Seokjune Hong
CSC 491 + 492
6/1/2011

Ghost Project

Seokjune Hong
CSC 491 + 492
6/1/2011

Intro

The game ‘Ghost’ is an reality based Adventure genre that emphasizes exploration over action. Adventure games were the main driving force behind the gaming industry during the late 1980s and early 1990s[1]. However, with the advancement in 3D technology the focus was shifted from Adventure to Action genre during the mid 1990s[2]. This is due to the fact that graphic development was focused on fast movement instead of detail orientation that adventure games needed[1]. Therefore, starting from the mid 1990s till current time of 2000s, adventure games are in great decline. However, there are some games that are shining even in this decline. Games such as ‘Gabriel Knight’, ‘Grim Fandango’, and ‘The Longest Journey’ offers an incredible mixture of imagination, detail, and possibilities[3].

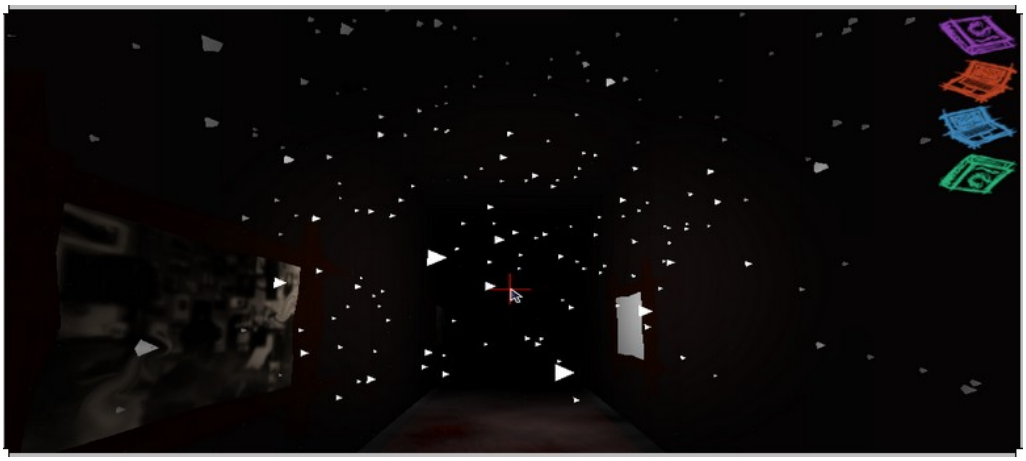
When our team was proposed with the idea of having a game based on Cal Poly, we immediately believed the exploration aspect of the Adventure game fits perfectly with the rich hundred year history of Cal Poly. To have a central location to focus, our senior project team selected the Robert E. Kennedy Library. An exploration of the Library seemed to fit perfectly to our goal of making a game about Cal Poly.

This game was developed by five members the first quarter and three members the second quarter. The first quarter developer were Nick Feeney, Billy McVicker, Michael Boyd, Seokjune Hong, and Jessica Wong. Nick Feeney and Billy McVicker, unfortunately, were not availiable the second quarter so we had to work with only three members the second quarter. The team’s distribution of work was based on individual team member’s strength in a given technology and was distributed by the Team Manager, Nick Feeney and Jessica Wong.

Seokjune Hong
CSC 491 + 492
6/1/2011

Project Overview

As previously stated in the Introduction part, the game's genre is 'Adventure' due to the obvious advantage of having a great background story and the need to have an exploration embedded to the game.



This project has been introduced by Department of English professor David Gillette as a collaboration between different majors at Cal Poly. The proposed goal was to create a virtually augmented reality so an android or iOS user can walk around Cal Poly's campus to play mini games and hear the history of the university. However, due to the lack of Android phones and iPhones for the team, smart phone development was impossible. Therefore, the game was based on C++ with OpenGL.

The story for the game is that the player's companion ghost cannot pass on and the player cannot leave the library until the ghost's last school project is completed and turned in. Therefore, the objective of the game is to collect four components of the project will be distributed on a floor of the Kennedy Library. To help the ghost complete the project, the player must explore the library and defeat enemy ghosts who will try to steal or prevent the player from obtaining the items.

Since the game incorporates ghost, the look and feel had to become dark and scary. There is no light source inside the library except the flash light that the player holds. Scary animation picture are added to enhance the scary feeling for the game. Since sound has the greatest scare effect on people we added a fearful soundtracks and sound effects in various location.

