first couple weeks of our project. Wade went and discovered that their tools and lobbying network were not going to be up and running until after our ship date. Needless to say, we scratched the plan for network play.

Even without the network play it looked like *Tony Hawk's Pro Skater* for Dreamcast could still be relatively painless: apart from the renderer there wasn't a lot of Assembly in the existing code base, the Dreamcast is a much more powerful machine than the Playstation so there were no worries about space or speed, and we had a reasonable amount of staff and a comfortable five months to finish the project.

First Steps

The first step in porting Neversoft's code base was to get it building on a Playstation development system. (Michael Montague, who was on the *Triple Play* team, did this for us because he couldn't wait to see what their code looked like.) Their code base was a hybrid of C and C++ code, originally written for *Apocalypse*, with a lot of Assembly for the renderer and some Assembly for the physics. *Apocalypse* was a Playstation game featuring Bruce Willis, which, we learned, is why in *Tony Hawk* the code for the classes of skaters is called "CBruce."

After that, Wade started building it with our Dreamcast tools. For *Draconus* we used the Metrowerks compiler so he tried that one first but then immediately abandoned it. Because Neversoft used GNU for the Playstation version, it made sense for us to use GNU for the Dreamcast as well. Still, he had to go through the tedious process of finding lines that didn't compile because they referred to Playstation library calls; for every one of these he created a dummy function that did nothing but print out a debug message saying that it had been called. After a couple of days he had a program that compiled, linked, and ran -- and did nothing but print out messages.

This is the scary part of the project because there's no real way to measure how far out you are from having a running game with a character actually animating in a level. During *Treyarch's Triple Play Baseball* port from the Playstation to the N64 it took two months to get the game up and running, but that time we didn't have a Playstation system on hand and we weren't that familiar with the N64. We figured that since for this project we had three programmers (Sean hadn't come on board yet), a Playstation dev system, and a lot of DC familiarity, we have it running in just one month.

While Wade got the file system working, Srini and I started filling in the empty Playstation calls, trying to write only the ones used by the renderer and the physics first. Early on we decided we weren't going to use the Dreamcast's wonderful floating-point math because we didn't want to introduce errors into the code before we even had it up and running. (And besides, as slow as the Dreamcast's integer math is, it's still faster than the Playstation's.) We actually had two options when it came to the math; the Edge of Reality guys had already done floating-point math for the N64 port of the game and we were given their code. By this time (see below on our problems communicating with Neversoft), we had already implemented a lot of the functionality in integer, but we had come up against a bug and were thinking of switching to Edge of Reality's floating-point system.

Ultimately we didn't switch, and the decision seemed to pay off: our game behaved almost exactly like the Playstation version, right down to the same quirks and weird little bugs: players can ollie through a chain link fence in a couple of places, occasionally bounce off half-pipes in interesting ways, and even get stuck on the lip of a half-pipe, able to do tricks but unable to move, trapped there until the end of the game. All of these bugs shipped on the Playstation side and we only fixed the ones were players got stuck.

There was one place where the integer math's lack of precision hurt: the animations looked a little jerky. (The worst was when you were grinding a rail you could actually see your trucks shift around on your board.) Sean rewrote that in floating-point and we were fine. We would make little test beds in the first few lines of main() that would evaluate different values of these different math functions in order to test that our version of the Playstation function behaved the same way on the Dreamcast. While we got all the needed functions done, Wade got the file leader going and was displaying the opening title screen.

We then needed to write a renderer so Wade and I pair-programmed on the first draft (more on pair programming is under "What Went Right"). Writing a renderer for the Dreamcast using their libraries is trivial, it's as easy as programming the Voodoo API and even easier than programming DirectX. Soon we had the level drawing flat-shaded, followed by a box for the skater and textures. Finally, we had animations and the game was up and running. All told, this took a total of three weeks. We managed to get our first milestone burn a few days early, and there was much rejoicing.

Tools

We had five Sega "Set 5" development systems. We bought them ourselves rather than borrowing them from Crave, though they did lend us the Playstation dev board. We got one for each programmer and one for the lead artist so he could see the art in the game without interrupting any coders. It would have been nice to have one more for our producer so he could do the build and smoke tests and burn the deliverables for Crave, but since Greg was busy with *Triple Play* it was better for me to do it.

