

Gama Network Presents:

# Gamasutra.com

## Audio Postmortem: *Moderngroove - The Ministry of Sound Edition*

By Leonard Paul  
Gamasutra  
December 7, 2001

URL: [http://www.gamasutra.com/features/20011207/paul\\_01.htm](http://www.gamasutra.com/features/20011207/paul_01.htm)

### The Idea

Moderngroove's simple yet innovative concept was to make a be-your-own-VJ product set to the music of the Ministry of Sound. The idea was to bring the sights and sounds of club life into your own living room.



### Tall Paul appears as one of the five DJs from the Ministry of Sound

It took a long time to convince the Ministry of Sound to use their music since they were used to a much higher profit margin on their CD releases. In the end, it all worked out and we ended up with five hours of exclusive DJ sets from one of the most popular super clubs and dance labels in the world. It also took a few months to secure our publisher, UBI Soft, since we had the resources to hold out for a good contract.

When we started *Moderngroove* in September 2000, we had approximately two months to complete the project. Everyone knew the timeline was silly, but tasks were scheduled to fit the development time anyway. This is definitely one of the most common errors in management and scheduling: making the tasks fits the timeline rather than making the timeline fit the tasks. This had the unfortunate side effect of making us very reluctant to schedule later on in the project. We felt that it only resulted in us feeling bad about missing self-appointed deadlines and wasted time

continually adjusting for arbitrary slippage and delays.

We continually believed that we could heroically manage the project as a team, which further extended the development time and stress on the team, especially the lead coder. I believe that a good manager with experience in similar small projects might have saved us a bit of time and a fair amount of stress, but I also believe that the project's schedule would have still remained largely unchanged and unmanageable. This was due to the sheer number of unknowns: new team, new company, new development environment, new workspace, new tools, new hardware, new product and so on.

Our basic problem came from a continual slippage of our ship date. Each month, from November 2000 onwards, it slipped a month until we finally shipped. To the team, we always pushed for the next final date, seemingly always just out of reach. We amassed many hours of overtime, which took its toll on us physically and mentally in the long run. We were quick out the starting gates with a simple prototype, but we eventually burned ourselves out. In hindsight, we could have likely gotten the project done sooner with less stress by working regular hours towards a more realistic final ship date.

Unfortunately, for small developers, there seems to be a lack of good management theory available. This is partly due to the fact that larger teams require more management, but more likely due to the fact that larger companies, universities and government agencies have the luxury of more time and money to spend on researching and documenting their process. In a small company, it is difficult to get paid to do proper software engineering since one is usually too busy trying to show how much stuff (coding) is getting done rather than concentrating on less quantifiable tasks like design, documentation and process analysis.

## What Went Wrong

**1. Difficult development environment.** I would have to say that PS2 is difficult to program due to the machine's multiple processor architecture. Most of the other programmers on the team were able to concentrate only on the CPU (EE), since we were using Renderware for the graphics. Working on audio, I did a lot of multithreaded coding for the secondary processor (IOP), which was roughly a tenth of the speed of the EE.

The audio code for the IOP was quite low-level and with our bare-bones Linux environment and tempermental debugger, I was left to debug most of my code via printf's.

The Sony example code was easy to get running early on for the prototype, but required deep understanding to modify for requirements such as streaming with dynamic loading. I believe I fell prey to the classic problem of attempting to extend the life of a prototype well beyond it's natural service life. As a result, I found myself with difficult bugs late in the project which required reengineering sections of the core code.

**2. Feature drop.** Working in Linux made development very solid, but made debugging threads a nightmare. They commonly ended up requiring paper traces. Fortunately for me, we dropped many features to reduce the scale to a more manageable size. We dropped USB audio input, MP3 CD-ROM playback, and CD-DA audio playback.

Many of these features were cancelled in a large part due to Sony. To try to support these features, I sent many emails to Sony worldwide, phoned all sources of tech support and gathered as much information as I could from the Sony Developer Website.

In the case of USB audio, I was unable to gain any sample code as late as January 2001, many months after the OS supported USB drivers. We even paid to have the Japanese documentation translated before the English documentation was supplied in a vain hope to gain enough information to program our own USB audio driver. In the end, we were told to buy a middleware solution at which point it was too late for integration into the product.

We were able to quickly port an open-source MP3 player, wary of the licensing rights surrounding MP3 players. That code, however, would have required significant optimizations to make it run within our product. I began to look into optimizations, such as utilizing the matrix math functions

on the EE, but also began a lengthy investigation as to whether Sony would even allow it, given their strong piracy protection schemes. After weeks of investigation, I was told that we could develop this feature, but they couldn't guarantee that future consumer PS2 units would even read consumer burnt CD-ROMS. Since the MP3 feature would require massive optimizations and a design shift to allow the removal of the DVD at runtime, it was decided this feature should be dropped.