The basic rule of the game is as follows:

- Must collect 4 items on each floor.
- Confined to the inside of the library.
- The ghosts will attack when the player is within a certain radius.
- "Destroy" the ghosts by shooting them with a light burst.
- You can shoot only when the battery bar is not empty.
- After a beam is shot, the battery bar will empty and recharge.
- Win by collecting all the items.
- Lose by losing all your health.

Ghost was primarily based on C++ language using the OpenGL library. However, using that library seems problematic for inserting sound, so we used open source irrKlang to add sound. Since, Ghost heavily emphasize pictures Gimp was selected for adding animation to the pictures and other textures in the game. Maya is used to model all the 3D objects, then we exported to Blender for animation on the ghost.

Related Works

Various games use the flashlight in different ways[5]. ‘Dead Space’, an Action genre, made by EA’s Visceral Games has a flashlight mounted on the gun to help the player navigate through the orbit space station. As an Adventure genre ‘Fragile Dreams: Farewell to the Moon’, the player controls the flash light with a Wii Remote and is used for to scour the ruins for items, clues, and

enemies that are often only detectable with light[6]. The most similar game to the ‘Ghost’ is the ‘Silent Hill’, which is an adventure and horror series. In ‘Silent Hill: Shattered Memories’, the character uses the flashlight to look around the world[7].

However, most games use the flashlight as an accessory rather than the primary weapon. In a horror game flashlights mostly used as a light saver[5]. So in this game the importance of the flashlight has been greatly highlighted. Not only is the flashlight used as the traditional way as a visual aid, it is also used as a weapon. An orb of light is fired from the flashlight to kill the enemy ghost.

The particle system used in this game is based upon Professor Chris Buckalew’s class, Computer Animation CPE 474. Also, the animated pictures was provided by the University Archives. Collision Detection and View Frustum Culling was coded thanks to the tutorials of NeHe Productions.

Algorithms Overview

The following is a list of all technology in this game.

- View Frustum Culling - Billy McVicker
- Particle Systems - Seokjune Hong
- Projective Texture - Nick Feeney
- Animated Pictures - Jessica Wong
- Sound - Jessica Wong, Michael Boyd
- Collision Detection - Billy McVicker

- 3D Model - Billy McVicker, Michael Boyd
- Animation - Billy McVicker
- Map Layout - Billy McVicker, Michael Boyd, Jessica Wong
- Mist - Michael Boyd
- Ghost Re spawn - Seokjune Hong
- Ghost Stealing - Seokjune Hong

Algorithms Details

Particle Systems

Particle systems are large number of particles, each responding to forces in its environment. Forces can be actual physical forces or “behavioral rules”. Each particles is modeled with mass, velocity, and other characteristics. Particle can represent different objects or different locations on one object. Some type of forces are unary forces, binary forces, and N-ary forces. Unary forces are forces like gravity, wind, air drag, fixed-point attraction. Binary forces are spring attraction or repulsion and N-ary forces are flocking and multi-body gravity. Particle systems are useful to simulate real life event and therefore crucial for explosion, tornado, or other force events.

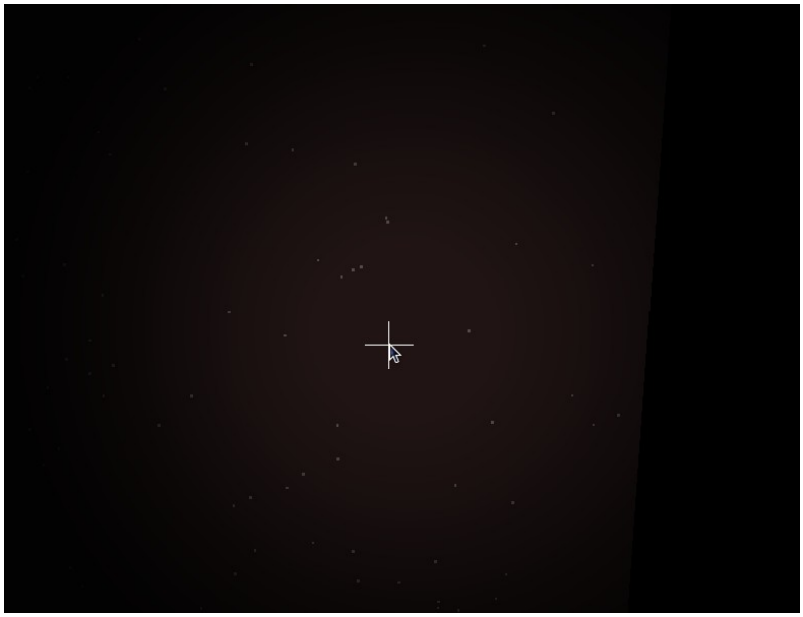
Two particle systems, dust effect and death of ghost, were implemented in this game. Both particle effects have an O(n^3) operation. Particles are stored in a particle structure array with a predetermined size. The particle structure has a boolean type called active; float type called mass, time; and float array of size three for each position, velocity, air, acceleration, and force. The reason for having an array size of three is to simulate the x, y, z coordinates. Active is used

