Postmortem: Resident Evil 4

PRINT

By Yoshiaki Hirabayashi

In this reprint from the October 2005 issue of Game Developer magazine*, Resident Evil 4 *cinematics lead Yoshiaki Hirabayashi writes about the overhauls and challenges which faced one of the franchise's most notable entries.

The *Resident Evil* series has a broad fan base, and in order to meet players' expectations, we decided to create a totally new entry with *Resident Evil 4*. Because the series has been around for so long, we really wanted to address the feedback from both our fans and our development team in order to revamp the game. This meant looking at everything from presenting a new way to experience fear, creating more frightening enemies, implementing new ways of using items, and much more.

For this postmortem, we'll use one element, which was also one of our biggest challenges, as the archetype for the game's development: the successful creation of the title's graphical style. I'll provide an overview of what this entailed and how we were able to achieve it with some specific examples from the game.

What Went Right

1. Cutscene Integration

When we began the project, one area we focused on was how playable portions of games usually shift into atmospheric pre-rendered movies. This seemed like an area that, if done well, would improve critical reception. As gameplay shifts to a cutscene, the change is usually quite noticeable. It's possible that in that moment, players regard what is on screen as just imagery rather than a true part of the game. The change might be appealing to those people who simply enjoy cinematics for the higher quality of the cutscene graphics, but in terms of keeping players focused on the game, it's possible that these moments interrupt the flow of the experience. We thought that if we could facilitate a seamless transition between gameplay and the in-game movies, people would be able to stay involved throughout the entire experience without interruption. Our solution was to keep the cutscenes in real time.

The action button system we implemented for *Resident Evil 4* was very complementary to our use of real-time movies. By incorporating an action button into the cutscenes, we made it possible for players to interact with the in-game movies. In a traditional game scenario, players change from being active participants to bystanders as the cinematics begin and play out. The player might not pay close attention or might even put the controller down, and either way, that's not what we want.

2. Improved Technology

In the current generation of consoles, the technical capacity of hardware has improved vastly over the last, and our

technology itself has also increased to the extent that we can maximize the full potential of that hardware. Technologically speaking, this advancement has made it possible to express scenes in real time that would have previously only been possible in pre-rendered cutscenes, for example those that incorporate complex facial animation. Up until now, we didn't have the processing ability or capacity to realize complex animation of the sort we have achieved in *Resident Evil 4* -- it was simply outside the hardware's capabilities. We solved this issue through programming and by packaging data intelligently. The same solution was applied to areas that required a lot of special effects, such as projection lights and explosions.

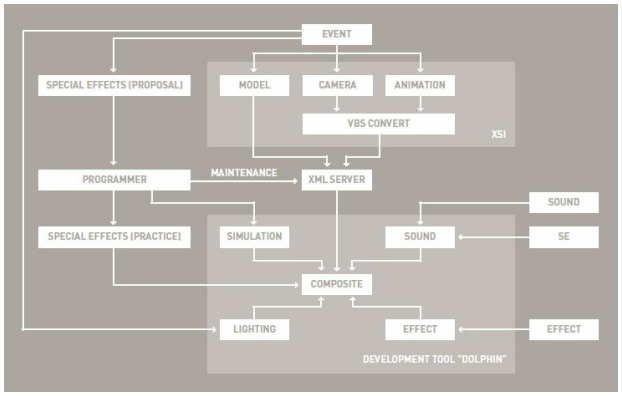
Using real-time movies also made it easier for us to change elements of the story according to the game specs and design. For example, in a pre-rendered situation, if a character or enemy in a movie had to be somehow altered, all the time and energy used to create it would have gone to waste. However, by using real-time movies, we could just rewrite a new model onto the existing model data.