Things got scary because *Tony Hawk* had movies streaming into textures that played while you skated and the current version of the Sega libraries (we were using Release 9) did not support that functionality. Release 10 promised to include it, but didn't beta until we were supposed to be feature-complete, and the final release didn't arrive until after our beta. When asked, Sega said it was confident in its ship dates so we waited for the new libraries and worried. We managed to get an alpha of the movie-playing stuff early, and worked with a hybrid library that was actually quite solid. Unfortunately, the movie library was barely documented, so it us took more than a week to get the thing actually working. We ended up missing our feature-complete milestone because of this one lacking feature. Still, because it was the only missing feature, the game could still be tested without it and we still shipped on time. We shipped using a hybrid version of Releases 9 and 10 which Sega didn't really approve of, but they tested the game thoroughly and passed it anyway.

The artists used Lightwave 5.6 for modeling both characters and exported them with Neversoft's El Pluggo Max plug-in, modified slightly to ease the production path and incorporate high-resolution Dreamcast textures.

Instead of buying top-of-the-line single-processor machines for the coders, we spent the same amount of money getting dual-processor machines with weaker processors. It took some doing to get the makefile that Sega provided for doing builds working with the dual-processor machines, but once we had that going it was fantastic -- a complete build took three minutes. Unfortunately, we had to upgrade the library to get movies working, and for some reason we couldn't get the new makefile working with the dual-processor support without introducing strange bugs into the code. We never understood why, but for the rest of the project we were back to six-minute builds and wishing we had the fast single-processor machines instead. Three minutes doesn't sound like much but for some coders it meant the difference between doing the right change in your code -- even if it means touching an .H file -- versus implementing a cheesy workaround.

Shadows

The code for *Tony Hawk* for Playstation rendered shadows by rendering black ellipses, one for each segment of the character model, onto the ground. We appropriated that code, made it work on the Dreamcast, but discovered that it looked terrible due to Z-buffer poke-through. We then switched to rendering to a texture and then projecting that texture into the world. We tried to find a good way to render that texture dynamically using light sources and projecting it on the nearest surfaces. And after a lot of passable results Wade and Sean came up with the idea of rendering everything near the skater in two passes: one for the geometry, and another to project the shadow texture on the geometry.

This resulted in near-perfect shadows, as they would conform to steps, slopes, and walls. It took Sean some elbow grease to get them clipping correctly so they wouldn't show up on the backside of walls or on both sides of the skater, but the end result was well worth the extra time spent on it.

What Went Right

1. We maintained a conservative game plan.
Most projects are too ambitious in this area, leading inevitably to painful feature cutting as ship dates loom. However, because we scheduled so pessimistically, we were able to survive even with fewer programmers than planned and actually get "wish list" items into the game. Our original plans were for a single directional light on the skater and keeping the straight-down opaque black shadow from the Playstation version. However, our conservative scheduling provided us with some free time to add features such as dynamic lighting and realistic shadows. I had something to prove with this project, which I learned from reading Steve McConnell's book *Rapid Development* (Microsoft, 1996): it's better to schedule pessimistically than optimistically. As our producer Greg John puts it, "Underpromise, overproduce."

We scheduled the project out in weeklong chunks and even though some programming tasks seemed like they should only take one or two days, we nevertheless gave them a whole week. When Don Likeness, Treyarch's founder and no slouch of a programmer himself, saw the schedule he said, "Why is this going to take a whole week? I could code this up myself in a day!" But we stuck to our guns.

This system turned out to work great, because even though it would often take a programmer only one day to implement a feature we had assigned him a week to do, it then usually took a couple more days to get each feature to the state where we could put it in a box and ship it -- we didn't settle for features that were 99 percent implemented. And then it took a couple more days to fix any new bugs that had cropped up.