to check if a particle is still needed to be drawn. Mass is to check to give a particle a certain weight to simulate real life gravity effect. Time is used to interpolate the particles position, velocity, and acceleration. Force array is used to input different force acting upon a particle. Air array is the drag effect to a particle. Each particle have all different structure to act as a truly separate real life simulation.

First iteration loads up all the initial value on particles. Second iteration goes through each particle and depending on each location and time adds different forces to the particle's force array inside its structure. Here the drag and gravity effect is also added. Finally, the third iteration is the Euler Method. If the particle is still active, the particles acceleration is calculated by dividing the force of the particle with the mass. Next velocity is found by adding the initial velocity with acceleration times particle's time. Lastly, the position of the particle is found by adding initial position with particle's velocity * time + 0.5 * acceleration * time^2. Both dust and explosion does not have a mass because in reality dust's mass is so small its negligible and explosion particles goes upward so mass has been set to zero.

Seokjune Hong
CSC 491 + 492
6/1/2011

When a ghost dies it explodes in every direction and its particles go up. To have this kind of effect the initial velocity of each particle is set to random on its x and z direction. Afterwards a reverse gravity effect is applied to the particles. After certain y is achieved the active boolean is set to false. So particle exceeding the player's vision is not drawn.



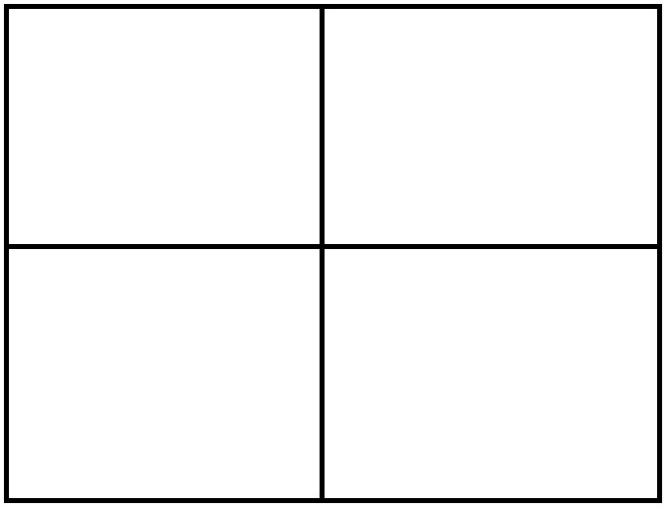
Dust effect is different from the exploding ghost. Instead of getting drawn in the 3D environment, dust is applied on the HUD of the game. Therefore, there is no z coordinates for the dust effect. Each particle is given a random velocity and position inside the screen size. In case the particle escapes the particle its values are initialized. At first, the dust particle were set to white color with pixel size. However, this caused the particle effect look like snow instead of dust. This was adjusted by making the particle to a square and a grey color. The particles still looked like ash falling from the sky instead of dust inside a library. Finally, by adding a random 90 degree rotation on the particle some disappeared and reappear causing the dust to look realistic.

