Postmortem: Sierra Studios' *Gabriel Knight 3*

By Scott Bilas

Gabriel Knight 3 (GK3) is a traditional "Sierra-style" murder-mystery adventure game that tells its story through a complex, nonlinear mix of dialogue trees, scripted sequences, movies, and puzzles. Most of us should be familiar with this kind of game -- it's paced at the speed of the player, and involves a lot of "inspect the monkey" and "use the banana on the monkey" type of interaction to move the story forward. *GK3* differs from its predecessors technologically in a variety of ways, but most important is the fact that it moves the genre to full 3D.

GK3 offers a freely roaming camera that lets players go where they please and zoom in on whatever they like. This isn't just a gimmick -- this single feature changes the game radically, making it more like an interactive movie and less like an interactive comic book. As we found out, moving from 2D to 3D is not just one-third more work, it's more like three times as much work. It affects nearly all aspects of the game, including both the design and the engine. It's not as simple as just drawing your actors and environments with polygons instead of sprites. Suddenly you have to start worrying about camera angles and dramatic effects that were never possible or necessary in 2D, at least not without resorting to a prerendered movie.

Another part of *GK3*'s design was the ability to give the player the option to turn off cinematic camera cuts during dialogue sequences. The idea was that players could be the director and choose their own camera as the action was unfolding. This had a serious and very expensive effect on the art: it meant that artists could take no shortcuts with their animations. In a prerendered movie, an animator has full control over the camera and can avoid bothering with anything that's outside of its view. This saves a lot of time. In *GK3*, because players may at almost any time decide to take control of the camera, they would be able to see the action from any angle they pleased and go "behind the curtain" so to speak. Therefore, the animators needed to make sure that the entire scene could be both viewable and good-looking from any angle. This increased their workload by an order of magnitude.

Previous Sierra adventure games, including *GK1* and *GK2* as well as the *Space Quest* and *Leisure Suit Larry* series were built using the "SCI" game engine. SCI was developed and maintained by Sierra Oakhurst and, for a variety of reasons, Sierra Northwest (the division I worked for) decided to stop using it. This single decision probably affected the project more than anything else did -- it meant that *GK3* was to be the first adventure game Sierra had built completely from scratch in a very long time.

From the start, the project had some important things going in its favor. Sierra hired an experienced engineer, Jim Napier, who got the game started by developing the G-Engine, a 3D rendering, sound, and animation toolkit that provided the low-level foundation for us to build the game upon. After the engine's completion, Jim unfortunately had to leave the project to start on the fledgling *SWAT 3* as its lead. The G-Engine was on the whole a successful part of *GK3*; it provided a stable base for the game and was easy to use and understand. The team was also able to reuse some of the tools and concepts that SCI had provided, such as the content database and lip-synching tools.

Despite these initial advantages, the project faced problems almost from the start. The team had to build a new game engine and most of the related content development tools from scratch, but team members severely underestimated the time, cost, effort, and experience required to construct these tools and establish effective development processes around them. The initial development team was not up to this task.

Hint: Don't walk on that bridge on the left

In the early days of the project, the engineering team must have been living in a magical dream world -- I can't find any other way to explain it. When I joined the project in early 1998, *GK3* had already been in development for more than a year and a half, and it was scheduled to ship at the end of that summer. I realized that this would never happen because at that point the game was a hacked-up version of a sample application that Jim Napier wrote some time earlier to demonstrate the G-Engine. Sample code being used in a production environment should send shivers up the spine of any experienced engineer. Malignant growths of code were added haphazardly whenever a new feature was required, making the game extremely unstable and difficult to maintain. This problem fed on itself and grew worse over time. One example of this problem was the game's horrendous startup time. The file and resource system, while sufficient for a sample application's minimal resources, completely fell over when faced with the tens of thousands of files and hundreds of directories in *GK3*. The game took over a minute just to start up and display the title screen.

There's a big ugly demon behind that veil.

Most of the game's non-art content, such as a story sequence involving simple dialogue exchanges between two characters, was initially hard-coded into the game in C++. This was a nightmare for a couple of reasons. First, engineers were creating content instead of working on the engine, and engineers are generally not the best people for creating good content (and they tend to be very slow at it as well). Second, the tiniest changes to the game, such as choosing a different line of dialogue or altering an animation sequence, required recompilation. This made the content development process unbelievably inefficient. Artists would potentially have to wait weeks to see their work integrated into the game. This resulted in engineers resenting artists "chucking art over the fence" and probably inspired similar resentment on the art side.