### **Music Controls to select one of 60 tracks of the Ministry of Sound**

It seemed that a good fallback position would be allowing the consumer to insert their own CDs to drive the product's visuals. We all knew that the PS2 would play CDs as easy as it would play DVDs. Wrong. First, I searched the libraries for the function to play audio CDs. The closest function I could find mentioned that it would not read audio CD information and gave no mention of a function that did. When I called Sony, I was told that no function was provided due to undisclosed legal reasons. OK, I thought, I'll just read the raw data, decode it and play it. Eventually, I found out that the low-level read functions would not read a long enough sector for audio CD data, as if Sony really did not want this to work. Entirely frustrated, I was at last informed that if I did find a way to read audio CDs from within our product, we would be possibly sued for reverse engineering their code.

At this point, I gave up on having any external audio input to our product. In many ways, Sony aided our development, especially the audio division of Sony UK. However, Sony also hindered us from implementing features which had initially seemed quite straightforward.

**3. Excessive overtime.** We amassed many weeks of overtime by the time the project was done, due to working hard for each slippage in our final date. This is where a good project manager would have been a strong asset. Unfortunately, things were very unclear from a business point of view, which made a solid final date very difficult to set.

As an example of overtime, I worked days averaging 16 hours for over a month while my partner was away so I could gain enough ground to take a week off to join her. The difficulty with doing one thing constantly is that you begin to dread it and will do anything to procrastinate with little things. One also loses perspective by close inspection for too long. Part of the problem was due to the fact that we sealed the final contracts with Ministry of Sound and UBI Soft very late in the development schedule. I still remember the day that our producer told us 75 percent of the way through the project that we now had six days to final the product. Obviously, things turned out differently.



#### **Circles with motion blur in sync with the music**

**4. Dev Kit Woes.** One of the most difficult hurdles to overcome during development was the discovery that our development hardware was faulty. Part of the problem was my nature to assume that the problem was my own rather than blaming the hardware or other external problems. It is a poor workman who blames his tools, or so the saying goes. Another problem was that the difficulty was intermittent (although it became more and more frequent) and so it was difficult to prove to Sony that we needed new development kits.

Initially we found that our DVD burner's air filter was plugged due to the constant construction dust in our new offices and immediately considered the problem to be tracked down. However, the problems persisted and we continued to hunt for the source of the problem.

Eventually we found that the problem was the same problem that plagued our DVD burner: dust. The excess amount of dust in our office plugged the air filters in the devkits, causing them to overheat and produce sporadic errors when reading the DVD. Once convinced that the dev kits were broken, Sony was generous enough to forward us two working kits within two weeks.

Although this is an extreme example, it should go to prove that hardware is not infallible, especially in the case of somewhat experimental new console development systems. I definitely learned a hard lesson to become more wary of considering more far-fetched potential problem sources.

**4. Poor project management.** Our basic problem with management was that we had no true project manager. Unfortunately, most of the responsibility fell to our lead coder. I remember him twice declaring flat out: "I am not doing any more coding." He even requested that we hire a manager, but this idea was quickly struck down since the team perpetually felt close to completion and we thought we could pull together and do it ourselves. I believe hiring a good manager early in the project and having clearer deadlines would have definitely helped the project, but also believe that a poor manager could have made things much worse.

The first large management mistake was to hire the wrong contractor to do the design and implementation of the core graphics system. What we needed was an strong independent coder who could quickly design on the fly (good luck). What we got was an ideas guy who made us doubt the little design we did have and made no significant efforts to do any useful coding. Fortunately, his contract soon ended which allowed the lead coder room to begin the work in earnest.

One of the largest problems with self-management is deciding and doing the undesirable tasks. Seeing as our team worked well together, we did well with this problem -- though a bit too many of the ugly tasks falling to the lead coder. Another difficulty with our sliding schedule was that many hard tasks were done last rather than first in an effort to get things running early. This was a major management mistake that resulted in core technologies such as dynamic loading and object database to be implemented in the final stages of development.



### **The Visual Mixer allows you to choose and sequence the visuals**

At first we tried to follow the software engineering principles which we all had read about and played with in university (and almost learned). We had one true code review (of the audio), but failed to follow up with subsequent reviews. We were able to keep regular team meetings until mid-way through the project at which point frustration prevailed at accomplishing discussed tasks on time, plus we believed we were almost done.