---

3. Improved Workflow

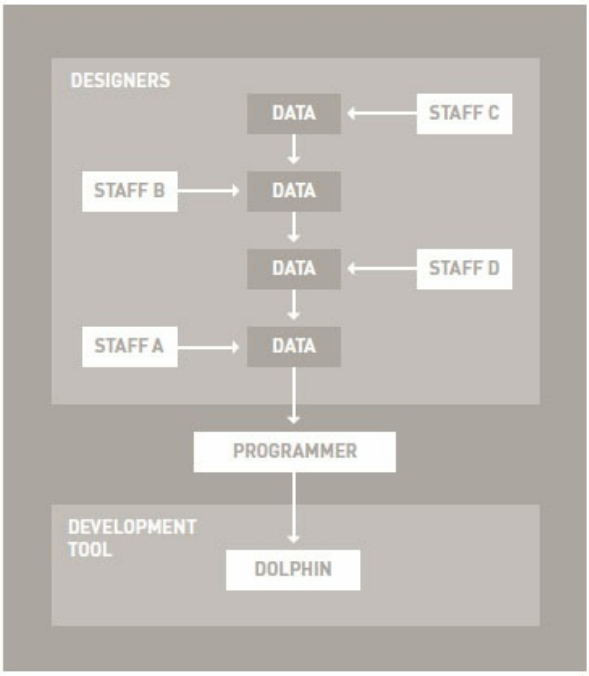
Since we had to basically reinvent the series, we needed to improve our workflow -- converting data, managing data, and troubleshooting -- in order to stay on target. We needed to find a way to complete these tasks more efficiently, which was how we came to use the XML data management system. This workflow management method saved us valuable time that we were then able to apply to the creative elements of the game design.

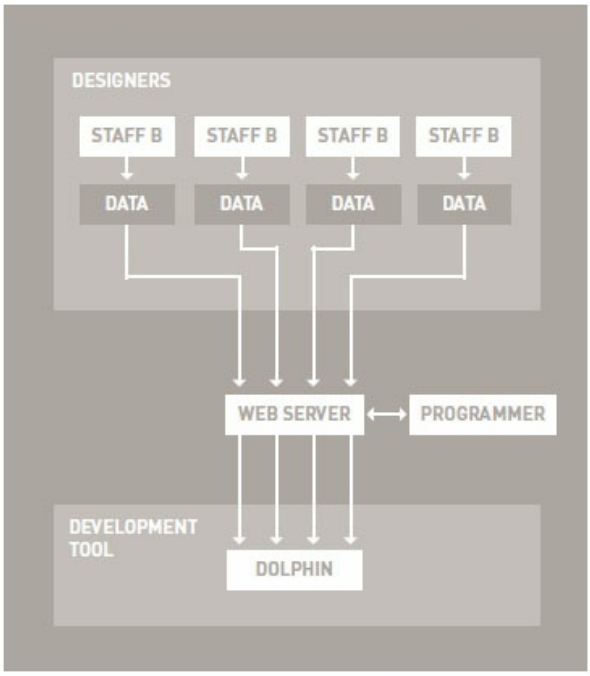
Figure 1 shows the system we used for converting movie scenes, posting scene data, modeling, and making slight adjustments for texture data, among other things. The XML system allowed us to evaluate the total amount of data in order to assess whether there was an excess or deficiency to support it. The system also eased the process of making multiple alterations to the customized models in each of the game's many scenes.



Looking at our previous system (Figure 2), you can see exactly why our new system has helped us. The traditional development pipeline in Japan is still quite hierarchical: With the previous system, you needed a programmer to transfer data to the development tool, which put limitations on the programmer's work capacity. This situation caused a tremendous loss of time.

As opposed to the old flow, which put the heft of the workload on the programmers, the new system (Figure 3) actually resulted in the designers being able to contribute more frequently and more directly, thus minimizing the amount of time lost.





4. Believable Images, Appealing Characters

Creating believable images doesn't necessarily mean that the images need to be factual and realistic. Rather, they should engross players and be believable within the game's universe. For example, in *Resident Evil 4*, a long skirt or long hair moves naturally according to the motion of the characters, but in a slightly enhanced way. This effect helps to immerse players and create a suspension of disbelief. If something swings in the real world, it also swings in the game; however, in real life, it might not move in such an exaggeratedly beautiful way.

We adapted the swings a little and brought the motion close to what players might expect, even though it isn't completely accurate.

In regard to creating more appealing characters, there's a wide range of opinions on the topic. In our meetings with Shinji Mikami, the game's director, he stressed that the most important factor in making a character appealing is to create believable facial expressions and gestures. Characters should exhibit appropriate feelings and expressions for a given situation, while also expressing their individual personalities.