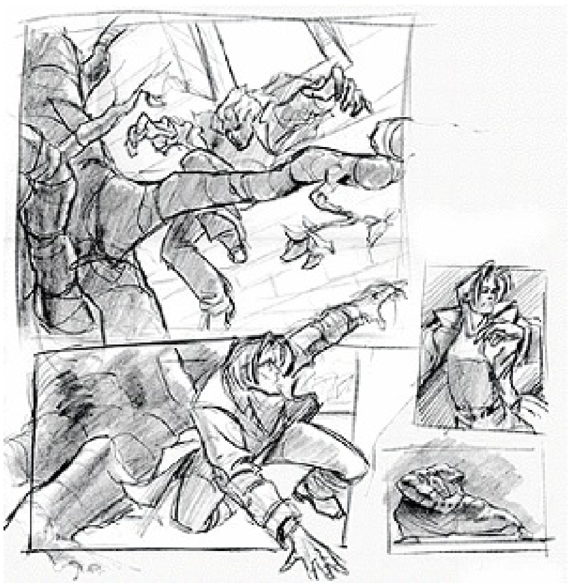
If *GK3* was to ship at all, all of this had to change. And so it did.

---

1. Redesigning the engine. A month after I joined the team, we decided to rebuild the game engine and a little while later I took over as its architect. We spent a couple of weeks in roundtable design sessions with engineering advisors from other projects (including Jim Napier from *SWAT 3*) and used a low-tech Class-Responsibility-Collaborator (CRC) card design technique to hash out the systems we would need and how they would fit together. I thought all of this went pretty well, though it was slower than most of us liked. Once we started coding, though, things really got moving. The application core was rewritten in a weekend, and then individual systems (user interface, scene abstraction and configuration, font rendering, the console system, and so on) were developed and integrated as fast as we could manage.

In proposing the re-architecture, we gained the full support of upper management, specifically Mark Hood, the general manager of Sierra. They really had no choice, considering that the only other option was to cancel the project, but I think it's important to recognize the risk that they took with us and give them credit for believing in our ability to rebuild the engine. Throughout development, Mark was always 100 percent behind us, and never wavered in his desire for us to ship a triple-A title of the highest quality. Despite our lack of experience, we largely delivered what we set out to accomplish.

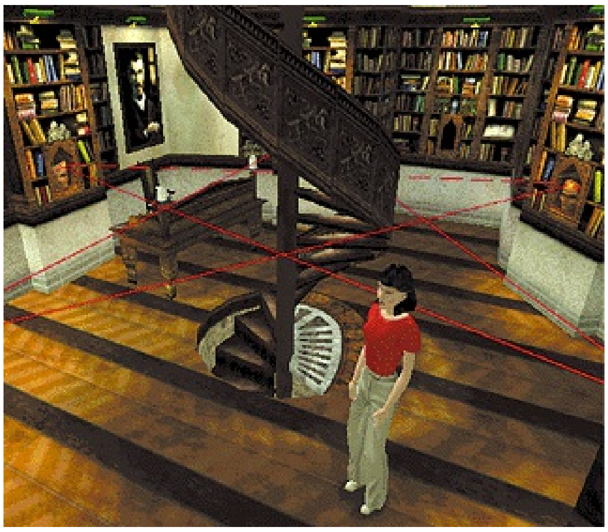
Unfortunately, I don't think a lot of the team members outside of engineering ever really understood why we rebuilt the engine. Looking back, I wish we had spent a little more time explaining to them the dire situation it was in. Although it took months to redesign and rebuild the game engine (a time that understandably confused and frustrated everybody), it ultimately improved our team's productivity immensely and made it possible to ship the game. The new engine was stable, flexible, and although it still had architectural problems (due to our inexperience), it worked properly and performed well.



Hand-drawn concept art.

2. Data-driven engine. *GK3* is a content-heavy game that ships on three CD-ROMs and includes more than 800MB of compressed non-movie data (consisting largely of texture maps, MP3 dialogue and music, and animations). There are thousands of lines of dialogue and almost as many logic rules tying it all together. During the re-architecture, we went with a data-driven approach, putting as much as we possibly could into text files so that non-engineers could create and maintain content. We were very happy with the results.

