Hydromancy Post-Mortem

Josh Dawidowicz - Jesse Errico - Jake Frank - Michael Feldman - Micah Lapping-Carr

The Original Idea

Yesterday, All My Troubles Seemed So Far Away

"I was thinking a game with water."
"Well, what's innovative about that?"
"Computational fluid dynamics and realistic rendering."
"Is that even possible?"

That was our first conversation. We were incredibly psyched and even more idealistic and ambitious. Hydromancy was envisioned as a combination of The Incredible Machine, The Sims, PipeDream (available originally for the Amiga), and a bunch of Rube Goldberg machines. The underlying technology was envisioned as a revolution in fluid dynamics for games, making full use of both particle physics and computation to provide realistic looking and feeling water. We had seen demos of Gish, the NVIDIA gas box, and HydroEngine from Blade Interactive, so it seemed feasible.

Hydromancy was supposed to be very simple. The objective was to get water from one place to another using a series of tubes and pipes and drains and other miscellaneous devices (e.g. water fountains, distillers, toilets). The water would have started in a bowl or from a dripping pipe and would have had to be collected and moved using whatever tools were available. The tools could have been positioned anywhere in 3D space using Wiimote input for the most intuitive interface possible. The GUI would have been almost like a level-editor with a final "Simulate" option to start the flow of water. As in The Incredible Machine, some tools would have already been in position and immobile, while others would have begun out of the way and would have required correct placement. The game's intended audience consisted of puzzle enthusiasts, physicists, and casual gamers. We wanted the interface to be easy enough that someone could pick up a Wiimote and learn.

We had intended to use a number of external libraries for the development of Hydromancy. Our graphics library was Irrlicht, as we all had enough experience with it to make it worthwhile. For physics, we started with Newton, which was too frustrating. We then investigated Bullet, which was a more intuitive engine but did not have anything to do with fluids. We settled on AGEIA's PhysX library, which had everything we needed for both rigid body and fluid physics. Had we purchased a Physics Processing Unit, PhysX would have been an incredibly powerful tool for us. Sound was done with FMOD and Wiimote interaction would have been done with Wii Yourself. Fluid dynamics, however, required more work.

The fluid dynamics library would have been based mostly in computational fluid dynamics for realistic water flow, but would have had extra features such as capillary action,

adhesion and changes in state. The computation would have been done according to the Navier-Stokes equations. As luck would have it, we discovered an algorithm by Jos Stam that promised real-time 3D fluid simulation with little adaptation work.

The Team

If you can find them, maybe you can hire…The A-Team

Jesse Errico – Hydromancy was his idea. He began studying computer science in high school, programming in "old school C++" taught to him by the "crack father/son programming team of Kevin and Steven Cedrone." Later he studied databases in the Bioinformatics Department of MIT's Whitehead Institute. When he came to Brown, he was trained in the ways of Java by Andries Van Dam in CS15, and he has since progressed from crafting database tools in CS32 to writing terrible raytracers in CS123 to writing viruses in CS166 and counting Excel squares while building Hydromaze maps. He met Mike through the Brown Band, met Jake through CS123, cannot remember the details of his first meeting with Micah (it's all a blur), and his first interaction with Josh was to make fun of his owning a Dreamcast. His hair is a magical forest.

Josh Dawidowicz – Got his first computer at the age of 3 and has been learning ever since. His first exposure to computer science came in 7th grade when he attempted to write a program in Visual Basic that could compete with those "AOL Toolz" programs. In his senior year of high school, he studied Robotics as an independent study, and wrote Lights Out, by Tiger Electronics, in Java, complete with a linear algebra solver for the game. At Brown, he joined the Cult of Shriram and plied his trade in PLT Scheme. His expertise is in databases, and will be working at Investment Technology Group this summer. He plays poker with the pope, the Incredible Hulk, Leon Trotsky, and Socrates on a regular basis.

Michael C. Feldman – Began his quest to attain all computer science knowledge at the age of 2. In CS15 at Brown, his Tetris implementation was the Tetris to end all Tetres. His CS123 raytracers were so awesome they slowed down the TA code. He got 70 points out of a possible 60 in CS176. He and Chuck Norris once both picked rock in a game of rock-paper-scissors; there were no survivors.