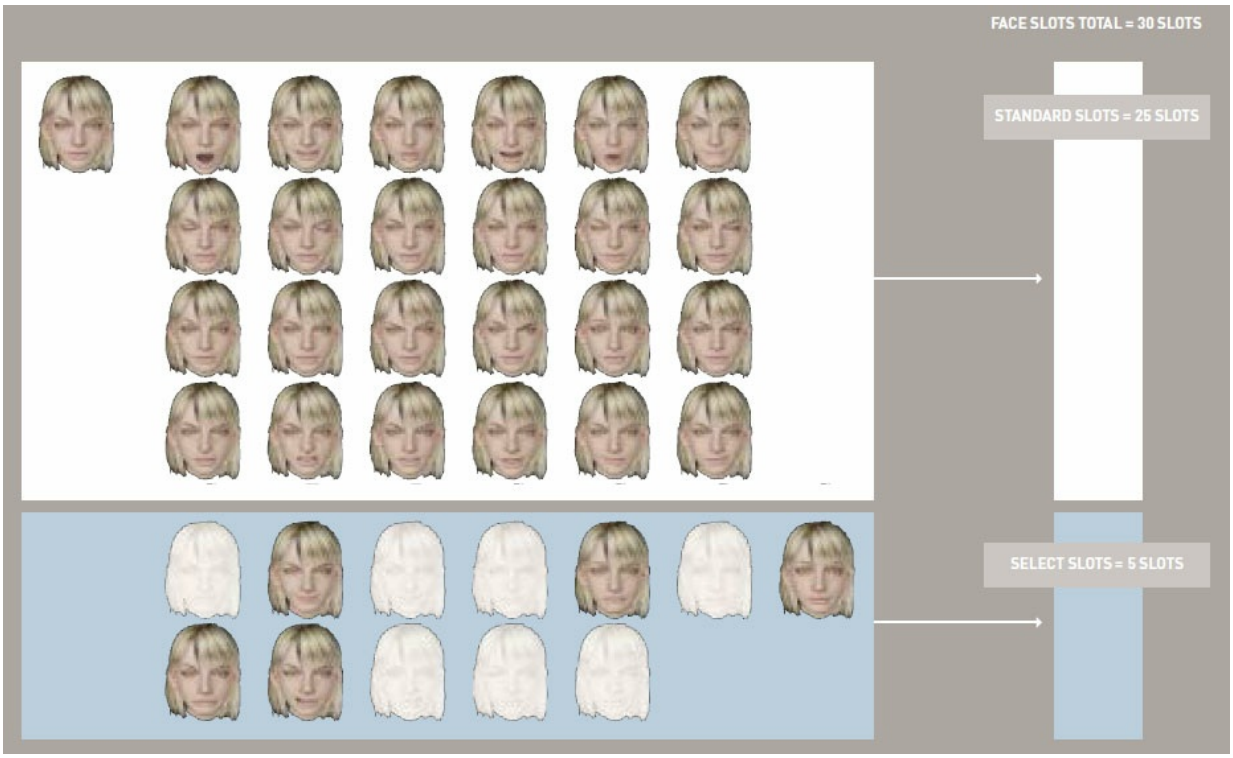
With Mikami's goal in mind, we put a great deal of effort into making characters' expressions believable (see Figures 4 and 5), and to that end, each characters' fingers have joints that move and articulate, for example. Still, we wanted to focus on even more refined details. In order to achieve this, we used a very large number of face targets (or facial expressions) for each character.

At key moments, we conveyed the tension that characters might be feeling by using extra lighting elements to highlight a character's facial expression. We actually created quite a lot of facial expression patterns for the characters. We also used special higher-quality textures to make the expressions look as good as possible.

For Ashley, the main heroine, 36 facial targets were used in total, which is one and a half times the normal number. Regardless of the effectiveness of the increased targets, we couldn't have that much data for just one character, so we used a method called face target packaging. Even though 36 targets were

prepared, they were not all necessary for each scene. We included targets required by each scene into models that implemented them on a scene-by-scene basis.

At first, I talked with a programmer regarding the amount of data which could be used for a character and decided to use 30 targets for each. We divided these into two categories (see Figure 6). One is the Standard Slot, which is frequently used and is the general default slot. The other is the Select Slot, which dictates less frequently used targets for each scene. We made one package with these two slots. Because the number of Select Slot expressions varied from scene to scene, we were able to efficiently manage the data allocation so there were only 30 targets in the package at any given time. By changing the package according to the scene, we were able to produce facial animations with higher quality.



