Ultima IX: Ascension

Programming Post-Mortem

5/15/2000

Table of Contents

Ultima IX: Ascension
Programming Post-Mortem

By Bill Randolph, Lead Programmer

Purpose

This document describes the Ultima IX: Ascension dev cycle, from the programmer's perspective, and hopefully provides some insight into what did and didn't work in the dev cycle, where the problems were, and possibly how to prevent them in the future.

It's my opinion that these problems can occur in any genre of game, whether it's solo-play or massively-multiplayer, 2D/3D, RPG or action, etc. So, don't make the mistake of thinking "This can't happen to me; U9 was a completely different game!" Read on, and learn.

History Lesson

To understand the points in this document, it's necessary to subject the reader to a (rather lengthy) history lesson.

Ultima IX began as a delta off the Ultima 8 engine; a 2D top-down game. That version didn't last long; it was decided fairly early to make it a polygonal 3D game, but still fixed-camera top-down. In fact, it wasn't even perspective 3D at that point. This was approximately in late 1995 or early 1996. Work on the engine was begun, as a delta from the existing Ultima-8-based code. Art was developed, and a software rasterizer solution was obtained.

After several months of work on Ultima IX, UO became the "hot item" in Origin. Personnel were pulled off of Ultima IX, and put onto UO. In fact, U9 was actually cancelled by EA. The project director at that time, Mike McShaffrey, added Glide support to the code to see how it would look if it were hardware-accelerated. The result stimulated interest in the product, and the product was kept alive with a skeleton staff of two programmers (Herman Miller and Matthew Lamari), but virtually no management structure in place. This limbo phase lasted about 9 months.

UO eventually shipped. It was at the tail end of the UO shipping cycle that I hired in (July of 1997). At that time, the code was still mostly a top-down engine. Herman had just converted the camera to be a 6-degress-of-freedom camera, to allow you to look around the world more freely. Roofs still popped off buildings when you walked into them; there was no sky. Frame rate was about 8-12 on most machines (P166 to P200), in Glide. The code greatly suffered from having started out 2D, being morphed into 3D, then having a free-form camera added. It was full of dead-end execution paths, stubbed out functionality, and legacy unused systems (like stubs for keyboard interrupt handlers, a carryover from U8 DOS code). The code base was also completely uncommented, which made it difficult to assess its status.

The run-time software appeared, however, to be a running engine, a basic technology demo ready to begin production into a game. The project had an editor ("Ankh") which was extremely feature-rich; it was, in fact, the coolest editor I'd ever seen for a game. Under the assumption that this code was robust enough to build on, we began production with this basic engine and its associated editor. The programming team became myself, Herman, Matthew, Gary Smith, and Jeff Wofford. Our producer was new hire Ed Del Castillo.

With this team, we underwent a TDR process that was very clouded by the fact that we had an existing code base. The TDR was essentially a documentation of the code base, and how it currently operated. It was a learning process for the team, and the company, since it was one of the first TDRs Origin had done. It wasn't very successful; EA felt it was too "light-weight". In retrospect, I agree. Shortly after the TDR, Jeff Wofford left Origin; we hired Dave Aldridge to replace him, and acquired Chuck Zoch, a new programmer who transferred from the design group.

The organization had a very dubious perception of the product at this point. Marketing (Chris Plummer, Alex Carloss) insisted that our product had to have multiplayer support in order to sell well. When Ed told them that we could, but it would add months to our development schedule, a

political conflict arose between marketing and Ed that gave our marketing department a negative view of the project.

After 8 months of work we had our first E3 demo. Preparing for the E3 demo of May 1998 was incredibly taxing. We crunched for several months. This period saw the departure of the entire design team, due to conflicts with producer Ed Del Castillo, and with Richard Garriott. New hire Seth Mendelsohn took over the Lead Design position. For the demo, the programmers enhanced or rewrote some of the functionality of the engine such as lighting and collision, but the engine stayed mostly intact. I found that the code had never been compiled with optimizations on, and the team had no tools for measuring performance, or detecting which subsystems were causing performance problems. The code was also largely uncommented, and unstable; it became extremely clear that major portions of the code needed a re-write (mainly the Usecode). The demo was very crash-prone, and we were unable to identify the source of the crashes. It also was quite clear that we had no understanding of the performance issues of our engine.

Additionally, the animation system was proving incredibly problematic. It was glitchy, jerky, and the code was impossible to understand. But, the E3 demo did show us how the systems needed to operate, and how they should fit together to give us the game we wanted. It also showed us that we didn't have production code; we had prototype code.

After E3 '98, we undertook a rewrite of certain subsystems, mainly the Usecode, the source of most crashes. Chuck Zoch undertook the Usecode rewrite, and suceeded beyond my hopes. Matthew Lamari left, and a new hire, Jim Short, inherited his animation system. We hired in Todd Hayes, who had a level of expertise with 3D programming the rest of us lacked, especially Direct3D. Todd's main task was to put D3D support into the renderer, and to improve its performance. However, after Todd, Dave Aldridge and myself had more thoroughly analyzed our code base, we determined that we could only improve our performance by re-writing more low-level systems, and re-structuring some of the underlying mesh structures in the engine. We began a rewrite of the underlying mesh structure, the animation system, and the renderer. This led to a rewrite of the art importer and large portions of the editor. The changes we made impacted the NPC activity system, the fledgeling combat system, collision, and just about every part of the game.