This algorithm is not optimized because the number of particles does not justify optimization. Explosive effect has 300 particles and dust effect has 400 particles which does not affect the

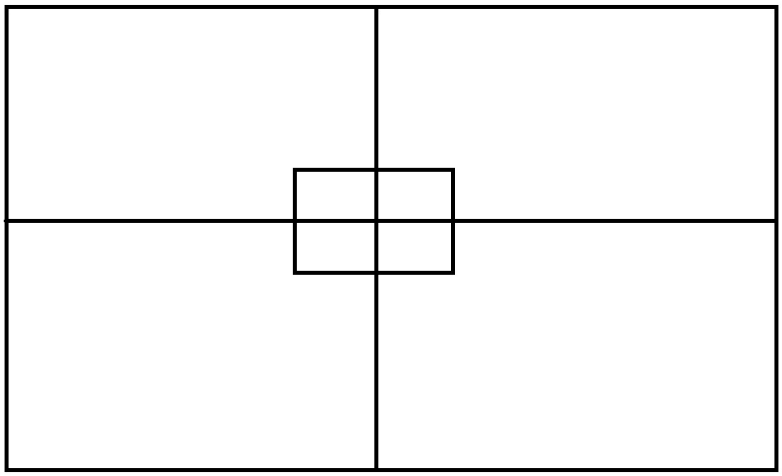
game speed at all. To cause a slow down the number of particles have to exceed at least ten thousand. If the game had a tornado an optimization might have been needed.

Ghost Re spawn

In the beginning ghosts were re spawned at any location inside the map. This caused the game to become impossibly hard due to ghost re spawning at the exactly the same spot that the player just killed a ghost. Since the player used all the battery to kill the ghost, he or she could not defend themselves if the ghost starts attacking. So the whole map was divided into four quadrants and made sure a ghost re spawn in random location in one of the three quadrant that the ghost did not get killed. However, this poses a problem when the player kills a ghost in the at the line between quadrants. In that case, the ghost could re spawn too close to the player. So the algorithm had to be improved to account for the intersections.



To compensate another 2D box had been placed around the player. This box does not really matter when a ghost dies inside a quadrant, but as depicted in the picture on the right; between different quadrants this 2D box help out by creating additional restriction on the ghost re spawn. A question arises what will happen when a ghost is killed exactly between the lines. Which quadrant would the algorithm choose? The problem is solved by thickness of the dividing lines. Since the dividing line is only a pixel thick the dilemma on which quadrant the ghost is killed in is negligible.

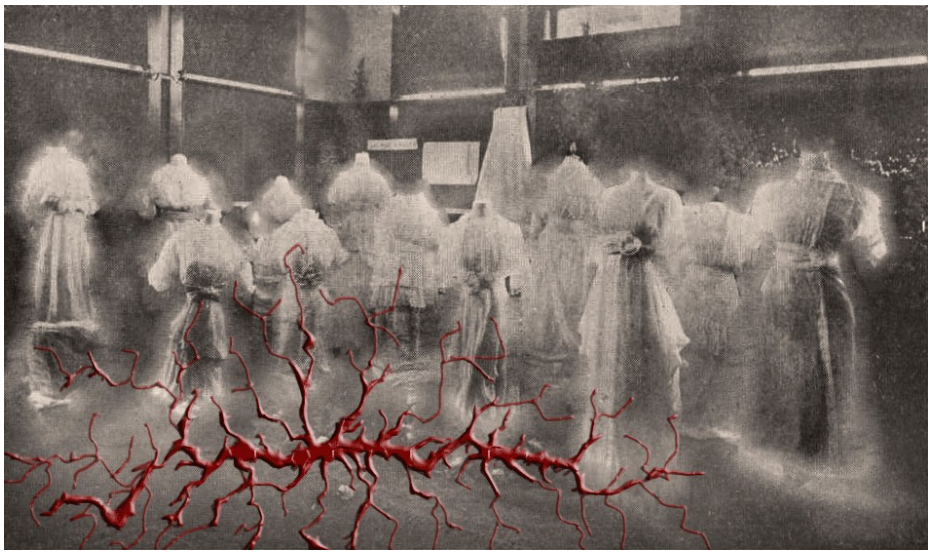


Ghost Stealing

When the second floor was added to the game, just having a ghost damage the player's health points did not seem different than previous level. To add some flavor to the game the difficulty had to be increased. That is why the ghost steals items from the player at the second level. This is accomplished by using a switch cases with the number of items the player currently holds. To balance the game, the ghost could not steal more than once from the player. Therefore, a boolean value was added to the ghost class that would indicate whether the ghost stole from the user previously. This approach had one huge defect. If a player gets attacked right after getting an item it could create an infinite loop by creating a ghost after every time the item is picked up

locking the player at the location. This was solved with adding a timer to make sure that ghost does not spawn next to an item two times without an ample time in between the spawning.