Nearing the end of our project, our company had financial difficulties due to ripples from investor unrest in the technology market (read: "dot-bomb"). It began with a missed paycheck, then another, then some layoffs, then a round of layoffs for almost the entire company. Fortunately, we completed the product before the final layoffs, but it was very difficult to continue to work difficult overtime hours with no paycheck in sight.

My main issue of non-management was that things were too lax. Part of the issue was the basis of the company in party culture, but mostly due to the tone set by our lead coder and artist. Many times they would come into the office in the late afternoon, be distracted until most employees were gone, work the entire evening and sometimes take the next day off to recuperate. Needless to say, this was difficult for myself and other members of the team who would arrive in the morning, sometimes to find the build broken and have to come back the next day with the hope that it would be fixed. The basic problem was that two shifts began to develop, a day shift and a night shift, which made communication very difficult and relationships strained near the end of the project. Core hours, or at least manageable regular hours would have substantially aided the project.

## What Went Right:

**1 Good design, documentation and tools.** I kept live documentation of the audio design throughout development to help keep the higher-level design clear. I found that when I frequently spent time documenting new design and changes, that it often led to better technical design: "I could easily hack this interface to make it work, but it'll look bad in the design doc...".

Documentation opens up one's code to review and criticism, which is why it is often neglected until the end: when any criticism is moot (if documentation gets done at all). Another reason a lot of coders in small companies don't write documentation is that nobody will read it. In my case, this turned out to be partially true, since to this day, I am the only person to read my entire design document. Having documentation improves maintenance of the code, which in the case of videogames is important when the code is reused and reengineered for the next product.

One of the strongest areas of the audio design was the tool set. Early on, we decided that we wanted a layer of high-level data describing the music, such as the tempo changes, when beats occurred, what instruments/sounds were playing and what the relative intensity of each element in the mix was. Since we were working in a musical context, the logical way to represent these attributes was to use the MIDI file format. In retrospect, this obvious decision was also a wise one since the format is easily readable by many different musical editing programs and the musicians were already familiar with it.



**MIDI even allowed us a simple method to code a system to allow synchronized lyrics: voilà - karaoke!**

Basically, we just lined up a MIDI song with the digital audio and used channels to separate the instrument data. We used different note values to represent if an event was a single occurrence or a particular pattern. The length of the note was used to represent the length of the given pattern and the velocity was mapped to the relative intensity of the pattern in the mix. MIDI even allowed us a simple method to code a system to allow synchronized lyrics: voilà - karaoke!

It took the musicians a while to create the data tracks, but we made every effort to keep things simple. We didn't need to recompose each track via MIDI, just describe the times when the visuals should change. In the end, we ended up with only a kilobyte of control data for each minute of music. Roughly, it took the musicians three days to complete coding the data for each hour long set, which was also very manageable.

Once the MIDI tracks were created, they were converted to a text format using the public domain program MF2T by Piet van Oostrum to ease the parsing process. I believe that it is advantageous to have the output of a tool in human-readable format for easy spotting of bugs and the possibility

to edit the data in a wide variety of programs. The text file was then parsed on the PS2 in a tool to create a binary data image that was loaded directly into memory during run-time. Care was taken to share the structure format between the tool and the game to avoid adding code synchronization bugs. Compiling outside of the game allowed for proper unit testing of related modules as well as optimizing the parsing code out of the game.

**2. Successfully worked at home.** One of the most positive aspects for me during the development of the product, was that I was able to work from home. My partner was originally due after the project's completion, but due to the project's sliding schedule, my son was born half way through the project.

Moderngroove was very supportive in having me work from home and provided me with the proper equipment to make it possible. We set up a secure VPN and I took a workstation home to log into my computer at work and the Linux development server to code. After a bit of configuration problems, I ended up with a very similar development system at home to the one I had at work, albeit a bit slower.

Since I was working on audio, we also set up a ShoutCast system on my work computer and broadcast high bandwidth audio to my home computer over my cable modem connection. We set up a USB video capture on the devkit, so I could even see the output of the devkit at home using NetMeeting. Obviously, I couldn't work on any areas that required close audio to video synchronization due to latency issues, or use the game pad, but was able to design and code areas such as tools and the MIDI synchronization system entirely at home.

In retrospect, I feel that, due to less distraction, I was actually more effective working from home than at work. I spent more time on design since I didn't feel the pressure to always be feverishly coding and looking "busy" working. I find that the code I developed at home is much more reusable and robust since I was more relaxed and focused while coding it.

**3. Company culture.** I have worked at several high-tech companies in the past few years and found Moderngroove's culture to be one of the best. Perhaps it was due to the dot-com mania one-ups-man's ship company culture mania that was sweeping the West Coast at the time, but a lot of the company's ideals for culture truly worked.