For scheduling my own programming tasks I was even more pessimistic. This is the first project I've been lead programmer on where I wasn't also the only programmer. After seeing how this company tends to give its lead programmers ulcers, I decided to do a couple of things: For any task assigned to me we doubled how long we thought it would take, running under the assumption that as lead programmer I would be so distracted I'd have a hard time getting anything done. We also didn't assign me anything tricky: just because I was the most experienced game programmer on the team didn't make me the best. And even though I had written a Dreamcast renderer before I didn't take on any mission-critical rendering tasks. I basically did crap work, the front-end stuff and memory card stuff, and left all the graphics and special effects and sounds and optimizations to the other guys, which not only made my life easier but also helped to keep them challenged and motivated. Still, even with my extra-pessimistic schedule, I barely managed to stay on top of my programming tasks.

How did we schedule optimization? The goal was to be at 30 frames per second, always, even in two-player split-screen mode when both skaters are being rendered in both windows. To track the scheduling progress we took one of the worst case scenarios, found the frame duration (it was around 60 milliseconds and we had to get down to 33) and made a rough estimate of how much progress Srini and Wade would need to make each week to hit 30FPS by alpha. Knowing that optimization gets tougher as you go along, we assumed that each week we'd make about half the progress of the previous week: trim off 16ms the first week, then eight, then four, and so on. Srini and Wade managed to stay well ahead of this wild guess, so we didn't have to worry. Hitting 30FPS was easy but hitting 60 would have been very difficult. (That would have meant cutting the worst-case frame time from 33ms to 16.) We estimated how much we'd have to do to get past limitations in the code that prevented the Dreamcast from hitting its full stride and it ran out into several weeks. Since we weren't doing a fighting game, all that 60FPS would have gotten us would have been a nice marketing bullet point.

We were often wrong in our scheduling estimates, but the mistakes balanced out. Sean would see things on his schedule that he was pretty sure he could knock out in a day and to prove it he would just go ahead and do it. So we found ourselves readjusting the coding schedule every couple of weeks as we learned that something we had scheduled a week for could be banged out in an afternoon (getting the sky rendering correctly, for example). But we also learned that things that we thought would take two weeks actually took four, such as the game's new shadows. We ended up with fewer programmers than we had originally planned for, but our pessimism weathered that storm as well.

Greg didn't use any funky "Project"-oriented software to do our scheduling, just Excel: programmers across the top, weeks across the side. One box represented one week and if he needed to reschedule, he would just cut and paste boxes -- we could have just as easily used three-by-five cards and corkboard.

In the end, we finished one week early and we could have finished two weeks early if we had better QA. We could have saved two more weeks if we didn't have to wait for Sega's R10 library, skipped implementing dynamic lighting, and left the shadows the way they were in the demo. (The demo version of Tony Hawk for Dreamcast, which ships with new Dreamcasts now, has shadows that have some artifacts). I ask myself; if we hadn't had to wait for R10, would it have been better to schedule a little more optimistically, lose those features, and shave those last two weeks off the project? I don't know. So it's possible we were too pessimistic. Still, having now been on projects that were both underscheduled and overscheduled, I'll take an overscheduled one, hands down. We had to put in some overtime, work a weekend here and there, but compared with other projects I've been on, it was a vacation.

2. We had a Playstation dev kit on hand and kept the Playstation build going and stable well into the middle of the project.
This one seems like common sense. Start with a project that builds, runs, and work from there. It's tempting to skip this step since you don't technically need a Playstation development board to do a port to a different platform and because they're so pricey you might rather go

without. There are obvious ways in which having the Playstation build on hand helps: First, you can step through the two versions and see why the behavior on the Dreamcast doesn't work when the behavior on the Playstation does. Also, if someone discovers a bug on the Dreamcast version, you can check to see if it's a Dreamcast-specific bug or if it's something that you also introduced on the Playstation as well. For example, we'd implement a new feature on the Dreamcast side and discover something wrong with it, and then we'd recompile on the Playstation and see if it was a bug we introduced or a bug due to just not completely implementing or understanding the feature. It's very unpleasant having a code base that just doesn't work and you don't know why and you don't know how much work you have to do to get something that does work. Having a working build on hand gave me the confidence to know that if something did go wrong, we could figure out why.

Eventually we got to the point where the time taken to make sure it would build and run on the Playstation wasn't worth the effort. This was about halfway into the project. After we let it go, there were a couple of times that I regretted not having the Playstation build anymore to check things with, but we survived.

