

A Movie is Worth More Than a Million Data Points

Hans-Peter Bischof and Jonathan Coles

Rochester Institute of Technology
102 Lomb Memorial Dr., Rochester, NY 14623
{hpb,jpc1870}@cs.rit.edu

Abstract. This paper describes a distributed visualization system called Spiegel, which displays and analyzes the results of N -body simulations. The result of a N -body simulation is an enormous amount of data consisting of information about how the bodies interact with each other over time. The analysis of this data is difficult because it is not always clear ahead of time what is important. The visualization system allows a user to explore the simulation by moving through time and space in a 3-dimensional environment. Because of its flexible architecture, the visualization system can be easily extended to add new features.

1 Introduction

The N -body problem is the problem of calculating gravitational force vectors between N particles over a given time period. Since each particle affects all other particles the complexity of this problem is $O(2^N)$. This becomes challenging to compute for astrophysicists who want to build models of entire galaxies where the number of particles is often in the millions.

To overcome this exponential growth problem, researchers in Tokyo, Japan developed specialized hardware called GRAPEs (GRAvity PipELines) [8]. A GRAPE board is solely designed to calculate force vectors on a given set of particles using a highly parallel architecture. Recently, newer models have been developed which are smaller and can easily be installed in computer clusters [2].

The Department of Physics at the Rochester Institute of Technology (RIT) has a 1 teraflop, 8 dual-Xenon node GRAPE cluster that can run simulations up to one million particles. The result of a typical run of a simulation is a file on the order of 20 Gigabytes. These files contain the information of how the particles interacted with each other over time. The difficulty is to interpret this vast amount of information.

One of the best ways to analyze this information is through a visualization system. Such a system, named Spiegel, has been developed as a joint project between the Department of Physics and the Department of Computer Science at RIT. The system projects the simulation data into a 3-dimensional world and allows the user to explore the world by moving through space and time. The basic functionality for the Spiegel system is to:

- Provide multiple viewpoints by having virtual cameras positioned in the simulation universe. The particles are represented as points colored based on physical properties.
- Allow a camera position and angle to be easily adjusted by using standard GUI widgets, the mouse, joystick, or other specialized input hardware such as Flock of Birds [3].
- Allow the current time index to be adjusted. The software should be able to create a movie of the simulation. Sample movies can be found on the project website [4].
- Provide a scripting language to control the system and allow new features to be added on easily.
- Allow collaborators across the Internet to connect together to view and manipulate the same simulation.

2 System Architecture

In designing Spiegel, one primary goal was flexibility. The system should be able to work on one machine or across many machines and allow new functionality, such as interface extensions or visualization techniques, without having to modify any of the code base.

The general system design consists of three major components: the Switchboard, the Feeder, and the ViewController. Each of these components talks to the others using a simple scripting language called Sprache. This language controls the creation, movement, and positioning of the cameras, and includes commands for loading particle information and changing the virtual time frame that is being viewed. Using this language, complex sequences of commands can be written without having to modify the system. These scripts can be saved and loaded on demand to be played repeatedly. An simple example is given in Figure 1.

```
camera create Camera0 # create a camera view labelled Camera0
camera 0 moveto 2 3 4 # move it to x=2 y=3 z=4
camera 0 rotate 4 3 2 # rotate it along the x,y,z axes by 4,3,2 degrees
updateview # commit the changes to the view
```

Fig. 1. Simple example of the Sprache scripting language

The Switchboard is the module responsible for maintaining the current state. All commands that affect the position of the cameras, or anything else in the system are first sent to the Switchboard. When commands are sent to the Switchboard, the Switchboard distributes them to all of the different displays. There is no limit to the number of displays that can be connected. Before commands can be sent to the Switchboard, however, they must first be sent to the Feeder.

The Feeder acts as a central point to which other pieces of the system send commands. The commands are then ordered and sent to the Switchboard one at

a time. The Feeder also allows an additional layer of interpretation. If a different, more complex, language were to be developed, the Feeder could provide a translation mechanism to convert the higher level language into Sprache commands. The rest of the system would be unaffected, but complex sequences could more easily be programmed, either directly by the user, or through an interface of some sort.

The ViewController is the visual part of the system that is connected to the Switchboard and it receives instructions from the Switchboard on how to update the display. Within the ViewController is the MapView and the CameraViews. The MapView offers a global view of the positions of the cameras relative to each other and the coordinate system of the virtual space. The CameraViews mechanism manages the different CameraViews. Each CameraView shows the particles as they appear from the point of view of one camera. The two pieces of the ViewController also act as an interactive interface. The user can manipulate the cameras by directly clicking and dragging the cameras in the map view, or by clicking and dragging in one of the CameraView displays.