**Moderngroove gets down with DJ/CEO John Stroppa**

The basis of the culture was to keep things fun to balance the heavy workload. Of course things changed near the end of the project when the company was in financial difficulty, but people were generally happy to be working at moderngroove.

The company provided us with regular overtime dinner meals and even kept an account at a local restaurant during a particularly difficult work period. There were regular events planned, a full

week's event list posted and a general relaxed attitude. We also had free pop and beer, pancake breakfasts, homemade lunches, overtime meals and weekly free tickets to go out to clubbing. Sounds pretty good to me...

**4. Good Team.** As mentioned in most postmortems, I believe the main reason we successfully completed the product was due to a strong team. Even with the long hours, high stress and sliding project schedule, we were able to stay objective on the larger goals of the product and have fun while doing it (well, except near the end...)

Part of the reason for our team's ability to work well together was likely due to the team being new, but also due in a large part to our company culture. We always tried to make sure we were happy to be working or take a break and refocus. We shared a lot of meals together at local eateries venting about the project or discussing new work ideas. In the evenings, we were happy to kick the crap out of each other, virtually of course, on *Tekken Tag* on the PS2.

We basically kept the same team throughout. Midway through the project, one member of our team was let go, primarily due to personal reasons and a new team member joined to close the gap. Near the end of the project, our producer left for a variety of reasons due to the company. All in all, the changes to the team were gradual and on good terms. As with most small companies, we focused on hiring people who had a wide range of skills and worked well in a team, which paid back well in the long run.

**5. It works.** At the risk of tooting my own horn (since I did the majority of the coding which determines how the visuals synchronize to the music) I still find myself trancing-out to the groovy visuals. I believe the major reason for this success boils down to a few basic design decisions.



**Screenshot of the "Green Monochrome Tunnel"**

One of our first decisions was to attempt to produce a display which had intuitive mappings from

audio features onto visual elements, such as making the visuals faster as the tempo increased and to have larger visual changes when the music was more intense. What we did not want is a simple spectrograph controlling the visuals in an unintuitive and mechanical manner, which is the way most visualization programs operate today.

To implement this, we made human generated control data for the visuals. The drawback of this method was that it didn't allow for arbitrary music to be synchronized and it took time to generate the data. However, since it was a closed-system console project, the non-extensibility was not an issue and even a possible advantage to offer downloadable updates for future internet-ready versions of the product. If we wanted, we could always add an extra layer of computer generated control data later.

To make the process easier, simple extensible tools were developed to speed the process and produce robust data. Strong tools are important to reduce the distance between inspiration and actualization. I believe the audio tools were quite successful at this since there were very few bugs due to bad or missing data.

Rather than having fully control the way the objects reacted to the music, much of the control was left to the artist who defined tables to describe how various elements of objects would react to the visuals. Animations were triggered for each visual element, such as size, rotation, morph frame and so on. Certain object attributes could be defined such that they were more easily modified by changes in the music or be entirely independent of the music. This gave the artist a good level of control to prevent the feeling like everything was overly synchronized to the music.

## Conclusion

Overall, I'm pleased with how the project turned out and even used it for a couple of my own shows when performing around Vancouver to a great response. I'm curious how the product will do in retail since I'm not sure how the public will perceive it: is it an inexpensive 5 CD set release from the Ministry of Sound, a VJ in a box or a way to show off your PS2? I believe that once the idea of a VJ becomes more commonplace (like DJs) that there will be more products like this in the future.



### **"unspoken" plays live with PS2 visuals at Club Sonar in Vancouver**

Looking back on the project, I am quite happy with the results of the music synchronization system but wish that I could have spent less time working out tricky bugs with music playback. It was a rewarding experience to work on a cool new project with a great team in a fun company.

## Game Data



### ***Moderngroove - The Ministry of Sound Edition***

**Publisher:** UBI Soft Europe

**Full-Time Developers:** 4 + contractor

**Musicians:** 1 + contractor

**Artists:** 4

**Budget:** Approx \$0.5 million

**Length of Development:** 8 months

**Release Date:** July 16, 2001

**Platform:** Sony Playstation 2

**Hardware Used:** 533MHz Pentium IIs with 128MB RAM, 30 GB hard drives, Matrox Dual Head

**Software Used:** Redhat Linux, 3D Studio Max, CVS

**Notable Technologies:** Renderware

**Project Size:** Approximately 70,000 lines of code (15 percent is audio code)

*Copyright © 2000-2001 CMP Media Inc. All rights reserved.*