Results



Through this two quarter project, I better understood decision making process on the direction of development, and a need of a good project manager. Once a direction has been decided the manager needs to make sure that course of direction is not diverted. Obviously, I learned much about game and graphics technique like 2D occupancy grid, particle systems, and lighting. An unique lesson from a interactive entertainment project is the importance of factors excluding graphics. A good graphics can never hurt a game, but a balance between story, playability, and etc. has to be established to have not just a good but a great game. I am proud that our team created two level of the library, have a flashlight, two particle systems, a 2D occupancy grid, view frustum culling, many scary animated pictures, sound, 3D Models, animation, mist, ghost re spawn, and ghost stealing.

Seokjune Hong
CSC 491 + 492
6/1/2011

We are extremely proud to have developed an interactive game that students can enjoy with not just great aesthetics, but also a great sound system. The look and feel of a dark library is perfectly catch ed in our game. The darkness surrounding the flash light inspires the player with fear. No other team did use lighting as effect as our team, using as a hinder to vision.

Our team also did not play testing and got feedback from staffs at Dream Works and student. Dream Works complimented on the ability to maximize the effect of sound. They also gave the team the idea to use a blast of light instead of a sphere coming out of the flash light. These are some comment from students that play tested the game 'Ghost'

1.Was the game stable (did not crash)?

Response	Number of People
Positive	5
Negative	0

2.Was the game fun?

Respons e	Number of People
Positive	4
Negative	1

Seokjune Hong
CSC 491 + 492
6/1/2011

3. Was the game too easy? Too difficult? At the right level of playablility?

Response	Number of People
Positive	5
Negative	0

4. What suggestions do you have?
   - Ghosts trap you means immediate game-over -> consider secondary emergency attack?
   - Perhaps some sort of hit or help as to where to go. Maybe not directly but somehow.
   - Mini-map! I felt like I was wandering blindly in the dark w/o any direction
   - Maybe slightly more light to orient the player more
   - Balancing the # of ghosts in the player's area vs. how much ammo is available
   - Lower mouse sensitivity
   - Add running option

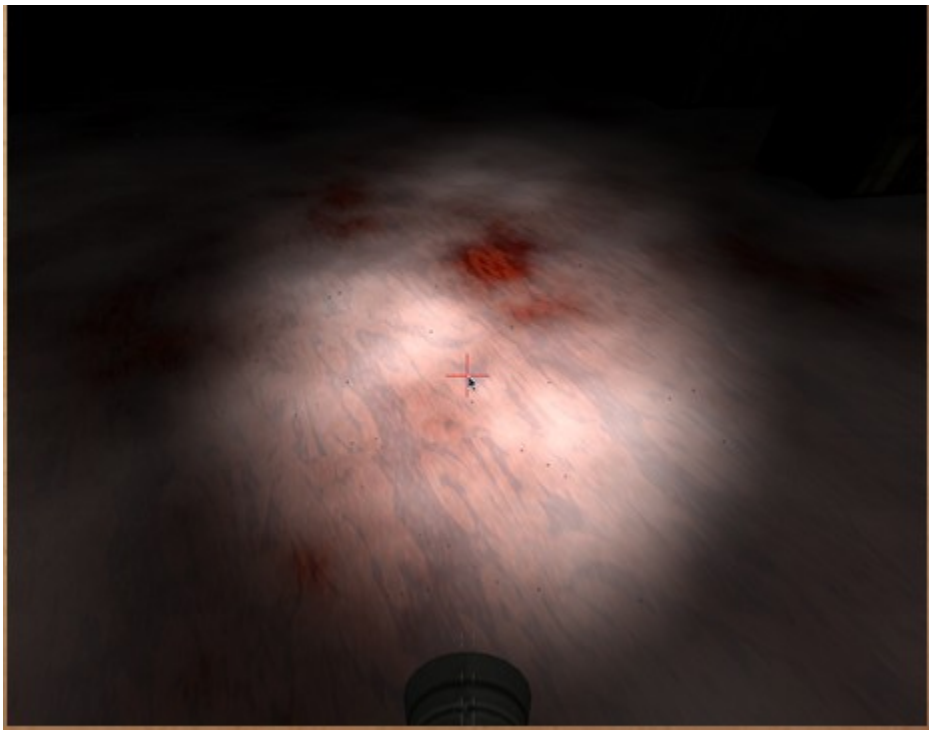
5. What did you like about the game?
   - It was unique and played smoothly.
   - Random ghosts spawning
   - Very good environment and the music suits the game perfectly
   - Good environment

6. Why did you dislike about the game?
   - Battery life of weapon…
   - It was a bit scary but that's what you were going for… The flashlight bursts aren't very noticeable so I wasn't sure I was scaring them.
   - Ghost AI was dumb, have it take cover from fire and stuff, looks like its just going straight for you.
   - Ghost swarming makes game too hard on second floor because there is not enough space to escape.