3 Component Interaction

The graphical user interface provided by the ViewController controls the main functionality of the system. Interaction with the UI generates a sequence of Sprache commands that are sent to the Feeder and then to the Switchboard. Once the Switchboard has sent the actual movement commands to the ViewController, the ViewController updates its display with the current state of the system. It is important to understand this sequence of events. The ViewController will not act on a user command unless instructed to do so by the Switchboard. This is to prevent the state in the Switchboard from became unstable.

The three components communicate using one of two mechanisms. Either they can talk over an TCP/IP socket connection, or, if they are running on the same machine, they can be connected directly in memory. The advantage of communicating in memory is one of speed. However, allowing for network communication yields two powerful features. First, any language that supports a network library can be used to design any portion of the system. If one ViewController would be better written in C++ or Python then it would be easy to plug in to the system. The code would simply have to understand how to talk to the Feeder and Switchboard using Sprache commands over a TCP/IP connection.

If there is a new input device that is better suited to manipulate three dimensional images, such as a glove, then the driver would only have to support talking to the Feeder. This allows for an incredible degree of freedom for anyone wishing to add to the system. Second, it means that the ViewControllers, for example, can be spread across the Internet, allowing researchers to view, discuss, and manipulate the data simultaneously. In order for researchers to do this, however, they must have access to the same data.

There is currently only a tentative design which allows the ViewController to retrieve particle data by talking to a module called the ParticleView. The ParticleView would have direct access to the data and would allow the ViewController to query for particle information at a given point in time. The data is stored in a format which gives initial values and then incremental changes over time. The time intervals for each change may differ between particles, but using some simple algorithms it is possible to extrapolate the position, velocity, etc. of every particle at a given time. The ParticleView performs the extrapolation at the request of a ViewController. A picture of the design is in Figure 2.

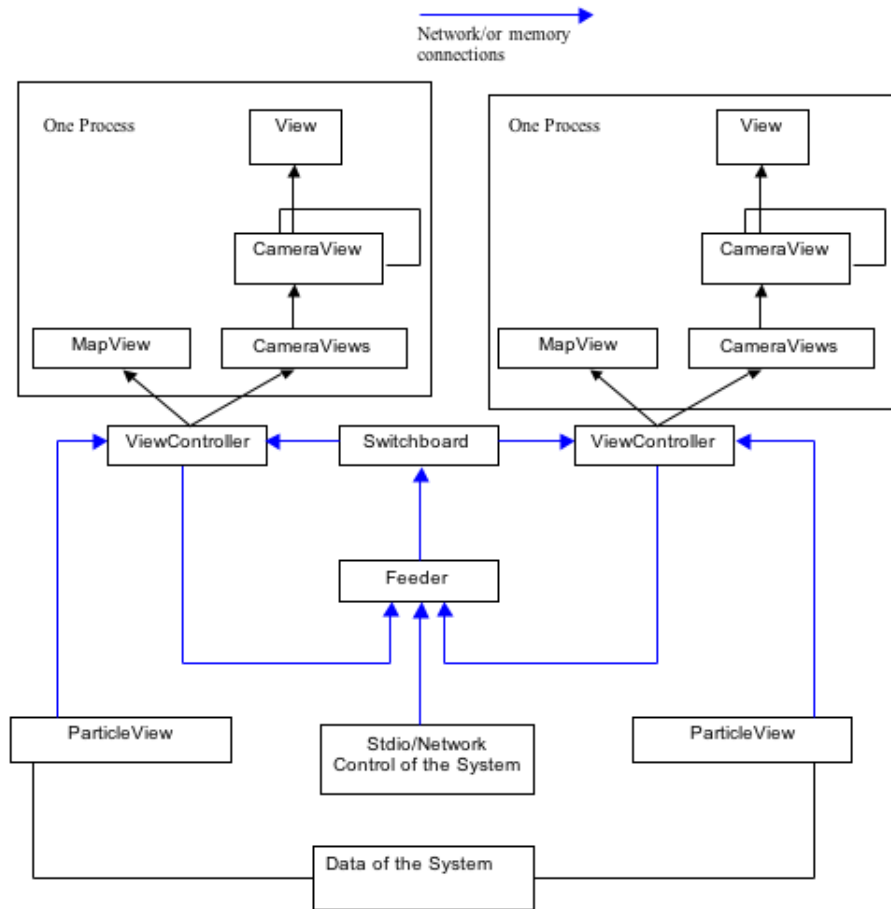


Fig. 2. System design for Spiegel

4 Source Data

The simulation is executed on a 1 teraflop GRAPE cluster system that creates a file containing the state of particles over time. This file is typically very large and in a plain text format. Before it can be used by the visualization system it must be converted into a denser binary format that is more efficient and can be loaded quickly.