5. Reinvented Gameplay

Even though the series is linked to horror, it also has become known for having strong gameplay and a high entertainment value. We wanted to make *Resident Evil 4* appealing to an even wider variety of players by raising the game's level of entertainment. Early in development, through trial and error, we found that the game was scary, but had a low fun factor -- we needed to rethink both the gameplay system and the fear component. We restarted the project four times in order to ensure that this title would be interesting and fun for the consumer; there had to be more to the game than just fear.

In order to change the gameplay, we challenged ourselves to implement features and elements that had been impossible to realize previously, in terms of environmental interaction, graphical quality, and more frenetic action. We took the existing archetype of *Resident Evil* and added extra play value that would make sense within that world. The main character's melee attacks were made to be more robust because he's a police officer and should be better at close-quarters combat than we had demonstrated in previous games. Enemies were made to be more intelligent and much faster; they can pick up weapons and open doors. Enemies also talk to each other, communicating plans of attack. This was used as a new way to inspire fear for the series, giving the player a feeling that he or she was up against a sentient and unrelenting force, rather than shuffling zombies, or simple monsters.

We had intended from the start to make the project something that we ourselves found intriguing and challenging. In

demanding this of ourselves, and largely meeting those goals, I think we were able to provide an exciting game for the fans.

What Went Wrong

1. Freshen Up

The goal of any game's storyline is to expand the depth and perception of that game's universe, as well as to inform players of what they are to do. In *Resident Evil 4*, the storyline was primarily conveyed through our real-time movies. We had a hard time achieving our concept of combining gameplay with the necessary story elements that could evoke a feeling of believability within the in-game movies.

Games are a visual medium, and throughout the series, it wasn't always easy to create fresh concepts. It was even more difficult to develop visuals that would continue to advance the series. This is why we thought of a new direction for *Resident Evil 4*'s development, which we call "complete concept changes." That is to say, we examined all the previous gameplay features and notions of fear and broke them down to change the entire concept for the game.

2. Data Constraints

Because this version of *Resident Evil* was far more action-based, we often needed to display a very large number of characters at once (see Cutscene Images A–D). As a result, we needed to carefully apply model textures for these types of scenes. Normally, an enemy character in the game has about 3,000 polygons and is about 400K of data space. There are some scenes in the game, though, that had up to 25 characters on screen simultaneously. We weren't able to adjust all the memory and image storage to the level that would be required for us to keep those 3,000 polygon models intact.

In order to address this issue, we decided to adjust the amount of data per model in each scene, and even in cuts within the scene. First, we would implement the model that required the most data (usually the closest to the camera). Then we adapted it by deciding how it would be drawn and processed, based on memory levels. Limiting the amount of data on all the models would result in poor quality, so we prioritized the portions of the models that would require the best modeling. We came up with three different scenarios for how to distribute the data for texturing the models, which could then be used in various situations by combining them.

A normal high resolution model for a Ganado, a generic enemy in the game, consisted of approximately 3,500 polygons. Normally, it would be difficult for us to render several high resolution Ganados on screen simultaneously. Therefore, we created two additional variations of the model: mid-resolution Ganado, which are approximately 2,000 polygons, and a lowresolution model with about 1,000 polygons. These three models would switch depending on the situation. Textures were treated in the same way. High-resolution textures were 512x512 pixels, mid-resolution textures were 256x256 pixels, and low-resolution textures were 128x128 pixels.

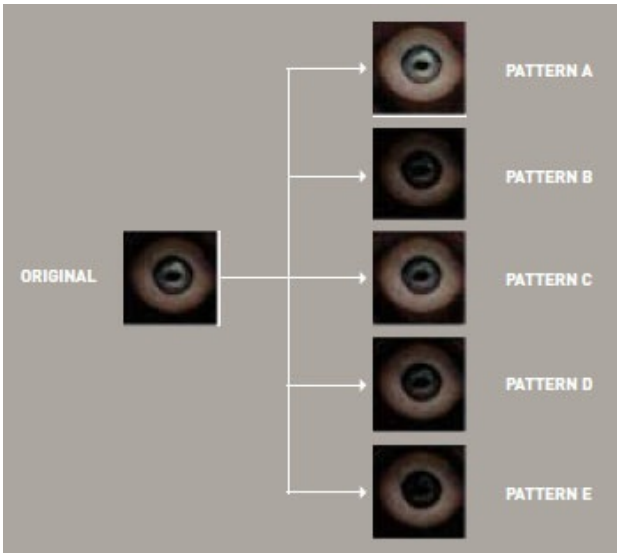
As mentioned, these models could be switched depending on the situation, according to two conditions: how many models the scene required, and how much data was available for the given scene. We always attempted to be efficient with our data sizes, so we would reduce the model and texture data by taking into account the camera and

lighting used for certain scenes.

