

Taking a Whirl Tour on the Nintendo GameCube

by Philip Chu

Table of contents

1 Publication Information.....	2
2 Game Data.....	2
3 Introduction.....	2
4 What Went Right.....	3
5 What Went Wrong.....	7
6 Lessons Learned.....	12

1. Publication Information

Copyright ©2003 by Philip Chu All rights reserved.

This postmortem article was published by the Nintendo Developer Support Group. Details covered by the developer non-disclosure agreement are omitted here.

2. Game Data

- Publisher: Crave Entertainment and Vivendi Universal
- Release Date: November 2002 (US) and March 2003 (Europe)
- Number of Developers: 15
- Number of Contractors: 2
- Length of Project: 15 months (plus 4 months for European SKU)
- Development Hardware: 1.5GHz PC's with GeForce 3 video cards running Windows 2000. GameCube devkits, test consoles and disc burners.
- Development Software: Microsoft Visual C++, SN Systems ProDG, Havok, 3D Studio Max, Adobe Photoshop, Adobe Premiere, CoolEdit, cvs.
- Project Size: 360,000 lines of source code. 900MB of compiled data.

3. Introduction

Papaya Studio's Whirl Tour was released on the Nintendo GameCube and Sony Playstation 2 platforms in November 2002 and March 2003, for North America and Europe, respectively. The game, published by Crave Entertainment and Vivendi Universal, combines a story-based adventure with Tony Hawk style extreme sports play on scooters. Although Whirl Tour is a multi-platform title and the merging of genres poses intriguing game design questions, this article focuses solely on the GameCube development.

We started work on the GameCube in July 2001, a few months after the Whirl Tour prototype was completed and running on a PC. Crave received milestone deliveries every two months - the first one involving a GameCube build took place in January 2002. We delivered a build to Nintendo of America (NOA) the following April for concept review, and Crave demoed the game at E3 in May. That summer, Crave forwarded our beta release as a presubmission build to NOA lot check, the group responsible for verifying compliance with

Nintendo guidelines. The final submission was approved in September 2002 for the North American release. Over the next few months, under the aegis of Vivendi Universal, we undertook localization for the European release of Whirl Tour, which was submitted to Nintendo of Europe (NOE) and approved in January 2003.

4. What Went Right

4.1. 1. Development Tools

When we started GameCube development, five months before the console showed up on retail shelves, development hardware was scarce and software tools were still under development. But we minimized the impact of this problem by getting started with Nintendo's software based emulator and obtaining the tools as early as possible and in sufficient quantities.

The GameCube emulator provides API-level emulation on a PC. Our team was already working with a PC reference implementation of the game using Visual C++, so while we waited for devkits (programmer version of GameCube hardware that contains twice the usual memory and loads data from the developer's PC instead of disc), we incorporated code that called the emulated video, graphics, disc and controller API's. In six weeks, we had a pseudo-GameCube version of our title running, albeit at 5fps.

We applied some optimizations to get the game running up to a slightly less painful 10fps, but there was always a nagging suspicion that time spent optimizing with emulator wouldn't result in anything useful on the real hardware. It was a relief when the first devkit arrived, two months after we started work with the emulator, but it took three months to really get the game built with the compiler (SN Systems ProDG) and running on the hardware. We had to wait for a GameCube port of the physics middleware, and we needed to update our offline data preprocessor to convert the little-endian data produced on the PC into big-endian data readable by the GameCube CPU (a custom PowerPC called Gekko).

Although we had to wait for a port of the physics middleware, it was still important that we received the devkits as soon as we did. Aside from the endian issue, there were a number of subtle differences between the emulator and actual hardware that had to be resolved, including differences between the Visual C++ and ProDG compilers. Likewise, we reaped benefits from obtaining the test consoles and disc-writer as soon as they became available. They arrived shortly before our first schedule GameCube milestone, so we were able to make that delivery to Crave on a test disc instead of a bunch of development files for a devkit.

Besides getting all the tools as early as possible, we obtained redundant quantities of each. The extra units saved time and provided peace of mind. For example, we ran 24-hour-plus

smoke tests concurrently on multiple test consoles - when one console occasionally reported a fatal error and the others didn't, we concluded it was a hardware problem and didn't lose sleep over it (while we were losing sleep over other things). And when we had to return devkits for repairs (video card damage) and upgrades (PAL support), we could send back one unit at a time without stalling development. We also ordered hundreds of blank test discs in anticipation of daily burns and crunch time. The last thing we wanted was to miss a deadline for lack of discs.