Todd and Jim were responsible for the primary re-writes: the renderer and the animation system, respectively. They worked for weeks on new systems that would run in parallel with the existing systems. When they felt they were ready to switch the engine completely over to the new systems, we removed all the old systems from the game, and had to comment out huge portions of the remaining code. We then began an amazingly painful "integration" process, which I estimated would take about 6 weeks. This was at the beginning of August, 1998. We targeted Sept 15 as the completion of our integration. This turned out to be a drastic under-estimate. The renderer required a huge number of features to support all the editor's and the engine's requirements and special effects. There were numerous undocumented assumptions in the original code that were very time-consuming to discover and emulate with the newer code. Rewriting one of these subsystems was sort of like re-writing DOS, and having your new version stay compatible with all DOS applications. However, converting most systems, such as collision

and lighting, took relatively little time. But, two unforeseen problems created enormous delays: the art database, and the animation system.

The art database consisted of thousands of Lightwave objects, scenes, and textures. The system rewrites required re-importation of every single piece of art (which would have probably been necessary even if we had not rewritten the engine). But, we discovered that there had never been consistent standards in the creation of the art files; the art had been created over a 3 year period, to conform to the rules that applied to the engine at that time, and these rules changed and drifted over time. In other words, as the code changed, the requirements for the art changed too; but the older art was never re-done or renamed to conform to the new rules. And the rules for art naming and creation were communicated verbally; but, rollover of personnel erased any knowledge of these rules. We therefore found that no single importer was capable of reimporting all the existing art; hundreds, perhaps thousands, of changes were required to be made to the art to make it re-importable. We also found that hundreds of art files were missing, or had been moved, which confused the importer. These problems resulted in 3 programmers (myself, Dave Aldridge, Todd Hayes), and several artists, working for a solid 6 weeks just to reimport the art! Here's a specific example: Animating textures were specified by putting a number into their filename (ie face01.tga, face02.tga, etc). But, a huge number of textures were created that had numbers in their name that were not meant to animate; they were named with numbers because there were several textures associated with a particular object, such as the sky (sky01.tga, sy02.tga, etc). Other serious problems had to do with the way Lightwave scenes had been created. Older scene files were completely incompatible with the way the newer importer worked, and huge numbers of objects in the world looked blown apart or simply absent from the world with the new renderer.

The second major delay in our "integration" phase was the animation system. Like the other systems, the assumptions and requirements of the original system were undocumented and unknown, so Jim Short had to discover them while developing the new system. But the format of the animations themselves were the main problem. During Matthew Lamari's tenure with the animation system, the artists had expressed a desire to use 3Dstudio Max's "Character Studio" application to create animations, rather than Lightwave; and Matthew developed a converter between Character Studio and our engine's native Lightwave format that (sort of) worked, but not very well. One reason it didn't work well was that the skeletal hierarchy used by Character Studio had a different number of elements than our Lightwave skeletons, requiring throwing away a certain amount of animation data; so the animations never looked the same in our game as they did in Character Studio when the artists generated them. Other incompatibilities between Character Studio and Lightwave included different and opposite coordinate systems (right-handed 3DS vs left-handed Lightwave, and the axes didn't match ours), and the fact that the Character Studio animations assumed that the skeletal elements all started life aligned with one of the ordinal axes, at the origin; but our Lightwave skeletons were created already aligned in a human (or whatever creature) arrangement. So, the 3DS animations were relative to axes and initial orientations that didn't match our skeletons. But the most insidious and horrible problem of all was that the Character Studio application (actually a plug-in) applied scale factors to its animation data, at various points in the skeletal hierarchy; and, when it would output its animation data to a file, it would *not* report what scale factor it had used to generate that data or those matrices. And here's the worst part: while Matthew Lamari's code was not well written

(frankly it was incoherent), Matthew had been able to do something no other engineer in the building could do: he was able to reverse-engineer the Character Studio scaling, and his original animation system was capable of (mostly) properly applying the Character Studio animations to our Lightwave meshes.

The fact that a relatively junior programmer had successfully been able to convert the 3DS animations lulled us into a false sense of security; we had no idea how difficult it was going to be. Jim Short worked on the new animation system for months, with little progress. We asked for Andy Mitchell (a 3D engineer on UO2 who knew 3DS Max very well) to help him out; and, the two of them weren't able to solve the problem either. Todd Hayes stepped in to help out, and made some progress; but he and Jim discovered that the final solution would never work if we didn't get the scaling factors from Character Studio. They called the Character Studio engineers multiple times, and emailed them back and forth repeatedly; and they were ***never*** able or willing to provide those values to us. We never knew if they simply didn't understand our questions; or, if they considered it proprietary; or if they were merely inept. We eventually had to conclude that, while their animations looked good when played back in 3Dstudio Max, the data output by their plug-in was, to all intents and purposes, useless garbage. We never successfully converted a Character Studio animation to Lightwave format.