One of the project's major successes in this area was its flexible scripting system, "Sheep," which used a C-like language and was implemented using a simple compiler and p-code interpreter. The compiler was built with our old friends Lex and Yacc from the Unix world. Originally designed just for the game's animated sequences, the Sheep engine ended up being used for custom rules processing, event handling, resource packaging, scene configuration, debugging, the development console, and even for a little bit of testing automation.

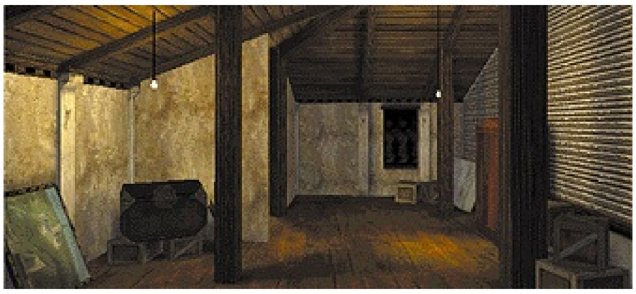


Construct a familiar symbol to open the stairs into the floor in one of the game's cooler sequences.

I can't stress this enough to developers: Build a scripting engine, even if you don't think you need one. Make it generic enough so that it can be reused in as many places as possible. I think many engineers are scared of building one because they think it will take too long to develop or that it will execute scripts too slowly. This was certainly the case with the original *GK3* engineering team. I've found these fears to be completely unfounded -- a scripting engine will pay for itself many times over, and can be easily optimized to approach the speed of C++ code. Also, don't invent a new language. Pick a programming language that one of those "Teach Yourself Something Useful in 21 Days" books exists for, and you can buy copies for your scripters if they don't already know the language (although this wasn't necessary for the *GK3* scripters). This will save you the time you would have spent documenting syntax and training scripters had you used a completely proprietary language.

Another benefit of Sheep was that when combined with our redesigned file and resource manager, we were able to cut down dramatically the time necessary to integrate art into the game. The old engine used C++ to reference art assets, which meant that artists needed to wait for the next build (at best) before they saw their work in the game. We reduced those days or weeks of waiting down to minutes or hours and almost completely removed the engineers from the picture. Under the new system, artists could check in their work and have one of the scripters integrate and demonstrate it immediately on the existing build.

We added clipboard support too, so that developers could use *GK3*'s console to paste Sheep code directly into the game. The scripters could Alt-Tab away from the game, grab a section of test code from their text editors, Alt-Tab back into the game, and paste it into the console to see immediate results. With these kinds of features in place, content poured into the game at a blinding pace.



Poke around in here and you may annoy an arch villain.

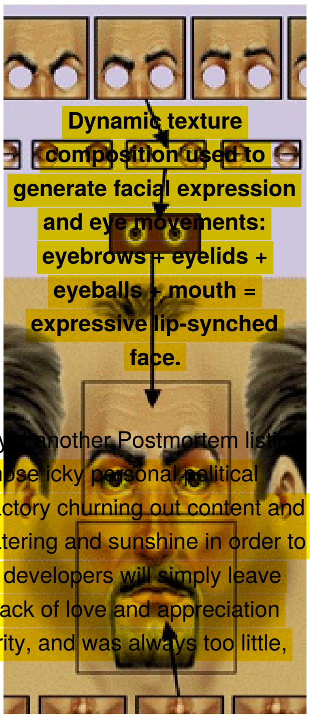
3. The design. The game's design was a major success and deserves special mention. *GK3* would have simply fallen over and died had we had a less experienced designer than Jane Jensen. Throughout the entire development process, the one thing that we could count on was the game design. It was well thought out and researched, and had an entertaining and engrossing story. Best of all, Jane got it right well in advance -- aside from some of the puzzles, nothing really needed to be reworked during development. She delivered the design on time and maintained it meticulously as the project went on.

4. The audio. Audio designers and engineers rarely get the credit they deserve and often end up taking second place to the people drawing the pretty triangles. But *GK3* is an adventure game, and as such it lives and breathes on the ability of its dialogue and supporting audio to immerse the player in the story. Many reviewers picked up on the great audio they found in *GK3*, often rating it as one of the best parts of the game (that is, those reviewers who didn't have a silly personal issue with Tim Curry cast as Gabriel Knight). From a development point of view, audio content was something we could always rely on. David Henry, GK3's composer and lead sound designer, had it all done long before we actually needed it, and was therefore able to spend time polishing the audio and adding lots of small details to it. And in stark contrast to the other parts of the game, integration and maintenance of the audio content went as smooth as glass.

