Bifröst Physics

Final Report

Arman J. Frasier
Advisor: Dr. David Heddle

Christopher Newport University
Department of Physics, Computer Science, and Engineering

The Problem

Demonstrations and examples are a key part in most introductory physics student's learning process. Students are well versed in how gravity functions because they have experienced its effects first hand, and do not have trouble visualizing objects falling off of cliffs or being fired out of cannons. However, when it comes to things like electromagnetism, visualization becomes a great deal harder. In fact, some students have trouble visualizing cosmological gravitational interaction, such as the path traced by a moon as its parent planet orbits a star. The replication of events can also become a problem, even for a professor. No matter how focused and practiced the instructor, it is highly unlikely one could throw a ball the exact same way two times in a row. As well, students who miss a class miss out on these vital demonstrations.

Almost twice as important as demonstrations are questions. When students ask questions about physical concepts, they enhance their learning experience not only by clarifying an issue, but also by opening themselves up to asking more questions. Despite the importance of inquiry, most students are too timid or ashamed to ask question during class. Many students completely avoid their professors outside of class, preferring to work by themselves or amongst other students to understand an issue which could have been resolved in mere moments by a simple question during class or in the professor's office.

The Solution

My attempt at a solution to these issues is the creation of a 2D physics simulator and tele-teaching utility. The objective of this tool is the accurate simulation two dimensional kinematics and electromagnetism, with a focus on experiment reproducibility and the ability to synchronize experiments among networked clients. This program would then be used for the distribution of experiments to students, allowing them to alter the experiment to get "hand on" experience with subjects like repulsion of like charges, and how charged particles move through electric fields. As well, if a student is having problems, they could link their clients to another student or professor over the internet to receive tutelage on the subject with which they are struggling.

The development process for Bifröst Physics is detailed on the following pages, in chronological order, to guide you through the evolution of the inner-workings of the program.

Developing the Solution – Preliminary Design Choices

The first choice that had to be made for the development of this program was that of the language in which to program. For this choice, I selected Java, which offered more beneficial traits than detrimental traits. The major benefit was platform independence, as students are using an increasingly varied field of operating systems. By using Java, my program would be useable by any student regardless of operating system.

Next, a layout for the program was devised containing three major areas: a menu bar, an environment canvas, and a chat display. The menu bar was to contain access to all pertinent dialogs including the entity editor and environment editor. As well, the menu bar was to contain any entity constructors and a playback control mechanism. The environment canvas would be where the simulation was drawn on the screen and where the user would interact with the environment for various actions like moving entities. Finally, the chat display would facilitate telecommunications for networked sessions.

Initially, the entity class was designed to be extendable to allow the development of more complicated entities, but it was later decided that the only entity that would be developed for the sake of time would be that of the spherical ball class.

Iteration One

The first iteration of the program was incredibly simplistic, having only implemented the generation of random entities and gravitational fields. This iteration was used mainly to test the conservation of energy in the system. Entities had simple options, such as being collidable, pinned, or traceable – placing a dot every ten milliseconds at the current location of the entity. The collision code was incorrect in this version, leading to loss of conservation of energy during entity collision; however, not only was conservation lost during collisions, but a stand-alone frictionless entity bouncing around would eventually lose all of its energy. After much experimentation with Java's floating-point arithmetic accuracy, it came to light that the way the physics was calculated was being done in a way which would not allow for accurate modeling. At the time, the physics was calculated after the entities were drawn, thus limiting the number of physics calculations to the speed of the rendering of the environment. To resolve this issue, the program was rebuilt from the ground up into iteration two.

Iteration Two

Iteration two brought the decoupling of the renderer and physics simulator. To resolve the energy loss of the prior iteration, the physics calculations were done in their own thread, separate from the rendering of entities. This physics simulator uses two time step values: a constant time step of one millisecond (known as the simulator step), and a user-specified time step (known as the physics time step). This system would do the physics calculations as fast as possible, and then sleep until another class awoke it. This other class is known as the taskmaster, and simply wakes the simulator every one millisecond. This prevents the simulator from simulating too fast on powerful machines. The simulator then uses the user specified time step in the physics calculations. The smaller the user specified time step, the more accurate the simulation. To run the simulation in "real time" mode, a user would specify a physics time step of one millisecond.

In addition to this decoupling of the physics and rendering, iteration two brought the completion of the physics calculation algorithm by the inclusion of the following other modules: electric fields, mutual electric forces, mutual gravitational attraction, fluidic resistance. As well, the method of tracing an entity was changes from placing dots to drawing a line along that path. Additional generators were developed: scaled random, square lattice, column, and row.

Iteration two also brought the start and completion of network development. Networking is done through a payload system. The server accepts any number of connections from clients, spawning two threads for each of them (one for incoming data, and one for outgoing data). Clients never communicate directly with other clients; instead any message from a client goes through the server and is from there disseminated to the cloud of clients. Physics data is pushed to clients before execution of the simulation, that is, the only transmitted physics data are the initial conditions. This reduces network overhead and also allows clients to maintain a working copy of the simulation upon disconnection from the networking session.

Even with all the improvements in iteration two, problems still existed when one started to use more than fifty entities on the screen at one time. This issue was solved in iteration three.

Iteration Three

Iteration three solved the issue of many-entity simulation quality by multithreading the simulation process. The first attempt at multithreading the process created a new thread for each entity, and then waited to join all the threads before continuing processing. This introduced an extremely high overhead due to the constant creation and joining of threads. The next attempt was to create only as many threads as there were logical processors on a machine, and dividing entities up evenly among all of these threads, and then processing and joining the threads. While this lead to some improvements, the constant creation and joining of threads still introduced an overhead which negatively impacted performance. The final solution was to develop a custom concurrency management scheme, and to simply create a thread which was reusable by calling Java's interrupt() function on that thread. By doing this, the program no longer needed to recreate threads after their initial creation and could instead reuse the currently allocated threads.

Future Development

Future development would likely include the creation of more entity types such as boxes and other rigid bodies, as well as an entity class which functions to enhance performance when trying to simulate fluids. Implementation of axles and springs would allow for experimentation with more complex entities such as pendulums and the simulation of planar molecules.

Conclusion

Bifröst Physics' development cycle was extremely interesting and quite a learning experience. It forced me to do many things that are common in real-world programming and not so much in the classroom: creating my own multithreading concurrency management process; optimizing, and reoptimizing; multi-client sustainable network programming; and extensive physics simulation. Throughout the time spent on development, I drew on my skills in not only my major field of Physics, but also my computer science knowledge from in and out of class as well as my minor field of Mathematics. Overall, I feel that Bifröst Physics could help students learn physics by demonstration in the classroom and over great distances through the power of the internet.