3. Texture vs. Light

In addition to the models, we also adjusted textures for each scene. We sometimes faced problems with how to adjust the way that characters looked when different lights hit them. This happened especially when we used a lot of penetrating or spot lights. We solved this problem by creating textures for each scene or each cut.

Not only was this used for something as detailed as eyes (Figure 7), but was used widely, anywhere from the characters' bodies, to background objects. This texture adjustment was also useful in addressing problems with dynamic lighting situations, such as when dented objects were being lit too brightly. In cases when a model would only be seen from a distance and required little to no shading information, we reduced the data amount because the dimensions visually do not stand out very well, and the difference would not be noticeable. In another case, when a character's shadow would stand out due to the camera's position, we wouldn't use the low-resolution model, as it would not be capable of properly rendering the delicate lighting details. Instead we would use a high-resolution model with low-resolution textures, still keeping data use down.



---

4. Depth of Field

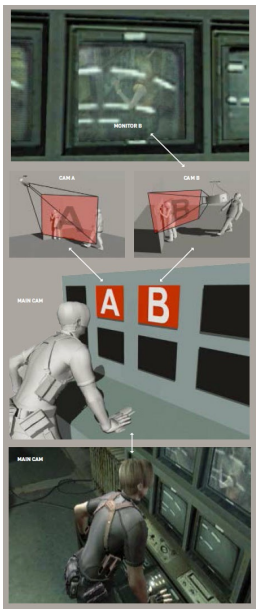
Another problem we had to address was related to depth of field within a given scene. One method we used is a blurring effect: We displaced several pieces of a given image at various depths of field when no camera movement was happening. In this case, processing power was light, and the hardware was not overly taxed.

The more difficult scenarios came up when manipulating depth of field in a scene that required camera movement. In these cases, we specified a distance from the camera past which we would blur the image. We dynamically blurred the image itself, just as you would in Photoshop. The method was artificial, but it enabled us to convey a depth of field that appears in the actual in-game camera. The processing power was very heavy in comparison with the first case, and therefore, we used the blur only when we needed to move the camera and adjust the depth of field from a creative standpoint.

5. Multi-Camera Strain

Some situations required real-time rendered textures, which in turn required the use of multiple cameras. By using multiple cameras, the processing strain is increased by about one and a half times the amount used with just one camera. Despite this, we decided to use this method because the workflow system we utilized made it very efficient to adjust the images. This technology was used in a scene in which Leon looks at a monitor on the screen. For the "video" displayed on the monitor, we didn't use pre-rendered movies, but animated scenes which were rendered to a texture in real time. With this method, we did not have to render character motion on the monitor in order to change elements of the scene, and so we were able to make changes quite easily.

Normally only one camera's data is used for one cutscene, but data from three cameras had to be used here (see Figures 8 A–D). One was the main camera, which projected the entire scene. The second was camera A, which created the image shown on monitor A. The third was camera B, which created the image displayed on monitor B. The screen images which were created by camera A and camera B were cached by memory, scaled down to a lower resolution, and used as the texture for the monitor. After that was done, we compiled the image for the main camera view. This technology was also used for reflections in things like sunglasses and car windows, with translucent adjustments to cached image data.



That's a Wrap

As you might have gathered, the development of *Resident Evil 4* was not necessarily based on innovative new technology, but rather on efficiency. Our improved workflow and our intense focus on details in the game allowed us to achieve the level of quality we had challenged ourselves to produce. Restarting the game multiple times allowed us to take several fresh perspectives on the game, and the survival horror genre as a whole. I think that ultimately we came up with something that was not only enjoyable, but which also helped to advance the series in a positive direction. The next *Resident Evil* is planned for next generation consoles, and will present a whole new host of challenges and opportunities. Hopefully we will once again be able to meet our own high expectations.

Return to the full version of this article