3. Daily builds and smoke tests, and an internal bug list.
This is discussed in Frederick Brooks's, *The Mythical Man-Month* as well as *Rapid Development*. Every morning (well, almost every morning), I would do a get of our programmers' latest changes and artists' latest data, make sure it built, and make sure all the levels loaded and ran. This process would take an hour or two out of my day, but we caught severe bugs almost immediately and dealt with them. So we almost always had a solid code base, and were never hit by the problem (all too common on other projects I've worked on) where a milestone was due and we would get ready to submit it and discover that half of the levels didn't even load because they hadn't been tested in a couple of weeks. Things broke in our code base quite often, more than once a week I would say, but any bug that was accidentally introduced would get caught by this process. I'd also do soak testing overnight as well, to make sure memory leaks weren't a problem.

Everything near the skater is rendered in two passes: one for the geometry, and another to project the shadow texture on to the geometry.

We had an internal bug list from the start. Those of us with development systems were encouraged to add bugs to the list by submitting them to Greg when we discovered them. We also would occasionally have people come in and do a little in-house QA, about one tester-day per week. A lot of the bugs on the list were nonfatal bugs that were due to us not having implemented a certain feature yet (or ones that we weren't planning on fixing at all). These were the bugs that we left until later to fix but everything else we got on as soon as possible. The "A" bugs were the kinds of bug that we'd stop what we were doing to fix, but as for lower priority bugs, we'd got to them as soon as we were at a good stopping point.

When we got to alpha, (feature-complete except for the movies streaming into textures), we shared our internal bug list with Crave, creating a master list which their QA and our QA could both add bugs to. This may not have been such a good idea because it meant they had to regress our bugs as well as theirs, and they could have spent more time looking for new bugs if they weren't so busy regressing bugs that we had already regressed internally.

It took six weeks from when we were feature-complete to when we made our final submission to Sega, and the bug count reached nearly 800, which surprised Crave since most of the ports they had seen had far fewer bugs. Part of the reason for this was because we were reporting bugs to the master list rather than fixing them internally and more than 200 of the reported bugs were discovered internally.

4. Continual communication between all team members.
We kept the *Tony Hawk* Dreamcast team fairly localized in the building. Sean and Wade shared an office and the artists were in adjacent cubicles. I spent the first few weeks in Srini's office while we were getting the game up and running for the first time and I think having Sean and Wade in the same office helped make shadows as good as they are; they got to throw ideas back and forth about how to make them look the best. We all had our own phones. I was also reachable by cell phone and Greg had a pager.

On top of that, there was "pair programming." Pair programming is discussed in Larry Constantine's *On Peopleware* (Prentice Hall, 1995) and also on the *Extreme Programming* site (http://c2.com/cgi/wiki?ExtremeProgramming). I'm a big fan of pair programming because I think it's almost as beneficial as doing code reviews. The advantages of pair programming are that it keeps you focused on the current problem because you have somebody looking over your shoulder, the programmer looking over your shoulder can spot bugs and bad practices in your code, and, since most programming is actually problem solving, having two sets of eyes working on a problem means the problem might get solved that much sooner. The disadvantage, of course, is that those two programmers could be working on two separate features in parallel instead of being tied up on one feature together.

We did a lot of pair programming in the beginning of the project. We pair-programmed the first draft of the renderer to get flat-shaded levels up and running, and the construction of the Playstation emulation math functions. I would team up with someone when the feature they were working on started to slip on the schedule and we would also pair-program some of the more involved bug fixes at the end of the project to make sure we didn't screw up.

5. Marketing requirements were usually requested with sufficient lead time.
This is something that's so easy for developers and publishers to forget. Suddenly you're two weeks out from having to get a demo going for a magazine but you have no time scheduled for it and the game isn't ready for prime time. For us, it was understood from the beginning of the project that we'd need to have a demo running halfway through, and thus we scheduled ample time for it. Because we had the demo running so early, we were able to get it on the disk that shipped with Dreamcasts, without having the demo impact our schedule.