The plain text file is divided into three parts. The first part of the file contains three lines holding header information:

- Number of particles used for this specific simulation n
- Time units for this simulation t_{\max}
- Number of black holes involved in this simulation nBH

Each of the remaining lines represents the state of a particle, i , at a given moment in time, t . A line has the following format, for $1 \leq i \leq n$:

$$i \ t_i \ x_i \ y_i \ z_i \ v_{x_i} \ v_{y_i} \ v_{z_i} \ a_{x_i} \ a_{y_i} \ a_{z_i} \ \left[\frac{da_x}{dt} \right]_i \ \left[\frac{da_y}{dt} \right]_i \ \left[\frac{da_z}{dt} \right]_i \ \left[\frac{d^2a_x}{dt^2} \right]_i \ \left[\frac{d^2a_y}{dt^2} \right]_i \ \left[\frac{d^2a_z}{dt^2} \right]_i .$$

Initially, all particles are listed with the state at time $t = 0$. The first nBH lines are the black holes and the remaining $n - nBH$ lines are the other particles. After the initial conditions, particle information is present for some $0 < t \leq t_{\max}$ only when that particle information has changed. Some particles are noted more often than others, because the energy level for these particles is extremely high. If the exact position is not known at time t , the position of the particle is interpolated. The following property is guaranteed: t_k on line k is t_l on line l if $k < l$. Pseudo-code for displaying the particles is given in Figure 3.

```
while (t < tmax) do
  collectionOfStars = findStarsInTime(t)
  collectionOfStars.display()
  t = t + deltaT
```

Fig. 3. Pseudo-code for extracting, interpolating, and displaying particles

The original interpolation predicted where a particle will be in the future based on state of the particle in the past. Given the number of time units in the future to predicate, Δt , and the first and second derivatives of acceleration from the input file, the equation for the future x coordinate is:

$$x_{\text{future}} = x + v_x \Delta t + \left(\frac{\Delta t^2}{2} \cdot \frac{da_x}{dt} \right) + \left(\frac{\Delta t^3}{6} \cdot \frac{da_x}{dt} \right) + \left(\frac{\Delta t^4}{24} \cdot \frac{d^2a_x}{dt^2} \right) .$$

Similar equations calculate y_{future} and z_{future} .

This worked well for stable galaxies, but for more interesting galaxies that are more active, the calculations became too imprecise. The solution for this problem is to find a particle's state s_1 and s_2 where $t_{s_1} \leq t < t_{s_2}$ and use linear interpolation based on time. This also decreased the amount of information stored for each particle because only the position information is important.

The original design did not taken into account the complexity involved in processing the particle information. Disk I/O speed was the major problem. More than 24 frames per second (FPS) are required in order to get a smooth visualization. The initial version could only read enough data to achieve 1 FPS. Preprocessing, removing non-essential data, and compressing the file increased the speed by 500%. The system can now display at 5 FPS.

5 The System in Use

The system was successfully used to visualize the collision of two galaxies, which resulted in the creation of a new, single galaxy. The astrophysicists created an initial situation where they assumed one galaxy would be stable. The visualization system proved that this assumption was not correct. Figure 4 shows the system at three different points in time. The black holes are the three dark masses at the center of the three galaxies. The stars are single points. The picture on the left shows the initial situation. Over time, the upper galaxy becomes unstable and eventually captured by the center galaxy.

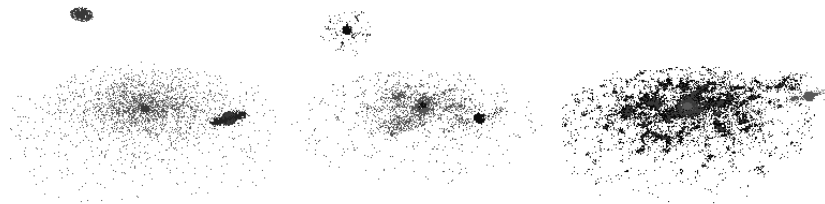


Fig. 4. Three snapshots of three galaxies merging.