Micah Lapping-Carr – Saw his first computer on his way out of the womb and wrote a program to use the hospital computer as a controller for an Aibo before he took his first breath. Before he ended his high school career, he wrote Minesweeper in Java so he could play video games in school. Upon entering Brown, he met Josh in a Scheme-for-n00bs focus group, and they became coding buddies. He will be working at EA LA this summer as a producer intern (not a code monkey!). His Defense of the Ancients skills helped him get the job.

Jake Frank – Started programming on his TI-83 Plus in 6th grade, and within a few months, had written a Turing-complete dialect of Python using only the trigonometric functions. By 12th grade, he won a Homeworld competition to become certified as an ub3r-l33t g4mez0r, and continues to wear his free t-shirt from that tournament. At Brown, he met Josh and Micah in CS17 and the three have spoken to

each other in Scheme ever since. Two summers ago, he worked at Streambase, a database company started by Michael Stonebraker, Database Overlord. He is Battlestar Galactica.

What Went Right & Wrong

Jos Stam 2D	Jos Stam 3D
SmokeDuel	Hydromachine
Level Design	AGEIA
Team Dynamics	Particle Fluids

AGEIA PhysX

The prospect of using a popular physics engine was very enticing. We figured that if PhysX was so universally popular that it would be both intuitive and well-documented. Sadly, we were mistaken. The AGEIA setup was not very difficult, as there were numerous online tutorials pertaining to setting up a PhysX scene and integrating it with Irrlicht. However, anything beyond the setup required extensive library hacking and documentation investigation. Picking the correct search term and eventually the correct help file entry was a challenge even before it came to then using the proper method.

Once we found the proper way to, for instance, create a joint between two actors or to apply torque to an object, we still had to contend with the fact that most PhysX features were incredibly unintuitive. First, most of the functions that would have been useful had not been implemented yet. Other functions and features that we used were the opposite of unintuitive or the documentation for them was incorrect. For example, the rotation about each axis was labeled incorrectly in the documentation, so we ended up locking rotation in the wrong directions and limiting the angle in others. The most frustrating problem of all was how AGEIA scales objects. Rather than measuring the actual dimensions, AGEIA made use of a radius in each direction from the center point.

Jos Stam 2D

Researcher and CS professional Jos Stam wrote a paper, freely available online, entitled “Real-Time Fluid Dynamics for Games,” which described an algorithm that, while mathematically intensive, promised to accurately simulate a fluid using the Navier-Stokes equations in real-time. We couldn’t resist—as indicated in the paper’s title, this is completely ideal for use in a games setting—and we quickly adapted the algorithm for use in 2 dimensions, using a simple visualizer, built with GLUT, to display the results. These results showed that the algorithm was very good at simulating smoke; they were visually pleasing and apparently scientifically accurate. It would be an easy matter to integrate the algorithm into a simple game.

Jos Stam 3D

The footnotes of Jos Stam’s paper promised that converting the algorithm to 3 dimensions would be trivial, and indeed this turned out to be mostly true—some stubborn errors turned out to be simple mistakes involving faulty C macros. However, in the conversion from 2 dimensions to 3, the utility of the algorithm in a games setting was lost. The 2-dimensional code represented the fluid as a series of

tiny squares, each of whose color was determined by the density of the fluid in that particular area, which worked well enough, but the corresponding cubes used by the three-dimensional version looked blocky and unnatural. Increasing the resolution to make the cubes smaller might theoretically have solved this problem, but the algorithm got so much more computationally intensive with the addition of the third dimension that running it on an average computer with a passable resolution was not feasible at more than 2 or so frames per second. It also became apparent at this juncture that while Jos Stam's fluid-dynamics solver was very good for simulating smoke, simulating water was a different matter entirely. There was no maximum value for the density of fluid that could occupy a single cube, which meant that any amount of fluid could be compressed into a single square if a marginal amount of force were applied to put it there. This means that any water simulated by this algorithm would be comprised of a series of cubes of varying color, with no natural wave motion. Obviously, our original idea was not going to work.

Hydromachine

Hydromachine was our implementation of NxCooking, which is how AGEIA "cooks" mesh files into actual objects. First, there is very little documentation regarding cooking, which follows a common theme in AGEIA. The documentation that did exist was sparse at best, as there were numerous ways to cook a mesh. We could have cooked straight to file and read the cooked file back, or we could have stored the mesh in memory. Secondly, once we had written the cooking code, a lot of supposedly pre-defined functions were missing, such as ways to read streams from the file. While this code was not difficult to write, the fact that it was missing made it very frustrating. Finally, we never were able to cook a mesh. Hydromachine crashed numerous times in the same location, which led us to believe it was an out-of-memory error, but due to poor library documentation, there was no definitive answer. It was at this point that we abandoned our original vision of Hydromancy for Hydromaze.