What Went Wrong

1. Not enough QA on the project after we were feature-complete.
When QA begins you can measure how quickly bugs are being reported and howquickly they are being fixed. This data can be used to give you a rough estimate of how long it will take to finish the product. For example, you might find that QA is averaging 20 bug reports a day and your programmers are fixing 10. If there are 400 bugs, it will take roughly 20 days to find them all and 40 days to fix them. In our case, we discovered that we were fixing bugs faster than QA was reporting them; we had a backlog of internally reported bugs to keep us busy, but eventually we were going to catch up to QA. Greg screamed at Crave for more QA, but there were plenty of days when the programming staff was idle while we waited for bugs to trickle in. QA became the bottleneck on our schedule; if they had found the bugs sooner we could have shipped sooner. QA was lacking in both quantity and quality because there were not enough guys on the project and they were not covering the game adequately.

2. Increasing the number of polygons doesn't automatically give you great results.
We tripled the number of polygons in the skater models but this ended up making the skinning at the joints look more noticeably wrong, and didn't improve the look of the characters much. Sometimes we couldn't even tell whether we were playing with Neversoft's original low-definition skaters or our new high-definition ones. (Although looking closely at the skateboard made it a dead giveaway; ours had nicely modeled trucks and theirs used flatcards.) With new power comes new responsibilities, such as fully taking advantage of all those polygons by implementing extra joints in the characters (at the shoulders and necks for example), weighted vertices, and cloth and hair systems. Our other Dreamcast games have these features but we were locked into the animations from the Playstation version of *Tony Hawk* so we couldn't add shoulders nor could we move pivots for more natural bends. And the skaters in *Tony Hawk* go into extreme poses while performing stunts. For instance, when they crouch really low on their skateboards their knees get pointy, and when they put their arms over their heads their shoulders dip and they look like balloon animals. We did the best we could by hacking in weighted vertices on the knees but the hack made the shoulders look even worse so we left them the way they were.

3. Communication difficulties with two publishers and the original development team.
Because Activision sold the rights to Crave, to obtain assets or anything else we had to ask Crave, who in turn would ask Activision, who in turn would ask Neversoft. When we would suddenly discover we were missing something, it took forever to do anything about it. For example, when we needed the raw sound files it was days before we actually got a CD with the files on it, and the CD wasn't exhaustive. We had similar problems with tools -- we never did fully understand Neversoft's Max plug-in and were limited in what we could do to improve the levels because of it -- as well as with some of the source code.

4. We didn't try to export everything right away.
Neversoft's El Pluggo plug-in exported a Playstation file and this was the model format that the game read in, Neversoft's proprietary binary mesh format. We were given source to El Pluggo so we could modify it as we saw fit but our mistake was that we only tried exporting levels on an as-needed basis. (We already had all of Neversoft's Playstation files and we were originally we were running the game with these pre-exported levels.)

We discovered this problem the hard way when we got to one of the last levels and it didn't export correctly. It turned out we somehow had ended up with an obsolete version of the El Pluggo source. It took a couple of days to rectify this problem, because the turnaround for getting deliverables from Neversoft was so slow, and because we had to integrate our changes into their new plug-in. If we had tried exporting every single level at the start of the project we would have gotten the correct source before we started modifying it.

5. "What do you mean the game hasn't been localized?"
Localizations turned out to be a slap in the face. Looking through the code at the beginning of the project we saw that they had localizations for various European languages, so we thought we were covered. Later on in the project we realized that for some reason when we activated

the localizations a lot of the English text wasn't being translated. It turned out that localization code we were looking at was for *Apocalypse* and that the Playstation version of *Tony Hawk* had never been localized. It could have taken weeks to get all their text translated. The only thing we could do was to skip the localizations.

A Slam Dunk

This morning, as I put the finishing touches on this article, *Tony Hawk's Pro Skater* for Dreamcast is hitting the shelves, and due to preorders it's already the number-one Dreamcast title at EB World. We all expect it to do very well. I never thought I would enjoy doing a port because I didn't think it would allow for creativity or give me the same sense of accomplishment that working on an original title does. In actuality, the ratio of feeling accomplishment to the time spent on the project is actually higher than for original titles; you only spend a few weeks on it and suddenly you have a functional game. And a few months later, you get to ship! It feels great and I highly recommend it (unless you're going from a Playstation to a Nintendo 64).

Jamie has been programming games in Los Angeles and San Diego for nine years. He still makes a lot of mistakes. Check out his website at http://jfristrom.home.mindspring.com/ or e-mail him at jfristrom@iname.com.

Return to the full version of this article