5. A talented, dedicated core of developers. *GK3* never would have shipped without the heroic efforts of critical developers in key places across the board -- art, scripting and logic, engineering, design, and testing. These people took over various parts of the project on their own initiative and kept pushing until things were done and done right. The loss of any one of these individuals would have severely crippled the game's chance of making it out in 1999, if at all. Among the crowd, two names deserve special mention. Halfway through the project, we picked up Jessica Tams as our content lead, who took over the content and put it in order. She wrote nearly all the scripts and logic for the entire game, completed them on schedule, and somehow made them all work despite the problems with the engine (more on these problems in a moment). Lead animator Ray Bornstein came onto the project with a year left to go, put the animations in order, created a realistic schedule, and made the animators stick to it.

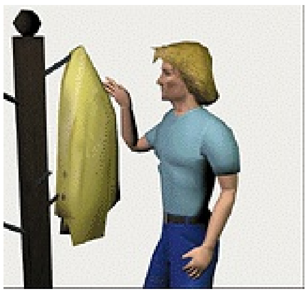
---

1. Team casting problems. When someone is placed in a role in which they don't belong, I call this being "badly cast." Many of the problems with *GK3* resulted from developers being badly cast in their roles, usually because the project requirements were so severely underestimated. To give you an idea of the casting problems we had, consider this: we went through a total of two producers, three art directors (we spent the last year of the project without one), and three project leads (the producer was forced to take over as project lead towards the end).


Dynamic texture composition used to generate facial expression and eye movements: eyebrows + eyelids + eyeballs + mouth = expressive lip-synched face.

This was an ambitious, massive project that required experienced engineers and the original team was simply not up to this task. *GK3* was initially built from members of the *Shivers 2* team (one of the last games built with SCI) and they had practically no 3D experience. Engineers under the venerable SCI engine were basically scripters and putting them in charge of building a game engine from scratch was like feeding them into a furnace. To make things worse, the developers that were in over their heads didn't ask for help, which gave management a false sense of progress.

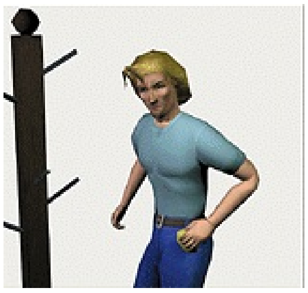
2. Severe morale problems. Hundreds of books and articles have been written about this and here we have yet another Postmortem listing it under "what went wrong." It's time for me to get on my soapbox. To managers everywhere: morale is one of those icky personal political things that many of you avoid dealing with, but you need to understand that your development team is not a factory churning out content and code. To paraphrase Peter Sellers in *Being There*, "The team is a garden of creativity that requires regular watering and sunshine in order to build strong roots." Loyalty is not something that comes easily. The job market is very competitive -- your best developers will simply leave and work for somebody else if they aren't treated well and maintained properly. On *GK3*, there was a serious lack of love and appreciation throughout the project. Recognition of work (other than relief upon its completion) was very rare, lacked sincerity, and was always too little, too late.



Internally, a lot of the team believed that the game was of poor quality. And of course, the many web sites and magazines that proclaimed "adventure games are dead" only made things worse. Tim Schafer's *Grim Fandango*, although a fabulous game and critically acclaimed, was supposedly (we heard) performing poorly in the marketplace. Rumors circulated among our team that *GK3* was going to lose money, due largely to our high burn rate.

The low morale resulted in a lot of send-off lunches for developers seeking greener pastures. *GK3* had a ridiculous amount of turnover that never would have been necessary had these people been properly cast or well treated in the first place. More than 45 developers worked on *GK3* (the average standing team size was 15 to 20), and now, just a few months after it shipped, only seven remain at Sierra. Strangely, the opposite also happened -- several of our developers were included in Sierra's mid-1999 housecleaning layoffs but these individuals were allowed to stay on for a couple of months, postponing their last day until we shipped *GK3*. I believe this was done in good faith out of respect for the developers' hard work up to that point but it ended up being a prolonged drain on morale. Having a small group of people who are (understandably) upset with your company for laying them off and actively looking for a job while still trying to be productive and contribute to a project is a tough situation that should be avoided.