The solution finally came in the form of a conversion program, which was able to convert Character Studio animations to Lightwave format (why could they convert them and we couldn't? Damn them!) So, we could convert the animation; then, we would convert the axis system to ours, then we would pre-transform our meshes to the inverse of the first animation frame (to put the skeletal elements into the same orientation as 3DS expected them in), and then, finally, finally, we were able to see the very first animation properly play in our importer; this occurred on about December 15th, 1998, 3 months later than I'd originally estimated.

After we got the animations playing, the rest of the engine fell into place pretty quickly. In early January, we were finally able to transition the design team from the old editor they'd been using to our new editor, and to begin production on the final version of the product.

While the programmers were struggling with the code rewrites, the political backdrop of the product was turning blacker and bleaker. Friction between Ed Del Castillo and our Exec Producer, Jeff Anderson, resulted in Ed's departure shortly after E3, in May of 1998. Jeff and Richard Garriott became co-producers. Seth Mendelsohn began re-staffing the design team with new members, who all had to ramp up on the complex, feature-laden U9 editor. In Nov of 1998, Richard and Jeff Anderson had to make a trip to EA, for the Key Franchise Meeting. EA was expecting a demo of our progress on U9. But, the engine was still pretty broken at that time; the art wasn't fully re-imported, and no animations would work. The world ran slower due to lack of optimization in the new code, many buildings & objects in the world looked blown apart, or had pink default textures instead of their normal appearance, and characters could only "skate" around the world without animating. To EA, it looked like we'd gone backwards with the project. They once again cancelled the project! Richard fought with them over this during the KFM, and laid his job on the line. Eventually, they gave him permission to complete the project, under the condition that Jeff Anderson would leave the project and spend all his time on UO2. Additionally, Richard negotiated a new relationship with EA, such that they would not dictate

ship dates and terms to us, but rather would work with us cooperatively to our mutual benefit (this lasted, what, 2 months?) So, when Richard & Jeff returned from the KFM, we once again had no producer, and Richard had used some serious political points just to keep the project alive. This was the lowest point in the entire project. The programmers had felt like they'd failed. During a team meeting, Jeff Anderson gave our team the option of not completing the project; he'd lost that much faith in it. We opted to complete it, however.

Around Feb 1999, EA began asking for an estimate of our ship date. Now, here's some more history on that: When I came aboard in 1997, there was some expectation that we'd possibly ship the product by Christmas of that year! That quickly became obviously impossible, though; so, with Ed Del Castillo as producer, we re-targeted our date as Christmas of 1998; this, to EA, represented a year slippage, even though our first date was just a fantasy. Needless to say, we didn't make Christmas of '98, and looked like we'd gone backwards. So, there was a certain amount of pressure to attempt to make Christmas of 1999. This date wasn't forced on us; but, it was made clear that EA had a lull in their product line in that time-frame, and U9 would have *much* stronger marketing support if it made that date. Richard presented this to the team in Feb (or perhaps March) of 1999; and, it looked to us like it was easily do-able. We had the engine & editor running again. A fair number of conversations and NPCs actually worked. Some creature combat was working. So, we told EA we'd be ready by Christmas.

Then, something happened that I still don't understand. Somehow, this "voluntary" ship date became carved in stone, and the threat of cancellation became real again. I re-did the programmer schedule, and Richard asked us to commit to a set of milestones, each one about 1 month apart, leading up to a "first playable" version, then an alpha version, then beta, etc. With Jeff Anderson's departure from the team, EA (or Neil Young, our GM at the time?) assigned us a new producer: Eric Lux. Eric approached the project with the attitude that it was doomed to fail, which is what he'd been taught to believe by those outside the project. He emphasized to us that if we didn't make our milestones, EA would kill the project again. We knew this was our last chance to make this game.

The Milestones read something like "Milestone 1: 3 cities, 2 dungeons, 4 creatures, 10 NPCs, Milestone 2: 6 cities, 5 dungeons, 10 creatures,…". We made Milestone 1 with very few problems, and Milestone 2 with some slippage. Eventually, in (I think) May, we made our first playable version on schedule; although, "first playable" didn't mean the entire game could be played; it only meant that certain features were finally on-line. That year's E3 was a very different story from 1998. The game worked quite well, although Richard was only demoing it in private. But the press received it well, and the company was again excited about the product.

June was the date we were supposed to make Alpha. By EA's definition, "Alpha" meant feature-complete, and code complete, with just limited assets missing; "Major features in minor areas", or some such. EA's definition of Beta was code complete, asset complete, and feature complete; only bug-fixing would remain. Origin had never developed a product with this definition of Alpha and Beta, and U9 was supposed to be the first. Note that we all knew that, in order to make Christmas, we would have to come very, very close to those definitions of Alpha & Beta; Alpha was required in June, Beta in August, and final would be in late September or mid-October, putting us on the shelves for Thankgiving. BUT: there was simply no way to get all the

voice acting recorded by Alpha. So, we knew in May, at first playable, that we wouldn't be close to asset-complete for Alpha, and we'd be lucky to have the voices done by Beta. Given our dates, we had to back-compute how the rest of the project would have to be done to have a hope of finishing. This resulted in conversations being written in a hurried fashion, and voice recording done before the conversations and plot had been played. There was no chance to iterate on the plot or the conversations; once they were recorded, they were final.

