Space System Modeling: Saturnian Moons

By Justin Winkler

Abstract

The Saturnian moon system is home to many fascinating and unusual astronomical phenomena. For example, Epimetheus and Janus share orbits and exchange momentum every four years. Hyperion has chaotic rotation. Our understanding of these phenomena, however, is unfortunately limited. This project hopes to add to our understanding of space systems by providing a comprehensive simulation of the Saturnian moon system. By doing this, this project attempts to expose what phenomena can't be explained with modern models and perhaps suggest theories to explain the unexplained.

Introduction

This project focuses on the modeling of complex space systems. A problem with the realm of modeling is that there are nearly always discrepancies in our explanations of certain phenomena. The purpose of this project is to create a simulation of the Saturnian moon system in hopes of better understanding unexplained occurrences within the system. This project therefore aims to reveal phenomena which current models do not explain, and possibly offer explanations of such phenomena.

The scope of this project is limited only by time and computer resources. By adding more parameters and factors to create more complex and accurate models, simulations could be improved with no foreseeable end. Unfortunately, time is limited and the calculations necessary for such a simulation may eventually exceed the computational resources of the lab after enough model alterations. Nonetheless, given the current resources, this project is still able to create a comprehensive model.

Background

The Saturnian moon system is a hotbed of interesting phenomena. There are moons that have odd orbital inclinations, there are moons that are unusually colored, and some moons may contribute to the regulation of Saturn's rings. One moon is effected by the forces within the system to such a degree that it's rotation is chaotic. There are two moons that share an orbit, with the appearance that one will overtake the other and the two will collide. This does not occur, however, as every four years they exchange momentum, making the slower moon faster than the originally faster moon. Nowhere else in the solar system do phenomenas such these occur in such abundance. This makes the Saturnian moon system a natural choice for simulating.

Numerous solar system simulators exist today. A simple example, named Orrery, can be found at http://orrery.unstable.cjb.net/. Other simulations have been made concerning the N-Body problem, which attempt to find subsequent motions of bodies based on initial parameters. One of these simulations, which uses NetLogo, is found at http://ccl.northwestern.edu/netlogo/models/N-Bodies. This project will build upon these past models by applying some of their techniques to the Saturnian moon system.

One major technique used to model space systems is Newton's Law of Universal Gravitation. This is a basic law used in innumerable simulations. Newton's Law of Universal Gravitation is as follows:

$$F = (G * m_1 * m_2) / (r^2)$$

Where F is the force (newtons) exerted on a massive body through gravity, G is the gravitational constant (approximately 6.67×10^{-11} N m^2 kg^{-2}), m_1 is the mass (kilograms) of the body upon which the gravitational force is being exerted, m_2 is the mass (kilograms) of the body that is exerting the gravitational force, and and r is the distance (meters) between the two bodies. To incorporate this law into a 3-D model, 3 force vectors are calculated to account for movements in the x, y, and z directions.

Please note that, while very important, Newton's Law of Universal Gravitation is not the *only* important factor. For example, an attempt to simulate the effects of Saturn's magnetosphere on the may be revealing, but would have an immensely different focus. Time is still a limiting factor in this project, after all, and the processes that would need to be simulated for this to be an adequate portrayal would be numerous and complex. As such, this project will generally avoid such factors and focus on the movement of objects within the system unless enough time can be put aside to add these factors in.

By simulating the Saturnian moon system, one hopes to better understand the extent of our understanding. By basing this simulation upon commonly used models, we can gage how accurate and effective these models are. Furthermore, we can determine which phenomena we know how to explain which we don't, making it clearly which events are worth further research.

Development

The simulation runs using the computer language C. It should also be noted that OpenGL is heavily utilized. OpenGL is a graphics library for C and C++, and was mainly implemented for testing purposes.

Because of the number of data pieces involved for proper modeling of a space system, I was drawn immediately to the idea of creating a struct to hold data for each object in the system. Since this was such a fundamental necessity to the success of this project, the first step I took was the creation of such a structure. While the design has been slightly altered since this project began, the struct has remained for the most part the same. The code for the struct is as follows:

```
/* Struct representing planets, satellites, or other massive bodies */
struct mBody
{
        double *xLocs;
        double *yLocs;
        double *zLocs;
        char  *name;
        double mass;            /* In kilograms */
        double xLoc;            /* X component of location (distance from one point to the
adjacent is in kilometers) */
```

```
	double yLoc;		/* Y component of location (distance from one point to the
adjacent is in kilometers) */
	double zLoc;		/* Z component of location (distance from one point to the
adjacent is in kilometers) */
	double xVelocity;	/* X component of velocity vector (km/sec) */
	double yVelocity;	/* Y component of velocity vector (km/sec) */
	double zVelocity;	/* Z component of velocity vector (km/sec) */
	double red;
	double green;
	double blue;
};
```

The pointers *xLocs, *yLocs, *zLocs are used to store previous x, y, and z locations which are then printed to the GL window as dots, thereby tracing the path of each body. The amount of data to be stored is user set, with a default of 10000. It should be noted that these pointers are used solely for graphical purposes.

The string *name stores the name of each body (ex: Titan, Saturn, Hyperion, Mimas). This string is used to create file streams to files with their names and the string ".txt" appended to the end (ex: Titan.txt, Saturn.txt, Hyperion.txt, Mimas.txt). These files are then used for data storage concerning their respective bodies.