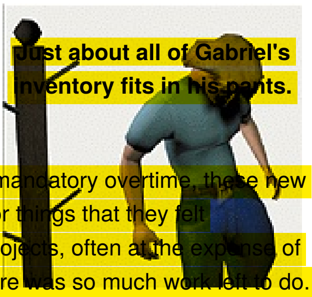
After a certain amount of time on a project like this, morale can sink so low that the team develops an incredible amount of passive resistance to any kind of change. Developers can get so tired of the project and build up such hatred for it that they avoid doing anything that could possibly make it ship later. This was a terrible problem during the last half of the GK3 development cycle and as a result there are many aspects of the game that we aren't proud of. These were problems that should have been fixed but nobody wanted to take the time to correct them because we were so focused on trying to get the game out. I don't think anyone on the team is directly at fault for this and I don't know what we could have done to correct this problem.



3. Schedule problems. Our engineers never had an accurate development schedule -- the schedules we had were so obviously wrong that everybody on the team knew there was no way to meet them. Our leads often lied to management about progress, tasks, and estimates, and I believe this was because they were in over their heads and weren't responding well to the stress. Consequently, upper management thought the project was going to be stable and ready to ship long before it actually was, and we faced prolonged crunch times to deliver promised functionality. More frequent and honest communication within the team would have avoided a lot of this.

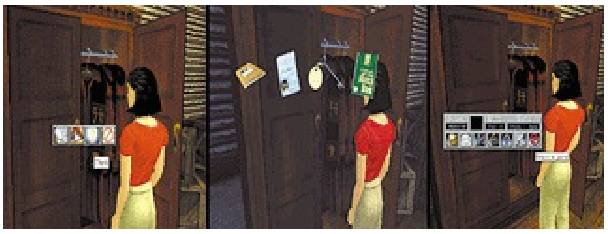
GK3 had few real milestones, which undermined our ability to track progress. There were some milestones very early on in development, but the focus on shoveling visible features into the game turned them into worthless smoke-and-mirrors demos. The concept of milestones eventually was discarded and was replaced with two simple and unofficial goals -- beta and release to manufacturing. In the push to ship the game, we simply forgot about milestones (because "we're almost there!"), put the blinders on, and worked like mad.

Crunch mode is a reality on most projects, but you should not gear up for one unless the light at the end of the tunnel is really in sight. The GK3 team was pushed into crunch mode three separate times, each time thinking that we were almost ready to ship. Most of the last year of the project we spent in this mode, which meant that even small breaks for vacations, attending conferences, and often even taking off nights and weekends were looked down upon. It was time that the team "could not afford to lose." The irony is that this overtime didn't help anyway -- the project didn't move any faster or go out any sooner. The lack of respect for our personal lives and attention to our well-being caused our morale to sink.



Just about all of Gabriel's inventory fits in his pants.

Because of the high turnover, *GK3* always had a high percentage of developers new to the team. Faced with mandatory overtime, these new people understandably felt it was unfair to be "punished" by paying for problems caused by the original team or things that they felt management had brought upon itself. *GK3* became a black hole that sucked in many developers from other projects, often at the expense of those projects. Artists were shifted off the team to cut the burn rate, and then pulled back on later because there was so much work left to do. Our art team finally got on track in the last year of the project (thanks to Ray), but engineering never got a solid schedule together and as a result we were nearly always late in our feature delivery.



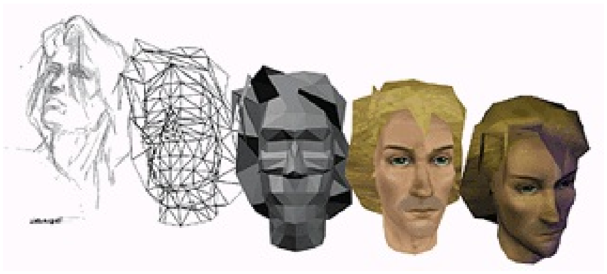
2D interfaces are overlaid on the 3D scene to keep the player immersed.

4. Engineering problems. When we rebuilt the game engine, we tried to retain as much of the original code as we could to get the game up and running again as soon as possible. With the exception of the G-Engine, this was a big mistake. Nearly every one of the systems that we kept caused us problems -- they were badly designed, buggy, inflexible, and should have been redesigned. Some of these systems never worked correctly throughout the lifetime of the project and had to be hacked around by the content developers to get the game to ship.