When June came, and we were supposed to be Alpha, we weren't close to EA's definition of "Alpha"; we didn't have all the art in place, the world wasn't finished, the voice acting wasn't complete, a lot of creature AI wasn't done, and D3D wasn't even close. From the team's point of view, this was a very confusing time. Apparently (looking back on it), Richard was driving us to complete our Alpha date, but according to a modified version of Alpha; he insisted that we could define Alpha to be whatever we wanted. It's possible that he felt if we moved our alpha date, it would jeapardize the project again; or, perhaps he really felt that the Alpha milestone didn't have much meaning, since historically it had not on previous Ultimas. Jeff A, on the other hand, was insisting that we hold to EA's definition of Alpha, and that by that definition we weren't making our Alpha date. My take was that our Alpha date had become unimportant; we still had the same amount of work to do regardless of what we called it, but other team members (Seth and Eric) felt driven to have EA's actual Alpha definition done by our required Alpha date, and were incredibly stressed. Regardless, we did not meet EA's Alpha definition (as we knew we wouldn't), and U9's Alpha process was viewed by the organization as a fiasco. (Note: this was an opportunity for the organization to observe that we weren't going to make our ship date; but, we missed it.)

We continued to drive to a Beta in late August, and it looked like we might make it. Voice acting would be done, and the programmer schedule showed us code-complete by then; the editor was working pretty well, and the designers were busily finishing their areas. It felt like more happened in these two months than in the previous 2 years! Every time we ran the game, there were new features and new areas to explore; everyone was doing a fantastic job. When August came, it actually appeared we were a little ahead of schedule (this wasn't true, but it looked like it). The game was in QA (although QA was still far away from being able to play all the way through), and our schedules were almost done. So, Richard undertook the task of soliciting input from various folks at EA and at Origin about U9's game play. The team then identified a list of "top 10" things to add to the game to make it a best-seller, and we gave ourselves 2 weeks to achieve this. Some tasks involved tweaking animations or the Avatar in some ways; some involved adding features to the game (like baking bread); all in all, we did an OK job of getting these things in. And, all in all, I'm not convinced it actually helped the product much; but, it probably didn't hurt either.

It was about this time that we diverted resource to work on our "demo". Marketing had been asking for a downloadable demo for months (at least, that's how Eric Lux presented the request to us; it is unknown whether the desire to have the demo downloadable came from marketing, or from Eric). We devised a way to strip most of the game data out of the art databases, and to have the game run with minimal art, and only a couple of maps; Jason Maltzen, a new hire, took on the task of generating the smaller downloadable database, and an install program for the demo. About this time, marketing reported that they'd struck a deal with Computer Gaming World to

have an entire CD dedicated to U9, so could we please make the demo big enough to take up a whole CD? We filled up the extra space with movies and interviews. The demo code was still pretty buggy, and represented what was really an Alpha version of the product; end-users who got the demo were disappointed with how buggy it was (despite warnings about pre-release code). All in all, the demo was a time sink we couldn't afford, and didn't help the product (in my opinion).

September came, and QA hadn't finished a play-through. Things began to look scary. QA's initial estimate of how long a single play-through would take was about 12-14 hours, with about 40 hours for the average end-user. Their new estimate was about 30 hours for QA, and 80-100 hours for an end-user! The world, and the game, were over twice as big as we'd thought, despite our attempts to keep it small and achieve-able. (Richard was later heavily criticized for having such a small world; but, it was the right decision, obviously!)

Also, we ran into serious D3D problems. The cards Todd had used to develop our D3D code worked pretty well; but other cards didn't work well at all, and had numerous seemingly unsolveable problems. For starters, there was no single texture set that could support D3D. Most cards didn't support 8-bit the way Glide did; and D3D didn't support chromakeying, which all our art had been developed for, so Todd had to convert the art & the code to utilize 1-bit alpha (1555 16-bit format); this introduce still more problems on various cards, with bilinear filtering introducing artifacts at the edges of the 1555 textures. Also, the requirement to run with 16-bit textures put a huge performance hit on D3D, so Todd undertook support for S3 texture compression, which (by all reports) would greatly improve our performance. It took him over 3 weeks to get this feature in, and it didn't help our performance at all! In fact, on some cards, it ran slower. And, our development time was gone; the only remaining time had to be used to fix bugs. There was no time to tune performance, or to perform our much-needed conversion to DirectX 7; the code still relied on DirectX 6, even though we were required to ship DirectX 7 on our CD.