4.2. 2. Developer Support

We were fortunate in receiving thoroughly good support from all our licensors and vendors - Havok for our physics middleware, SN Systems for our compiler and IDE, and Nintendo Software Developer Support Group (SDSG) for just about everything else.

SDSG resources got us started on a lot of code. Our preliminary implementation of dynamic shadows, using shadow volumes, was guided by an SDSG white paper and refined based on ideas exchanged in the developer newsgroups. The final shadow implementation used shadow maps and was initially based on one of the SDK demos. For a screen snapshot function, we reused the source from a utility posted on the SDSG web site.

Access to SDSG was helpful throughout the project but proved especially so during the submission period. Our US submissions were rejected twice by Nintendo lot check, and our publishers wanted fast turnaround for each resubmission. Email and phone queries to SDSG resulted in quick clarification of guideline items and evaluation of proposed solutions. This enabled us to get the first resubmission out in less than a week and the second resubmission out in one day. Combined with lot check turnaround of a few days for each submission, the total time from first submission to final approval was just under a month.

The delay in getting a port of the physics middleware could have been serious, but a few weeks after we got our devkits, Havok sent us a linkable library that allowed us to complete a build. When it became apparent that we needed endian-conversion for our collision surfaces, they quickly sent us a run-time solution, and eventually a preprocessor solution that we integrated into our offline data converter. The GameCube port was basically complete in five months and we received incremental updates throughout the rest of the project.

We opted for SN Systems ProDG over Metrowerks Codewarrior because we were already using ProDG for the PS2. This minimized compiler differences between the two platforms (we already had differences in code acceptance between Visual C++ and ProDG), but meant starting out with a beta compiler. The few bugs we encountered were addressed quickly by SN Systems support, and the official compiler release was ready a few months before the first GameCube milestone. After that, patches and upgrades were readily available on the SN Systems support site, and they had quick responses to questions sent via email or posted to

the gamecube.snsystems newsgroup on the SDSG site.

4.3. 3. GameCube Architecture

Another reason we got the game running quickly on the GameCube is the friendliness of the architecture. The graphics hardware (Flipper) supports multiple dynamic lights, multitexturing, mipmapping, blending, and fog, so most of our basic renderer features were easy to implement (though optimization required some work - see What Went Wrong). The GameCube combines a unified memory architecture (UMA) with fast 1T-SRAM, allowing flexibility in managing textures and the video framebuffer.

The hardware features a generous 16MB of auxiliary RAM (ARAM). We used half of the ARAM for sound effects - given more time, we would have used the remaining ARAM to increase the fidelity of our sound samples. For songtracks, we used the hardware streaming direct from disc. The disc is small and fast, so we never had a problem with load times. The disc capacity is lower than a standard DVD, but at 1.5GB, it was plenty for our 900MB of data.

4.4. 4. Memory Management

Although fast, the GameCube main memory (MRAM) is limited in size (24MB) compared to the PS2 (32MB). We stayed within the memory limitation by aggressively keeping on top of that issue. By running on the test consoles as soon as possible, we forced ourselves to reach the 24MB target early, within the first six months of the schedule. The UMA meant the video framebuffer occupied MRAM. Since PAL resolution is higher than NTSC, we sized the framebuffer for PAL in all builds to ensure we wouldn't have memory problems later. We configured all of the devkits to use only 24MB, so the programmers would know immediately if any level didn't fit. Testers went through all levels in each daily build and in their email report listed any levels that didn't load.

Flipper supports S3 texture compression, so we compressed as many textures as possible. Any such textures that didn't look good after compression were reworked. This considerably reduced memory usage (and as a side effect improved performance and load time) and didn't seem to degrade the visual quality noticeably. After E3, new assets pushed us over the memory limit, so we turned mipmap generation off until we had more space and a more selective procedure for mipmapping.

4.5. 5. Testing

We made a point of testing early and often. Starting about a month before the first GameCube milestone, we released daily builds on test discs for our QA group. The builds

were created from the latest code and data checked into cvs. Our testers issued a daily test report noting new bugs, verifying fixes, and listing minimum single-player and multi-player frame rates for each level (the build included an on-screen fps display).