The variable mass refers to the objects mass. The variables xLoc, yLoc, and zLoc all store the current location of the body (with Saturn always at the origin). The variables xVelocity, yVelocity, and zVelocity indicate the vector components for the speed of the object. Red, green, and blue are used solely to determine the rgb values with which GL prints each body to the window.

This project stores the entire space system in a single array (s[]) of this struct. The project then iterates to the appropriate runtime, each time recalculating the values of each mBody within the array. It should be noted that I created helper functions that does the necessary projectile calculations. Here follows these helper functions:

```
double distance(struct mBody a, struct mBody b)
{
	double xDist = a.xLoc - b.xLoc;
	double yDist = a.yLoc - b.yLoc;
	double zDist = a.zLoc - b.zLoc;

	return pow(xDist * xDist + yDist * yDist + zDist * zDist, .5);
}

/* Recalculate an mBody's parameters during a time step */
void recalcL(struct mBody s[], int ind)
{
//	printf("\n\nFrom Recalc:\nCurrent distance: %lf\n", dist);
//	printf("%s => x: %lf y: %lf z: %lf\n", s[1].name, s[1].xLoc, s[1].yLoc, s[1].zLoc);
```

```
//        printf("%s => xVel: %lf yVel: %lf zVel: %lf\n\n", s[1].name, s[1].xVelocity, s[1].
yVelocity, s[1].zVelocity);

        s[ind].xLoc = s[ind].xLoc + (s[ind].xVelocity * tstep);
        s[ind].yLoc = s[ind].yLoc + (s[ind].yVelocity * tstep);
        s[ind].zLoc = s[ind].zLoc + (s[ind].zVelocity * tstep);
}

void recalcV(struct mBody s[], int ind, int numbodies)
{

        double g = 6.6742 * pow(10, -20);    /* Newton's Gravitational Constant */

        double xDist;
        double yDist;
        double zDist;
        double dist;
        double gforce = 0;
        double xForce = 0;
        double yForce = 0;
        double zForce = 0;

        int number;

        for(number = 0; number < numbodies; number++)
        {
                if(number != ind)
                {
                        xDist = s[number].xLoc - s[ind].xLoc;
                        yDist = s[number].yLoc - s[ind].yLoc;
                        zDist = s[number].zLoc - s[ind].zLoc;

                        dist = pow(xDist * xDist + yDist * yDist + zDist * zDist, .5);

                        gforce = ((g * s[number].mass * s[ind].mass) / (dist * dist));

                        xForce += gforce * (xDist / dist);
                        yForce += gforce * (yDist / dist);
                        zForce += gforce * (zDist / dist);
                }
        }

//        printf("\n\nFrom Recalc:\nCurrent distance: %lf\n", dist);
//        printf("%s => x: %lf y: %lf z: %lf\n", s[1].name, s[1].xLoc, s[1].yLoc, s[1].zLoc);
//        printf("%s => xVel: %lf yVel: %lf zVel: %lf\n\n", s[1].name, s[1].xVelocity, s[1].
yVelocity, s[1].zVelocity);

        s[ind].xVelocity = s[ind].xVelocity + ((xForce * tstep) / (s[ind].mass));
```

```
        s[ind].yVelocity = s[ind].yVelocity + ((yForce * tstep) / (s[ind].mass));
        s[ind].zVelocity = s[ind].zVelocity + ((zForce * tstep) / (s[ind].mass));
}
```

First of all, please note that commented sections, particularly printfs, are most likely used for debugging. As for the functions, distance is pretty self-explanatory, returning the distance between mBody a and mBody b. recalcV recalculates the velocity of the mBody at the passed index (ind) by summing the gravitational force vertors exerted by surrounding bodies and recalcL recalculates the velocity of the mBody at the passed index (ind) based upon the current velocity. These commands are executed for each iteration of the program for every mBody. Note that tstep is a global variable that denotes the amount of time that passes in an iteration, and the size of tstep determines the accuracy of the model (accuracy is indirectly related to tstep). Tstep can be user set, with a default of 500 seconds.

These sections of code are primarily responsible for all calculations (excluding the for loop that calls them). After I created these functions, I focused on making the project easier to test by setting up a graphical depiction using OpenGL, as well as setting up filestreams to store data. As such, these calculator functions have changed very little since their conception. I believe that now I will focus again on calculations. Here are some ideas I intend to implement in my code:

-First and foremost, I need to get a plotting software to allow for proper analysis of the stored data. I am currently tinkering with gnuplot.
-Another major concern is that I need to account for the irregular shape of the bodies, particularly concerning moons or objects similar to Hyperion (the irregular shape of Hyperion may partially account for its chaotic motion). This will be a difficult idea to implement, but I believe it is necessary for the accuracy of the model. I believe I can accomplish this by having each body being composed of numerous particles, and each will be acted upon by gravity. The mechanics of this technique, however, require more research.
-Because of the growing computational demands of this program, I have been considering using MPI to increase the programs speed. This task should be relatively simple to accomplish.
-For an accurate model, I need accurate data on the initial and relative positions of the bodies in question. I will need to obtain a sky chart for Saturn (or something to that effect).
-There are many more phenomenas that could be modeled, and I intend to look into the simulation of these phenomenas at a later date.