In mid-October, QA still had not completed a start-to-finish playthrough. Richard was on a 1-week vacation. The programmers had worked crunch hours, 6-day weeks, for months; we met a technical milestone (I don't remember which), for which I promised them a weekend off. The weekend Richard was out was the weekend the programmers took off; QA and design took the weekend off too. Jeff Anderson came into the building, and decided to see how U9 was going; he was horrified to see that, apparently, no one was working on it. Jeff raised an alarm to Jack Heistand, who began investigating that week. After looking closely at the product, and seeing its potential value, Jack concuded it was very important to ship it on time; so important, that he implied to us that he'd consider cancelling the project if we missed the Christmas date. (Another major factor in this decision was our mandate from EA to move into online-only gaming, and Ascension didn't fit that profile.) Eric Lux was removed as producer, and Jeff Anderson was assigned as our temporary producer. (It is noteworthy that, over time, Eric Lux had become a strong believer in the product; he saw it for the awesome game it was; but Eric wasn't viewed as an effective producer.) Jeff put QA into a 24-hour mode, their primary goal to complete the game's play-through. We began removing many bugs from the bug list, assigning them a "ship-with" status. After about 3 weeks of 24-hour crunch, we began to get versions that QA could

play through from start to finish. Unfortunately, our time was up. Our bug list looked pretty good at this point, but that was for these reasons:
- QA hadn't had time to exercise the rest of the world; they'd only had time to play through the main plot, usually in a hurry, and usually playing only one particular way.
- We'd put a large number of bugs on the "ship-with" list
- We hadn't had the time to properly test the install on machines that had never seen the game before (this requires external beta testing, or acquiring fresh machines, or re-formatting & re-installing Windows on existing machines, all of which are time-consuming)
- We'd concentrated far too much on playing in Glide, not D3D; we had an idea that we had D3D problems, but they weren't "real" to us, and no one was saying "Wait a minute; the game won't run on this or that card!" Glide was more reliable, and ran better; play-throughs were more likely to complete on Glide, and all the dev team had 3dfx cards, so we ran in Glide too.

Under those circumstances, we went "final". We submitted our version to EA CQC. EA CQC failed the version; they tested it for several days, and found some problems with the game that they felt justified holding up the product's ship date. This was the *right* decision, but for the *wrong* reasons! The problems they found were, simply, laughable. For instance, you could install the game, and after it installed, go into the hardware settings program and change your screen resolution to lower than 640x480; then, the game would crash. The install program wouldn't allow this; and, true, it was a bug in the hardware settings program, but it was surely not worth holding up the product for. Also, there was a bag on the bed in Lord British's castle that, if you clicked on it, and jerked your mouse around in a certain way, the game would crash. It took hours for our QA to even reproduce this; and, this also wasn't worth holding up the game for. If CQC had found something that our end-users would have cared about, it would have given us the chance to fix some truly important problems; but they didn't, and we remained ignorant that the game still had numerous serious bugs & crashes.

The final insult was that nVidia changed their TNT2 drivers just days before we went final, and broke our game; the "Microsoft-recommended" method we were using to display alpha textures no longer worked with the latest drivers. We knew then that we were going to have to patch, so we began working on a patch almost immediately, with plans to have it out by the time the game hit the shelves.

The rest of the story is history: the game shipped, had numerous problems with numerous players. Patch 1 addressed the TNT2 problem, but didn't address any of the other serious game issues. Statistically, 1/5 players had no issues, and loved the game; 1/5 players couldn't even run it; the other 3/5 players had varying degrees of problems, from crashes, to corrupt save games, to horrible D3D performance problems. Also, there were numerous plot problems, side-quest problems, creature AI problems (creature's just going dumb and standing there, etc); it became obvious that we hadn't spent enough time searching for bugs, and that we'd shipped the product too soon. Patch 2 addressed numerous crashes & game play problems, but, tragically, introduced another game plot-stopper. Patch 3 fixed that, and numerous other crashes, and introduced full DirectX7 compatibility, and greatly improved D3D performance (but, it broke specular lighting).

To the credit of Origin's management, Jack Heistand agreed to send out new install CDs to all registered users, which would include a remastered version of Patch 3. This helped Origin's

relationship with its now-outraged fans, to some degree. And, to Alex Carloss's credit, he refused to ship in Europe until the remastered Patch3 disk was available; so, Europe had a much better launch than the US.

End of History Lesson.

Analysis of Ascension's Dev Cycle Problems

Obviously U9 had a checkered history. Let's take a look at the specific problems that interfered with the product's ultimate success:

Lack of Time

Ascension's primary problem was that it simply ran out of time. This is a pretty bold statement, considering that most folks believe it was in development for 5 years! But was it? No, actually it was in production for 2 years. I hired in in July of 1997; I spent 2 months on Longbow 2, and a few weeks here & there in transition cycles. I didn't really start on U9 until about October. That's when we had our TDR; that's the time when UO finally mostly wrapped up, and people began to be available for work on U9 again. That's when production finally really began on U9; and, we wrapped in Nov of 1999, 2 years and one month later. This is actually a short dev cycle, compared to most games.

The problem here is that EA, and many others within Origin itself, believed the game was in production since 1994, when U8 shipped. The project could never shake that stigma; to this day, I don't know what we could have done about it, except maybe, just maybe, better "expectation management", if we'd had a consistent producer, or a management champion of the product (see below).