Since 'Ghost' had a clear outline and objective at the pitching level. Each team member had a clear idea on the goal of the project. This helped to create a clear understanding on the look and feel of the game. There was no clash of the direction of development between team members. However, due to the loss of two team members, some part of the project had to be re-written to

offer a clearer understanding for the replacement coder. A clear object and consistency seems to be needed in a two quarter project. One negative aspect on working in C++ is the lack of debugging abilities. Even though GNU Project Debugger can be used to trace the segmentation faults a different language like C#, could have offered a better debug process.

Conclusion



Foremost problem that our team faced was the sheer size of the project. In the first quarter the problem of not having an UML diagram is not as prominent. However, starting the second quarter it became increasingly hard to navigate through the classes. The lack of UML diagram made debugging difficult. Since three people in our group where Computer Engineering Students, they don't have to take Software Engineering courses, the need of an UML diagram and Software Requirement Specification was minimized. Also the lack of having a standard in documentation hurt the people trying to achieve a common objective to implement. A good foundation documentation is crucial in a two quarter project. Spending a week hashing out a document would have been very helpful.

Seokjune Hong
CSC 491 + 492
6/1/2011

If a solid back story exist in a game, the need for a solid modeler also exist. Our team did not have an expert modeler which hurt us extremely. Modeling is as hard as coding a game; a person need artistic vision, good modelling skills, and understanding of the project. If a dedicated modeler exist, outsourcing all the model need can save great amount of time and pain. However, the modeler needs to have dedication to help both quarters and not quit after the first quarter.

One unique factor in our team was the lost of teammates. Losing two solid programmers, Bill McVicker and Nick Feeney, was a huge blow. The problem extrapolated when documentation was insufficient for the another person to work on the code. Teams need to make sure that teammates can spend two quarters on this game. Team fall out slows down the project, make inconsistent programming, and wreck havoc in moral.

Seokjune Hong
CSC 491 + 492
6/1/2011

Future Work

Adding more floors to the game and additional ghost types would be the greatest addition for the game. However, the greatest need for this game is a better documentation. The lack of software development cycle, software requirement specification, and an UML diagram is making the game harder to manage. Therefore, any future person to work on this game would have to refine the code by adding a clearer picture of how the classes interact with each other. One suggestion would be to translate the C++ code to XNA studio. By having a solid SDK like Visual Studio with the functionality of C# could add the clairvoyance this project desperately needs.

Ultimately, if this game should be loaded into an Android and iOS system as a virtual augmentation reality to be used at a Cal Poly's Open House event. A game called Spec Trekking on Android seems to be good example to base upon [4].

Seokjune Hong
CSC 491 + 492
6/1/2011

References

[1] Jens Erik Vaaler. (2009, Oct 06). The History of Adventure Games (1st Ed.) [The Gaming Vautl]. Available: http://www.thegamingvault.com/2009/10/the-history-of-adventure-games-part-1-the-rise/

[2] Aaron Farley. The History of Adventure Games [eHow] Available: http://www.ehow.com/facts_5173068_history-adventure-games.html

[3] Unicorn. (2001). What's happening to adventure games? [Moby Games] Available: http://www.mobygames.com/featured_article/feature,13/

[4] Spec Trekking [Android Game]. Available: http://www.spectrekking.com/

[5] Flashlight (video game thing) [Giant Bomb]. Available: http://www.giantbomb.com/flashlight/93-156/

[6] Fragile Dreams: Farewell to the Moon.[Giant Bomb]. Available: http://www.giantbomb.com/fragile-dreams-farewell-ruins-of-the-moon/61-21406/

[7] Silent Hill: Shattered Memories. [Giant Bomb]. Available: http://www.giantbomb.com/silent-hill-shattered-memories/61-25707/