Crave provided us with a tester from their staff who worked on-site with us and tested our daily build. At first, we sent our Friday build every week to the rest of their QA group, but in the last two months before submission we Fedex'ed them our builds, daily. We also set up remote access to their bug database and used that as our sole bug reporting and resolution mechanism. Since changes to the database would show up on the client GUI instantly, we would be alerted to the new bugs as they arrived and could investigate and update the database entry immediately with comments or requests for closure.

The Crave QA group provided valuable console testing expertise, including experience with a recent GameCube title, and were familiar with the Nintendo developer guidelines. Soon after they started testing Whirl Tour, their GameCube bug database contained about 200 entries, many of them memory card and disc error handling issues that would have been flagged by lot check at submission. By the end of the project, we had around 700 bugs logged and resolved for Whirl Tour GameCube.

4.6. 6. Schedule

Originally, we estimated the GameCube development of Whirl Tour would trail the PS2 by one to three months, for reasons including the expected delay for hardware and middleware and less in-house familiarity with the newer GameCube platform versus the PS2. But that meant we would at least partially miss the critical holiday retail season. When it appeared that we could deliver our first GameCube milestone on disc, it made sense to synchronize the GameCube schedule with that of the PS2. This allowed us to target September for our North American submission, which would allow the game to arrive on the retail shelves in November. And it greatly simplified our overall project schedule, since we didn't have to track different milestones on each platform.

The decision to sync the schedules was consistent with our approach of top-down milestone-driven scheduling. Instead of building the schedule bottom-up by summing up low-level tasks, a method which accumulates error over time, we built the schedule top-down, placing the most importance on critical deadlines like E3 and the target submission date, followed by our milestone commitments to the publishers, and then juggling tasks to accommodate those dates. Adhering to the big-picture deadlines proved even more important later in the project, when Papaya was gearing up for the next project - any delay in this title would have just propagated to the next.

The schedule was tracked with Microsoft Project. Milestones were marked as such using the milestone task annotation in Project, and features required for each milestone were listed as

subtasks, down to a granularity of no less than one week per task. We marked only real dependencies between tasks, no artificial links to force a nice-looking sequence, and we let Project calculate the expected dates. When a task was completed, we filled in the date for the finished task and let Project recalculate the dates for the remaining tasks. Thus we used the schedule not so much as a micro-management and task-scheduling tool than as a schedule feasibility tracker and early warning system.

We had weekly team meetings that initially involved only the leads and proceeded in a go-around-the-table-and-report fashion. Eventually, the format changed to include all team members and followed an announced agenda. This resulted in more focused and shorter meetings, sometimes lasting only fifteen or twenty minutes and rarely lasting more than an hour. Minutes were distributed by email afterwards. The meeting time was also minimized by requiring each team member to send a company-wide status report every Friday on his area of the project.

5. What Went Wrong

5.1. 1. Performance

The biggest disappointment was our failure to achieve our original performance target of 60fps for all levels in single-player mode. We did achieve the desired 30fps for multiplayer mode in all levels, but for single-player mode we could only maintain a consistent 60fps in two-thirds of the levels. The remaining levels alternated between 30fps and 60fps, so we had to lock them to the lower frame rate to keep the visuals smooth.

We made a mistake in not achieving and maintaining the target frame rate early. We started with a consistent 30fps in our first GameCube milestone, and the code performance improved all the way through the project. We planned on achieving 30fps for the first milestone and reaching 60fps by E3. The fact that we only had a few levels running at 60fps by E3 should have been a wake-up call, but we succumbed to wishful thinking and hoped that further optimization techniques would get us to our goal. We did improve code performance by about 50 percent between E3 and submission, but with more NPC's, water effects, particle effects, dynamic shadows, and so on, we just kept pace with the additional assets. In the end, we still left a third of the levels below target in single-player mode.

It became apparent, too late, that many of our optimization techniques did not improve the worst-case performance in our problematic levels. If we had recognized this earlier, we may have been able to restructure the levels to exhibit more balanced performance. There were some architecture-specific optimizations that we did not fully explore, but again, we ran out of time.

5.2. 2. Submission (and Resubmission)

The NOA submission was rejected twice before approval. That isn't a particularly bad showing, but there was a lot of pressure to get each resubmission out quickly, and it wasn't fun. We cut it close in terms of getting the release out in time for the holiday retail season, so it would have been more comfortable if we could have knocked it down to one or two submissions.