Running out of time created these other more immediate problems with the game:
- The game play wasn't iterated enough; this didn't allow us to balance the game well. Shipping the game just days after completing our first play-through didn't even come close to allowing us to make sure the game's difficulty was consistent or challenging.
- Similarly, the conversations and plot weren't iterated. Recording voices right as they're being written was just dumb; the convos didn't come out like we wanted, pure and simple. If we'd been able to do a solid play-through, and iterate the text of the convos, we'd have easily seen how to improve them.
- Inadequate QA: Ascension should have been in QA for at least 2 months longer than it was. Simply sweeping bugs aside and saying "There! No more bugs!" doesn't help the project. The truth was that QA simply ran out of time finding bugs; they didn't have time to properly exercise the side quests, D3D, or fresh installs of the game. A game that size should certainly have had that; we simply lacked the information we needed when our time "ran out", to be able to intelligently decide whether it was OK to ship or not. The time pressure also gave us tunnel vision regarding our performance; it was much easier to believe that our performance was OK than to admit it would take months to fix it, and that we wouldn't have a chance of making Christmas if we tried to fix it.

Ultimately, these problems conspired to destroy the product's launch in the US, due to the bugs and performance problems. I believe it hurt sales, and it certainly hurt the reputation of the product and the company.

Solution: Manage expectations with the upper echelons. If you think you're going to slip a year, say so. I really believe that's better than signing up for a date that everyone knows is impossible, just to keep the project alive. If you do that, you will run out of time, and lose

credibility. The second half of that, of course, is having an organization that is able to commit to completing projects, and doesn't just pull the plug every time a problem occurs (which, by the way, *will* happen with every single software project known to man! If you pull the plug on every problematic software project, you'll never ship anything. But that's a separate argument.) We didn't have a supportive environment on Ascension; the only way to finish the project was to repeatedly sign up for unrealistic dates. Our only options seemed to either do the project wrong, or not do it at all.

No Production Vision

There was never a consistent production team (producer, assistant producer, and / or development director) on the project from start to finish; in fact, no production team lasted longer than 6 months! There were various producers and managers at various times, but their goals often conflicted with their predecessors', or they simply had no goals other than to get the project the hell out of the way. This actually caused numerous other problems that were extremely destructive to the project. The producer's role of coordinating the various disciplines (art, design, programming) was essentially missing, leaving it to the leads to do themselves. This simply increased our work load; we achieved some measure of coordination and communication, but it wasn't enough. But perhaps most importantly, no one was present to manage the project to upper execs, or to EA.

Solution: This has to come from the top. Make sure the production team that starts a project is top-quality; then, make sure they're kept on the project. Don't underestimate the damage that rollover of the upper levels of exec management has on project management. Rollover of an executive often means that the new exec will disagree with, or even replace, the production management on existing projects, and this does nothing but damage those projects.

Reactive Development

This problem was caused directly by lack of production vision. Simply put, without a management champion to properly present the project to Origin upper management, marketing, and EA, the project could only stay alive by doing the "next critical thing". The dev team was reduced to a permanent mode of "We have to do this <next critical thing>, or we're dead." This resulted in poorly written code, lack of proper documentation, lack of art development standards, and lots and lots of wasted time. The dev team, sifting through the ashes, has often wondered what we could have done differently; and no one has ever believed that there was one single thing we could have changed, even if we'd had fore-knowledge of how it would play out.

Solution: Company-wide awareness is required here, combined with upper management support. Understand that proper development requires a consistent plan. Be sensitive to teams that appear to be in permanent demo mode; that's a train wreck waiting to happen. Allow laying of the proper foundation, and manage the expectations of EA so they understand that we're doing foundation work, not just giving demos.

No long-term art standards

The massive art database, containing hidden landmines of conflicting standards, cost our dev cycle months. Again, personnel rollover and lack of consistent top-down management vision is the culprit here.

Solution: The Leads need to understand the importance of an art standards document, and make sure that the programmers & artists all understand the document, and stick to it. Also, the programmers should count on periodically having to reimport all game art; in fact, this should be done regularly, to ensure compatibility with the standards, and to uncover missing or moved art.

Lack of communication between Programming and Design

Our lack of consistent seasoned producer leadership resulted in a number of miscommunications between Programming (me) and Design (Seth). Specifically, Design had a tendency to over-react to broken subsystems; they seemed to assume that, if something wasn't working, it never would. (This also may indicate a lack of responsiveness on the part of certain programmers to the designers' crashes.) Examples are clocks and lanthorns. We spent a fair amount of time programming a very cool clock, with moving hands and a swinging pendulum; yet, not one single one of those clocks exists in all of Britannia! There is a clock on Earth; and, sadly, that's the only one. The period of time that clocks were crashing caused the designers to pull them all out!

Lanthorns are the same story. They were a very cool floating lantern, that cast a beautiful light; and, they were all pulled from the game because for a certain time period, they were problematic.

The (possibly) worst miscommunication was related to the "placeholder trigger" scheme we devised to improve performance. As a desperate move to improve performance in the game, we (the programmers) created a trigger box that could un-placehold a bunch of objects inside a building, when you enter the box; and re-placeholder them when you leave the box. So, if a lot of the miscellaneous objects in a building were placeholdered, they could be revealed when you get close to the building (close enough to look in through a window, or enter through a door), then re-placeholdered when you leave the buildng. So, all those extra objects don't have to be culled, clipped, or drawn (since we didn't have a BSP-based world). When we conceived of this plan, we immediately were concerned about NPCs and large objects (like tables); if a table were to un-placeholder in the middle of an NPC who's moving around his shop, the NPC would either be stuck, or would teleport to a nearby, possibly bad, location. So, when we presented this plan to Design, we told them not to placeholder large objects. But, the designer who actually did all the work placeholdering things did placeholder all the large objects; so, NPCs in the final game ended up sometimes standing on their counters or tables.