6 Creating a Movie

It is relatively simple to create a movie. First, a system must be defined that controls the transition through space and time. This systems output are simple Spiegel commands, which are sent to the Feeder.

The rendering is done off-screen and each individual image is stored in a file. The individual images are later assembled into a movie using the Java Media Framewor

7 Related Work

Spiegel was designed with minimum hardware in mind—no special hardware or software is required to run the system. Hut [7], followed a different approach by using the planetarium of the American Museum for Natural History (AMNH) in New York City as a display device. The user gets a very realistic visual experience. This system is based on partiview [1], which uses OpenGL [9] to render images. This allows the program to interact with the museums projection system, an Onyx2 with 28 CPUs, 14GB of memory, 2TB disk space, and 7 graphics pipes. The user interaction with partiview requires some practice; it does not allow interpolation of the position of stars over time. Partiview is very well suited for display existing galaxies and not very well suited to visualize the results of simulations.

Most visualization systems, like [6] provide a generic framework for information specialization which is mainly focused on dynamic hierarchy computation and user controlled refinement of those hierarchies for preprocessing unstructured information space. Spiegel has a very structured information space and would not benefit, and therefore does not need, the immense overhead required by the framework described in [6].

8 Future Work

Spiegel is work in progress. The decision to use the scripting language Sprache turned out to be a great idea. However, Sprache is a small, simple language which does not include any kind of control structure. The system would benefit if the language would be extensible and act like a macro processor interpreter [5]. This would allow new, often used, commands to be added without changing the language or the systems that depend on it.

The Feeder is designed to listen on a network port for connections from programs that supply Sprache commands. With this, the system can be easily extended to accomplish tasks which have not been programmed into the system. Smarter, better suited tools can be programmed and attached to Spiegel in order to perform more complicated tasks, like a specialized fly-through path that follows a particle, or connection and disconnection of cameras in a sophisticated time driven fashion. Some of these ideas are planned for future development.

The visualization part would benefit from the study of perception. It would be beneficial to choose colors, fog effects, and other perceptual characteristics in order to give a better visual presentation of the physical reality. No human has seen the collision of galaxies in a short period of time. Therefore, what can be visualized is not based on experience; it must be based on perception.

Since the system is extensible it could visualize gas clusters if a plug-in were available. These parts have to be developed in order to prove that the framework is flexible enough.

The current system can render an animation sequence at 5 FPS. More than 24 FPS is required in order to see a flicker-free movie. The major performance

bottleneck of the current system is file I/O. Around 80% of the CPU cycles are used to read the data from disk, around 5% are used for data interpolation and graphic rendering. At the moment, the file is a compressed. Clearly, the goal is to decrease the amount of data that describes the system. Each particle follows a 3-dimensional spline curve. Therefore, it should be possible to describe paths of the particles as a function of time and not with discrete points. More research needs to be done to answer this question.

Acknowledgements

We would like to thank the following Computer Science Students at RIT for their contribution: Andrew Bak, Meng Jiang, Raun Krisch, Andrew Rader, Prachi Shinde, Christopher Stelma, and Peter Weisberg. We would also like to express our thanks to David Merritt, Andras Szell, and Peter Berczik from the Department of Physics at RIT. They provided the actual data and the interpretation of the visual representation.

References

1. Brian Abbott. partiview. <http://www.haydenplanetarium.org/hp/vo/du/partiview.html>.
2. Ernst Nils Dorband, Marc Hemsendorf, and David Merritt. Systolic and hyper-systolic algorithms for the gravitational N -body problem, with an application to Brownian motion. *J. Comput. Phys.*, 185(2):484–511, 2003.
3. Flock of birds. <http://www.ascension-tech.com/products/flockofbirds.php>.
4. The grapecluster project: A dedicated parallel platform for astrophysical dynamics. <http://www.cs.rit.edu/~grapecluster>.
5. Brian W. Kernighan and Dennis M. Ritchie. The m4 macroprocessor. 1977.
6. Matthias Kreuseler, Norma Lpez, and Heidrun Schumann. A scalable framework for information visualization. *Electronic Edition (IEEE Computer Society DL)*, 27.
7. Peter Teuben, Piet Hut, Stuart Levy, Jun Makino, Steve McMillan, Simon Portegies Zwart, Mike Shara, and Carter Emmart. Immersive 4-d interactive visualization of large-scale simulation. volume 238, page 499. Astronomical Society of the Pacific, 2001.
8. The grape project. <http://grape.astron.s.u-tokyo.ac.jp/grape/>.
9. Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.1*. Addison-Wesley.