Physical Systems Simulator

CS 190 Project Requirements

George Quievryn (gquievry)

February 2, 2000

Overview

Humans are visual creatures by nature. In acedemia, however, the jump from theory to visualization is often a difficult on. This is particulary true in the mathematics-based fields of physics, engineering, and dynamic systems analysis. Most students, educators, and researchers have no problem visualizing what will happen if you place a positive charge next to an electron. However, what happens if you introduce many variably-charged particles and take into account gravity and quantum-mechanical forces? Many students can handle the mathematics of these systems, yet fail to gain a strong intuitive feel for the problem that comes only through visualization. Researchers can postulate a mathematical model to fit these systems, yet how can we be sure that the description is accurate without visualizing the system?

I would like to propose an answer to these visualization problems. This program would present a blank canvas (the “universe”) to the user similar to conventional drawing programs. The user would then have the ability to add items to this universe and specify properties for each object. These properties can dictate how an object effects other objects, and how this object is affected by other objects. The application would then animate these objects to simulate physical events.

Examples of object properties include mass, charge, quantum spin, price, altitude, or any other variable that the user can invision. The great power of this application is its versitility. Systems can represent sets of charged particles, a ball rolling down an incline, a spring, an atom or molecule, and even dynamical systems such as the stock market or a weather system. The user simply needs to specify properties and how to apply these properties, and the application will simulate the outcome.

This program is extremely useful in predicting the accuracy of mathematical models, as an educational tool, and even in forecasting future events. The project is clearly very scalable and divisible. In addition, the software developers do not need

to know or understand anything about physics, weather systems, or the global economy. The only thing the developer needs to know is how to apply a given set of rules to a system.

Functional Requirements

The following is a list of functional requirements for this project. The number in perenthesis represents the priority of the requirement, with the lower number corresponding to higher priorities.

- Allow the addition of objects to the universe (1)
- Allow the user to specify properties for each object (1)
- Allow the user to specify formulas for object behavior (1)
- Animate the system according to user specs (1)
- Automatically store object position (1)
- A reasonable intuitive GUI that can be used by non-CS experts (1)
- Allow user to specify animation speed (2)
- Allow user to specify time resolution (3)
- Allow the user to save/load systems (4)
- Allow user to track individual objects (5)
- Turn on/off visual grid (5)
- Allow the user to save a library of sample objects (6)
- Have library of constants (6)
- Allow users to alternate point of view (7)
- Allow users to zoom in/out (7)
- Allow for more complex objects, such as connectors and springs (8)

System Requirements

The application must be able to run on the local machines available in the Sun Lab. The system complexity increases exponentially with the number of objects added, and it is difficult to predict how this might effect performance. As a conservative criteria, the system should smoothly animate a minimum of 10 objects.

Testing and Evaluation

The application should be completely stable and free of crashes and memory leaks. Since the target audience includes students and educators, the project should be tested by several members of the Brown student community from departments other then the Computer Science department. Ideally, the program should be intuitive enough to be useful to high school students and teachers. The number of objects that the system can handle while allowing smooth animation should be tested and documented - this number should be greater than 10. The number one priority should be accuracy (i.e. the program should perform the proper animation given the user-defined rules). To test this, basic systems of proven outcomes should be tested (i.e. 2 charged particles, a falling ball, and a orbiting planet).