Another communication problem was that Design sometimes counted on features that we either didn't know about, or weren't planning to do. There simply was no mechanism on our team to ensure that Design and Programming were operating to the same game feature set, and this created numerous hiccups and problems.

Solution: Make sure someone's role on the team is to keep the feature set in mind, and to make sure Programming and Design both understand that feature set. (This would probably be a producer; it could be a development director.)

Lack of Proper Benchmarking and System Performance Measuring

This problem is a purely technical one. I don't know who's really responsible, since this occurred as a result of the project running for months on end with no managers at all (during UO development). When I came onto the project, there was absolutely no understanding at all of how the engine performed, what the slow points in it were, how much time it was spending in various systems, etc. This was one of the first things I did on the project; and, it still took us a long time to develop a solid understanding of what systems were problematic, and how to fix them. This was complicated by over-use of memory-mapped Win32 files. All the old U9 files were memory-mapped, and we never had any way of knowing how much of our time was spent in our engine, and how much was spent in the VMM system of Windows; and, we never could measure how much RAM we were using, until we got rid of those memory-mapped files.

Solution: Do this early, or immediately. Always have the ability to measure the performance of any time-critical system. Always have the ability to measure your RAM footprint. Don't ever turn over RAM management (or any resource management, if possible) to Windows; you lose control, the ability to measure it, and the ability to tune it. Even if Windows provides it as a "feature" like memory-mapped files; just don't do it.

The Infamous "Integration" Phase

Re-integrating the re-written portions of the engine wasn't itself our problem during that phase. Our main problems were that I had dramatically under-estimated the size of the task. I think in many ways it would have been easier to rewrite the entire engine than to rewrite just some pieces of it and try to have the new pieces "look like" the older ones, and emulate their features and quirks. Also, the animation debacle cost us more time and credibility than anything else on the project.

Solution: Give up sooner. By this, I mean not to continue to flail at a problem, believing that "any day now we'll solve it"; if it's not working, try something new. If we'd tried other animation solutions earlier, we might have avoided that whole mess. Or, maybe not. Also, this experience emphasizes the importance of art tools. Having tools that make nice-looking art or animations is only one side of the coin; having those tools be able to output solid data is the other. Choose your tools carefully, and make sure their compatible with each other.

Various Software Architecture Negatives

U9 had no scripting language; it relied on C++ for everything, including Usecode and Convos. Having Usecode and Convos as part of the EXE made development slower than it should have been. Convos at least should have been in a DLL! Load/Save was always problematic (it is in every game). Collision was also always problematic. But, the worst part of the engine was the

combination of the overly-complicated (and buggy) "Item Manager", and the overly-complicated (and buggy) "Fast Area". The Item Manager would handle "items" in the world; an "item" is simply a world object, whether a plant, building, or monster. The Item Manager would create items as the engine demanded, destroy them, or move them in the world; it would cache them in or out of memory; it would handle creating collision shapes for them, associated meshes with them, etc. But, it also physically moved them around in memory, and in the map file! The rationale for this was to keep items that are geographically close to the player close together in the map file, to better utilize Windows memory paging. Because items could move in memory behind the app's back, the game had to refer to them through an abstraction called an "Item Handle". This hugely complex feature of the item manager ended up causing some horrible bugs, some of which we never found. Also, we never made some much-needed changes to items, for fear of having to re-create all the game maps (items' math was all integer-based, from the '486 days, and created a huge performance problem on Pentiums; and items never knew if they had a creature AI, an animation process, a movement process, etc; we really needed to centralize all those processes under one concept, but never could because we couldn't change the item structure too much.)

The "Fast Area" is the part of the world that's actually loaded into memory near you, the player. It caches in & out of memory as you move around, and even as you move the camera around. Conceptually, this is the only conceivable way to implement a continuous world without "levels"; but, our engine's implementation was always problematic, buggy, and just plain crappy. It was unreliable, had needless levels of complexity, and unmaintainable. It wasn't very compatible with our collision system, or with the overly-complicated Item Manager.

These systems should have had a single execution path to cache objects in & out of memory, to create their sub-processes & destroy them, etc; but, there were many execution paths, and many different ways of caching in objects; and thus, there were many many bugs with these systems. These systems never worked very well; the underlying concepts were OK, but the implementations were just plain broken. We should have scrapped them & started over (but we didn't have time! Arrrgh!)

U9's Usecode was originally written for "re-entrancy", such that, at the end of each function, some assembly macros would save the registers and the variables on the stack; at the start of the function, a macro would restore those variables, thus creating a sort of hacked multitasking environment. Of course, enabling optimizations in the code completely broke this scheme; and, the macros were compiler-sensitive. Hence the need to completely rewrite Usecode.

Analysis of Ascension's Victories

Wow, with all those problems, the game must be terrible. Not true! Ultima IX: Ascension (Patch 3) is a truly great game, vast in scope, immersive, and really, really fun. How did this happen?