Specifically, we had serious problems with the "fidget" system (used by character models when idle or when involved in dialogue), the character model's walker, the vertex animation system, and the conversation and dialogue systems. All of these failed regularly and were regularly "fixed," but each bug fix introduced new bugs, usually in the form of hidden time bombs. The engineers responsible for these systems became very defensive about the problems with them, and usually ended up blaming artists and scripters or even other engineers for the cause. Management, thinking that it would save time, often encouraged content developers to hack and work around the problems rather than fix them properly. We should have ripped these parts of the game out and rebuilt them, rather than continually attempting to work around a flawed legacy design. It would have saved a lot of time and hard feelings.

We also faced a lot of problems that were out of our control. Most notable were the technical difficulties with the DirectX drivers provided by hardware vendors for their 3D graphics cards and sound cards, but this probably isn't news to any 3D game developers. These problems were generally features that were implemented improperly or inconsistently, or just outright bugs that caused system crashes or hangs.

We also wasted a few weeks trying to add copy protection. During the final push to ship, we repeatedly attempted to make Macrovision's SafeDisc product work with *GK3*. SafeDisc has a set of special (and we felt completely unnecessary) antihacking measures that got in the way of the game's execution. It heavily affected performance, dropping the frame rate to a third of its original speed and adding strange intermittent freezes of several seconds while the camera was moving. After getting nowhere with Macrovision's engineering department, we decided to ditch SafeDisc and roll our own (which took less than a day to do). This entire process wasted several weeks of our time and frustrated us all the more because, apart from this one remaining task, we were ready to ship the game. Lesson learned: If you are required to use copy protection, don't put it off until the last month, especially if it's SafeDisc. We weren't the first game to have severe problems working with SafeDisc and probably won't be the last, so if you're using this product be sure to do your homework and try it out well in advance of your ship date.

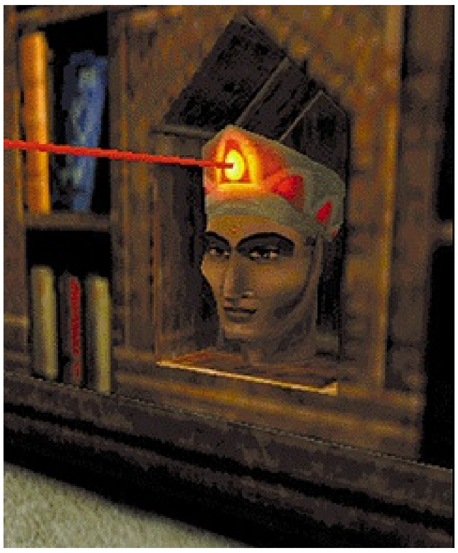


Gabriel Knight, from concept art to dashing real-time 3D model.

5. Art and (more) engineering problems. One of the most expensive mistakes a team can make is ramping up art before the engine is ready. This often happens at large game companies because developers need a place to go after they've shipped their most recent game.

Unfortunately, this was a serious problem with *GK3* -- artists were brought onto the game while the original engine was in development, and long before a stable engine was available. They created content for an animation engine that was untested and in doing so built up enough inertia that we ended up having to keep the design. Later we discovered that the engine's design was seriously flawed.

GK3's animation system is vertex-based, meaning that a model's individual vertices are animated. Even using some creative compression methods, this is very expensive in terms of memory usage. Contrast this method with a typical skinned skeletal animation system, which only requires that the bones be animated. The worst thing about this system was not the memory usage, however. It was the impact on content creation, and the repercussions of this requirement were not fully realized until the art team was ramped up and churning out models and animations.



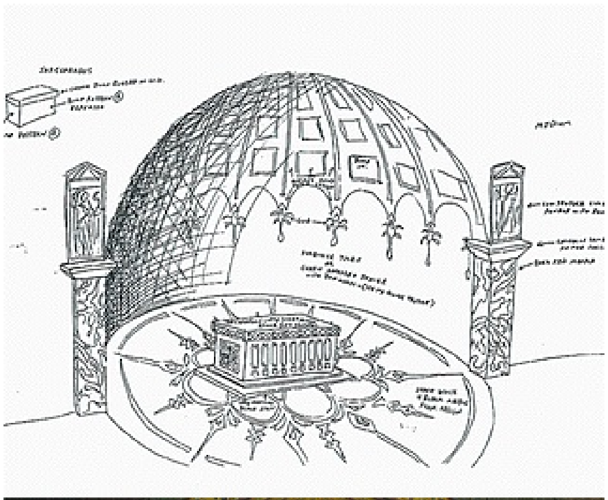
Rendering lasers required a little bit of custom code.