The most persistent problem areas that kept showing up on our lot check submission reports included memory card error handling and on-screen depictions of controller buttons. We could have gotten a pass on the button graphics, but we updated the button graphics several times anyway in hopes of making them more acceptable to lot check. The memory card issues for the most part were not negotiable - there are comprehensive guidelines involving error-handling flow and messages. These are not very exciting features to implement, but they are required, and we should have tried to get that stuff done earlier.

Our programmers and testers were not familiar with the GameCube submission issues, and as a result, we were overly reliant on Crave's QA team to identify the problems. Once Crave started testing the game, we basically had two months to rewrite our memory card, disc error, and reset-handling code, while simultaneously adding our shadow code, performance optimizations, and trying to fix all other bugs before submission. This was complicated by a couple of attempts in the last few weeks to advance the submission, which, in retrospect, now seems unrealistic.

We should have educated ourselves on the submission issues much earlier by reading up on the guidelines and examining other games, and we should have scheduled completion of all the guideline requirements by our beta release early summer. Crave sent the beta to lot check, but we didn't receive the results until two weeks before the final submission. The presubmission report from lot check was useful, but would have been more so if we had tracked it down earlier, and if didn't have so many violations (too many to fix in one pass).

We complicated the memory card situation by attempting to share much of the framework with the PS2. We delayed much of the GameCube-specific implementation until the PS2 code was nearly done, so most of the work was completed in the last three months before submission. The code sharing seemed convenient initially, but there were significant differences between memory card guidelines for the two platforms. The resulting error-checking and error-handling flow was somewhat atypical for a GameCube title, requiring clarification among the QA groups, lot check, and programmers. After hasty code mangling during the resubmissions to accomodate QA and lot check objections, the shared code was a mess. It seems obvious now that we should have completely separated the memory card code for the two platforms.

5.3. 3. Manual Builds

Although we had a daily build, it was not automated. We performed the build on a developer station using the ProDG/Visual Studio environment. This allowed us to make quick fixes before releasing the build to QA, but it was not the type of clean and reproducible build that good configuration management practices mandate. An overnight batch build would have saved time (each build took anywhere from 30 minutes to three hours, depending on fixit attempts), been reproducible, and, if we set it up properly, automatically logged and emailed build results.

Moreover, an automatic build could have helped to establish a sense of regularity and process. The team members were never sure when the next build would be available, so eventually we had to send email every day announcing the build. Aside from the testers, most of our team members were still overly reliant on the PC version on the game and there was a slightly disruptive tendency to go straight to the programmers instead of checking the latest official build. And the fact that the build took place manually and not at any designated regular time made it too tempting to request new builds at increasing and irregular intervals, particularly during crunch times.

We had our data converter running every night by the end of the project, allowing us to check the latest art and sound effects changes. But the data build did not include processing of songtrack and FVM data, which were provided by external sources as WAV and AVI files requiring lengthy (hours to do a whole set) conversion to GameCube-specific compressed streaming formats. The FMV conversion was particularly involved, as we required versions for NTSC and PAL, and we had to extract the audio, resample and merge it back in to the converted FMV files. Unfortunately, the songtrack and FMV sources arrived fairly late in our schedule, so we were processing iterations of this data well past our attempted one-month data freeze before submission.

5.4. 4. Wanted: Tool Programmer

We could have used a tool programmer. Our data converter was augmented by several different programmers in an ad-hoc manner, so new features were not consistently implemented with all the target platforms in mind. For example, although mipmapping and texture compression were technically multi-platform features, the code ended up in an "ngc-utils" file, and triangle-strip generation and texture palettization ended up in PS2-only code. A dedicated tool programmer could have developed the converter with more of a big-picture approach and taken some of the load off the other programmers, particularly the console programmers and lead programmer. A dedicated tool programmer could also have addressed the artists requests for previewing tools, which we just didn't have time to work on.

5.5. 5. Lengthy Localization

The localization process for the European release took a lot longer than expected. Crave's QA group was regularly testing our PAL builds well before the NOA submission, so we expected to have a submission to NOE ready as soon as we had the text translations. The game engine modifications turned out to involve new texture fonts for the extended character set, additional character selections for text entry screens, resized GUI elements so longer European words would fit, and proper synchronization of the in-game language with the console ROM language setting. The code changes took a while, but a greater delay was incurred by an initial one-month wait for the first set of translations from Vivendi, then several iterations with Vivendi's QA group to fix translations, typos, and text trailing off screen. We made our submission to NOE three months after the NOA submission, and then it took an additional three weeks for NOE to reject the first submission and approve the second.