SmokeDuel

With approximately three weeks left, we needed to come up with a game that used the aforementioned 2D solver that was simple, modular, and fun. Mike proposed an asteroids-style game that made use of the fluid dynamics code, and it quickly turned into SmokeDuel. This game required no changes in the 2D code, though a clean wrapper in C++ was written to make the solver easy to interface with. Because the game was so simple – inasmuch as it is 2D, required minimal gamestate tracking, and is similar to SpaceWar and Asteroids – we were able to put together a demo extremely quickly. Additionally, because of our initial design, we were able to iterate quickly; adding new features such as a splash screen, sound, additional weapons, color selection support, shield indicators, etc. was not difficult.

Particle Fluids & Joints

Seeing as Jos Stam's code did not apply to water, we resorted to AGEIA's particle system for fluid dynamics. However, there was very little computation involved based on actual fluid dynamics, and the system looked like a lot of very small boxes interacting; it was the equivalent to watching sand instead of water. The examples of water from AGEIA did not properly represent water, so we did some

significant tweaking of the fluid descriptor. Eventually, the water behaved close to what real water looked like.

Creating the physical maze also took a significant amount of time after debugging. The AGEIA documentation suggested that the best way to connect two actors is to use a joint. Using a joint worked very well to join the maze with the universe to create a 6DOF joint, but connecting the walls required more than a fixed joint. With enough applied force, the joints would shift a bit and some of the water would leak out. Instead, we had to retool the entire CreateMaze process to create one solid actor instead of a number of jointed actors.

Hydromaze Level Design

Making a serious effort at Hydromaze was now a possibility, but the final product wouldn't be particularly fun unless we had a decent maze. The project of level design was not one that could be made easier with any amount of fancy programming or mathematics. In the end, a low-tech method was selected—a game of Dots and Boxes. (see Wikipedia) If one cuts off a game of Dots and Boxes halfway through, right before someone scores for the first time, the result is (usually) a passable maze. With basic design handled this way, and with the exact spatial mathematics worked out in (surprisingly) Microsoft Excel, building an interesting and complex maze in Irrlicht and PhysX was a relatively simple affair. Both of the Hydromazes were designed this way.

Team Dynamics

We entered this project as friends, which can often result in either no work being done or a set of broken friendships. Neither of the aforementioned problems occurred. Since the start, there was a tacit split in the work and everyone gravitated toward the part of the project that suited them best. Micah and Jesse worked on Jos Stam's code while Jake, Mike and Josh worked on cooking meshes and dealing with AGEIA's water. Once we reached the turning point of realizing Stam's 3D code and Hydromancy wouldn't work, there was a new silent split in the workload. From that point on, Mike and Jake instantly began work on SmokeDuel and Micah, Jesse, and Josh started designing Hydromaze and investigating the parts of AGEIA that had previously gone untouched during the initial phase of the project.

In Conclusion…

Creating Water Is Harder Than We Thought

Despite our eager and optimistic first intentions, things took a turn for the worse and we had to change course after floundering in AGEIA like a dolphin on a sand dune. We learned a lot about AGEIA, Irrlicht, and FMOD along the way. However, because of the time investment involved in learning about these libraries, we were forced to give up on the Wiimote plan rather quickly. In addition, our original project got scrapped due to its large scale and the unsuitability of Jos Stam's code for simulating water.

Hydromancy Post-Mortem

Josh Dawidowicz - Jesse Errico - Jake Frank - Michael Feldman - Micah Lapping-Carr

In the end, though, we were able to come up with two small yet very fun games. All in all, a job well done. *wipe hands together three times, then walk away*

Hydromancy	
Principals	Josh Dawidowicz, Jesse Errico, Jake Frank, Michael Feldman, Micah Lapping-Carr
Budget	$50
Development Time	3 months
Release Date	5/9/08
Platform	Windows/Linux
Hardware	SmokeDuel: Pentium 4, 256 MB RAM, non-integrated video card Hydromaze: Pentium, 512 MB RAM, GeForce 6150
Software Used	Development: Microsoft Visual C++, GCC, GDB Libraries: Irrlicht, Ageia PhysX, FMOD Art: Adobe Photoshop
Notable Technology	Jos Stam's Real-Time Fluid Dynamics Ageia PhysX
Size of the Project	Approx. 3000 lines of code