Team Dedication: A Labor of Love

The Ultima IX team was truly dedicated to making this the greatest RPG of all time. Every day, with every game feature, we were thinking "This is an Ultima! This is even the last Ultima! This had better be the best." We loved what we were doing, and we respected who we were doing it with. This applies to art, design, and programming.

A good team can push a product through just about anything.

Consistent Creative Vision

First, let me distinguish between "production vision", and "creative vision". "Production Vision" involves talent in the areas of scheduling, milestone management, representing the team and the product to upper management and EA, and interfacing with other studio departments (audio, translation, etc). "Creative Vision" involves the game design aspects of game creation; what the play dynamics should be, what the scope of the game should be, and the "look and feel" aspects of the product.

Richard and Seth worked quite well together to provide a (fairly) consistent vision about what the game play should be like, what was important, and what wasn't. This didn't always gel with what fans wanted (hey, nothing would have, with Ultima fans). But the result has a consistent feel to it; it wasn't slapped together, the game really feels designed well.

Well-Managed Crunches (well, mostly anyway)

Richard emphasized early on that crunching a team to death is simply destructive, and doesn't help the product. He also observed that during a crunch, the team tends to wander into work at about 11, leave for lunch, work the afternoon, leave for dinner, then work until about 10 or 11. Realistically, you only get about 9 hours of work out of people in that scenario. So we decided to still require folks to come in by 9 AM (or 9:30-ish), and we'd provide a "crunch lunch" to keep lunches short and focused, then work until 7 or 8-ish, then get the hell out of here. You get about 10 or 11 hours a day out of people, and they still have a somewhat healthy personal life. Doing this helped us immensely; also, we tried to make sure crunches only lasted about 6 weeks or 2 months. This was only partially successful, though: the "Integration crunch was actually 4 months, and the final ship crunch was about 4 months. Both of these were far too long for effective productive work, and the final ship crunch was so intense that it actually damaged our ability to work, and impaired our judgement.

The Tools

Ascension has the best editor ever made for a game. I think this is because the programmer on the editor, Gary Smith, was on & off the project for over a year, working on the editor at various times; and, he developed it with the end-user (designers) in mind, so it was full of user-friendly features. Of course, it was also full of bugs, since it relied on the game engine to do anything, and the engine was under development. I'm still not sure whether it's better to make the editor independent of the engine or not. The Editor's network architecture was also incredibly useful; multiple users could edit the same map at the same time, and the file system arbitrated access, and kept everyone up to date. Except for bugs, this worked really well.

Various Software Architecture Positives

Ascension had a pretty cool basic engine design; the Kernel is, in particular, a useful game engine abstraction. Other areas of coolness were the collision scheme (not the code, but the concept); Ascension is about the only truly free-form 3D engine I've ever run across, and it worked really well. Collision was driven purely by the art. The engine also supported unprecedented interactivity with the world. The world itself is the most immersive ever done; this includes visuals, sound, etc. The weather system is pretty cool too, as are the spells and the physics. Also, the interface does a really good job of allowing manipulation of the 3D world.

Lessons Learned and Vows Taken

This section is my personal venting section. I'd like to enumerate what I learned from serving as Lead Programmer on this project.

Never Work Under the Sword of Damocles

The next time someone threatens to cancel my project, my answer will be "OK. Do it." Because the only game worth doing under those hellish conditions is the Finale to the Ultima series. I can't imagine anything else justifying that absurdity and torture.

Re-write Sooner; Re-write Often.

I believe we should have re-written more of the engine. Code always accumulates baggage, and this code had more than its share. The reality is that, if you know exactly what you want a piece of code to do, writing a clean implementation doesn't take that long. It's figuring out what the code should do that takes time. Clean is good, and more bug-free.

Be True To the Product

I remember a CGW interview with (former project director) Mike McShaffrey. He commented on Ultima 8 (another problematic Ultima, for which he was Lead Programmer), and said "No one ever remembers if you ship on time. But everyone remembers if you ship with problems." True, how true! And, we still did it anyway. If I could do it over, the ONLY thing I'd really insist on changing is shipping too early. I'd have laid my job on the line to buy the time if I knew then what I know now. All our work was almost destroyed by that mistake; the product never recovered from its bad rep from shipping too soon, even in Europe. And, there's simply no way to recover all that work and time; there's no way to ever go back & fix it, or do it over. When a game ships, the cat is truly out of the bag. Bottom line: If you're going to spend years doing something, don't blow it at the last second. Follow through. The product is our bread and butter. Don't sacrifice it because of some other goal, or because of "corporate strategy". The only real strategy that matters is to make great products; if you do that, you'll make money.

Dedication is More Important than Anything

Both C&C and Ascension taught me that games aren't made with schedules, meetings, planning, marketing, or even great ideas. Games come into existence primarily with lots of dedication and hard work. Sure, proper engineering design is important; but it won't make the product happen. Sure, schedules are useful, and planning is important; but it won't get you through the tough times. If you don't have dedication & love for your product, you won't get a game made. Making a game is like sailing across the ocean. You can't say for sure when you'll get there; and you *will* encounter storms along the way. The main reason U9 didn't ship in 1996 is that the organization kept changing the goal! If, every time you encounter a storm, you turn away, you'll never reach the goal.

Set your course with wisdom, and planning. Then, *stay* the course, and *weather* the storms.