GK3 animations are completely coupled to the meshes of the models that they affect. Once an animation is exported from 3D Studio Max, the models it involves cannot be changed in any way, otherwise existing animations created from the old versions of those models would break. Vertices can't be added or removed, and texture maps can only be tweaked, not remapped. Changing a model required re-exporting every single animation that affected the model, which was very time consuming, tedious, and repetitive for animators, and was often impossible to boot. The source assets (the original Max files) had a way of getting lost and usually ended up stored on backup disks as artists left the project. The end result was that once a model was created, it could never be changed. Consequently, *GK3* shipped with a lot of bad art that the team was dissatisfied with but had to use. An example of this was the Mosely character (sometimes not-so-fondly called "T-Rex man" internally), whose arms were way too short. Just seeing this guy in the game hurt our morale, and made a lot of us feel poorly about the game. The art was bad and there was nothing we could do about it. Nearly every attempt to change a model was met with the response, "That will require re-exporting all of its animations, which will take too long."

What we should have done was just stop the presses and fix the art. When we first recognized the severe problems with the vertex-based animation system, we should have rebuilt that part of the G-Engine and replaced it with a simple skeletal animation system (as the SWAT 3 team later did on their project). We should have also thrown out all of our existing art and recreated it. In retrospect, that would have saved us a lot of time, given us better performance, and made development proceed more smoothly.

Out The Door

Gabriel Knight 3 was an extremely ambitious project combined with an extremely inexperienced team. With all that went wrong, you would have every reason to believe it would never have been finished. Despite all that was messed up with our development process however, we somehow managed to get *GK3* out the door and ship it in time for Christmas 1999. Best of all, the game works and it works well, against all odds. This is a tribute to the testing skills of our QA lead, Matt Julich, and his team. Though severely understaffed, they did a great job ensuring that the game we shipped was of high quality. There are no crashes and no hangs in *GK3* -- Sierra will not be issuing a patch.

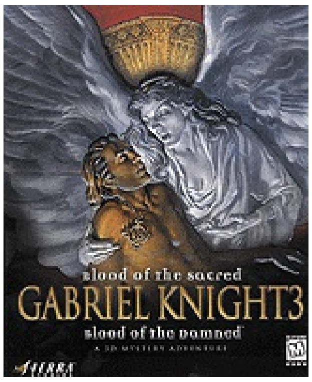


Concept and final environment for the end game.

This isn't to say that *GK3* has no problems, just not problems that we had any real control over as developers. To the best of our knowledge, every single critical problem with the game is caused by either bad hardware drivers (sound or 3D card) which can be fixed by upgrading them, or CD-ROM problems due to our oversized CD copy protection. Also, the first disk that shipped in some retail boxes was difficult to read on some CD-ROMs, apparently due to duplication problems. This can also be worked around, as the same file exists on the second disk and can be copied to the hard drive.

GK3 is certainly not the game it could have been had we been able to knock down a couple of those "what went wrong" issues early on, but it is a high-quality, entertaining game that has been well received by adventure gamers. And that's something we can all be very proud of.

Game Data



Gabriel Knight 3
Sierra Studios
Bellevue, Washington

http://www.sierrastudios.com

Publisher: Activision

Full-Time Developers: More than 45 total, averaged 20 at a time.

Contractors: 3

Budget: Originally well below $2 million, final budget $4.2 million.

Length of Development: Almost 3 years.

Release Date: November 1999 (over a year late).

Platforms: Windows 95, 98, NT, 2000.

Hardware Used: Lots of expensive stuff that quickly became obsolete -- Dual Pentium Pro 200s, Pentium II 266 up to Pentium III 500. 3D cards by Matrox, Nvidia, ATI, and 3dfx, and whatever else we could get our hands on.

Software Used: 3D Studio Max, Character Studio, Visual C++, SourceSafe, CodeWright, VTune, MKS Lex & Yacc, Photoshop, special "sound guy magic."

Notable Technologies: DirectX, Bink Video, Standard Template Library, LZO and zlib (free, open-source compression libraries).

Project Size: 350,000 lines of C++ (not including original engine), 40,000 lines of script and logic, 8,000 lines of (spoken) dialogue, 36,000 unique resource files, 1,500 average triangles per actor, 185,000 cups of coffee consumed

Return to the full version of this article