5.6. 6. A Mild Case of Featuritis?

We didn't completely meet our performance targets, and we had a crunch time involving a flurry of fixes and resubmissions. So any work that didn't result in obviously necessary functionality warrants reevaluation:

5.6.1. Audio Effects

We had audio reverb and delay effects on our requirements checklist, but they were used in only one area in the game. It was gratifying to say we used the audio effects hardware on the GameCube, but it made our audio code layer significantly more complex and took several weeks to get it right. Compounding this, we were worried about the audio DSP dropping voices when oversubscribed, so after E3 we rewrote the system to handle this eventuality. But the rewrite took two to three weeks, the resulting code was messier, and once we had all the sounds in the game, the built-in GameCube metering hardware indicated that we were only using twenty five percent of the DSP cycles in the worst case.

5.6.2. Screen Capture

We had a publisher requested for a screen snapshot function, which we implemented shortly after the first GameCube milestone. It worked only on a devkit and was used by our testers for their bug reports. But it's unlikely that it was used by anyone else, internally or externally. Submission releases were not supposed to include debug code, so we ended up removing the screen snapshot code before the presubmission. All in all, we spent five months maintaining that feature.

5.6.3. Video Modes

Nintendo recommended but did not require support for progressive scan. We implemented it, but neither we nor our publisher's QA group had progressive scan monitors, so we relied on informal testing off-site using personal equipment. Our progressive scan support was noted by some reviewers, but there was a cost in coding, testing, and added risk to the submission, as Nintendo has guidelines on proper implementation of progressive scan.

Similarly, Nintendo recommended but did not require support for 60Hz mode in PAL releases. The number of users who can take advantage of this feature is probably higher than with progressive scan, and this feature was testable in-house and by the publisher's QA group, so it is a more justifiable feature. But again, there was a development and QA cost, code complexity to handle both 50Hz and 60Hz modes, e.g. in selecting the appropriate FMV to play, and additional localization required for text displayed when switching modes.

5.6.4. Shadows

We implemented detailed dynamic shadows for player characters and enemy bosses. If we had limited the detailed shadows to just the player characters and used blob shadows for the bosses, at least two more of the levels would have run at 60fps instead of 30fps. It wouldn't have looked as good, but would have been no worse than many high-profile games that have recently appeared. We used higher-resolution shadow maps for the player characters, while allowing the boss shadows to have a more jagged appearance. The hi-res maps looked good, but the effect was subtle, and there may have been a performance penalty. Late in the project, we combined the shadow map blending with a location-based shadow value so the dynamic shadows would blend with pre-shadowed areas. Yet again, this looked good, but the effect was subtler still, and it was risky to make such a change right around submission.

5.7. 7. IT

Our server crashed right around E3, and although we had an automatic tape backup running every night, it turned out the data was unrecoverable. Fortunately, we had local copies of the working data distributed around the office, and we had performed periodic manual backups on DVD. It took a while to piece together all our code and assets, and some of the work done just before E3 was lost. It could have been worse, but we did pay the price for failing to ensure the integrity of our backup system.

After the server crash, we spent the next several months upgrading our IT infrastructure. This didn't distract too much from the ongoing development work, but we could have done a better job of minimizing disruptions by communicating and coordinating the IT plans. For example, switching from local accounts to domain-based accounts made some development

tools unusable until some Windows registry values were updated. Even then, we were unable to upgrade our ProDG installation, which meant we were unable to upgrade the Nintendo SDK. And we had to use the original local account for our disk burns. We could have attempted to diagnose the problems or at worst perform a complete reinstallation, but with crunch time starting and the submission deadline looming, we really didn't want to take the time. This might have been avoided if we had discussed the IT planning in our weekly meetings and coordinated some upgrade tests and fallback policies.

6. Lessons Learned

This article focused on the GameCube development of Whirl Tour, but most of these issues are applicable to console development in general. As a console developer, many things are outside of your control, so control what you can. Get the development tools as soon as possible, and get more than enough. Make full use of developer support. Achieve memory and performance targets early and maintain them. Establish daily automated builds and testing early. Take care of the requirements before working on bonus features. Schedule around the really important milestones and then stick with the schedule. Do early and multiple presubmissions if possible. Plan for a few resubmissions in the end - allow a month. And localization will take longer than you think.