A Real Time Rocket Simulation

Alex Zylstra

May 10, 2007

1 Abstract

The purpose of this project is to create a real time simulation of a Saturn V rocket in various systems, which will allow for real time user control. The two systems currently implemented are the earth and moon as well as a binary planet system, although with the organization of the simulation larger, more complicated systems would be relatively easy to implement. The results of experimental trials of the two simulations indicates that human control is unlikely to be sufficient to achieve efficient orbital entry or to achieve a circular orbit, such as before the translunar insertion in the NASA Apollo missions, which should be reproducible in the simulation.

2 Introduction

The motivation for this project is to create the final product, the simulation, which will allow users to control a Saturn V simulation rocket in real time. This should be both fun and physically interesting, as it will help develop an intuitive sense of how rockets and orbits work. The project is also highly motivated by personal interest into the history of space exploration, and especially the Apollo missions.

Some simulation constants are attached as Table 2. These values are averages I found for the actual Saturn V rockets as used in the Apollo missions. [1]

3 Theory

The theory behind the physics of this simulation is fairly basic, as all we need are two simple concepts: Newton's law of universal gravitation, and rockets.

[1] From http://spider.ipac.caltech.edu/staff/waw/mad/ and http://en.wikipedia.org/Saturn_V

3.1 Gravitation

Thanks to the work of Newton, we know that every point mass exerts an attractive force on another point mass along the line joining them. Thus,

$$F = G\frac{m_1 m_2}{r^2} \tag{1}$$

where F is the magnitude of the resultant force, G is the gravitational constant, m_1 and m_2 are the masses, and r is the magnitude of the separation.
However, in this simulation, we are not dealing with point masses - we have planets and rockets. In the calculations, I treat these bodies as point masses, and we can quickly show that this is correct for spherical objects:
We need to define the mass density:

$$\rho = \frac{M}{\frac{4}{3}\pi R^3} \tag{2}$$

where R is the radius of the spherical body. Then,

$$a_{total} = G\int_v \frac{\rho}{(\vec{r}+\vec{r'})^2} d\tau' \tag{3}$$

$$a_{total} = \frac{GM}{\frac{4}{3}\pi R^3}\int_{\phi'=0}^{2\pi}\int_{\theta'=0}^{\pi/2}\int_{r'=0}^{R}\frac{1}{(\vec{r}+\vec{r'})^2} dr' d\phi' d\theta' \tag{4}$$

omitting the details of evaluating the integral,

$$a_{total} = \frac{GM}{\frac{4/3}{\pi}R^3}\frac{\frac{4}{3}\pi R^3}{r^2} \tag{5}$$

$$a_{total} = G\frac{M}{r^2} \tag{6}$$

which is what we would expect for a point mass of the same mass as the spherical object, at it's center.

3.2 Rockets

The main concept behind a rocket engine is conservation of mass. The rocket, which contains some amount of propellant, can be thought of as a single system with some initial momentum,

$$p_i = m_i v_i \tag{7}$$

A rocket engine essentially shoots fuel out of the back in a direction opposite to that of the rocket at some (usually high) velocity. By conservation of momentum, we know that

$$p_i = p_f \tag{8}$$

$$m_i v_i = (m_i + dm)(v_i + dv) - (dm)(v_i - v_e) = mv + m(dv) + (dm)v_e \tag{9}$$

Taking a time derivative gives us that

$$\frac{dp}{dt} = m\frac{dv}{dt} + v_e\frac{dm}{dt} \tag{10}$$

In the absence of external forces, of course,

$$\frac{dp}{dt} = 0 \tag{11}$$

$$m\frac{dv}{dt} = -v_e\frac{dm}{dt} \tag{12}$$

and we can easily see that the left hand side of the above equation is a force, so, the rocket experiences a thrust. Since $\frac{dm}{dt} < 0$, we can say that the thrust is

$$F = v_e\frac{dm}{dt} \tag{13}$$

which we can use for numerical calculations.

4 Equipment

As a simulation experiment, the needed equipment was limited. A computer with a Java compiler and run-time environment was used to write and execute the necessary code.

5 Procedure

The code for the simulation was written in Java, with the following approach:
The basis of this simulation is object oriented programming. There are three crucial objects:

5.1 Objects

- MassObject
  This is a top level class which implements a few important methods that will be needed by any physical object in the system, chiefly the following:
  - timeStep
    This method takes as inputs the force on the object and a time, and does an Euler approximation over that time period for the object, and updates the velocity and position.
  - Setter/Retriever functionality
    A variety of functions to either set or return the mass, position, or velocity, which will be needed when implementing the simulation.

- Planet
  This class extends the functionality of MassObject, and implements the following methods:
  - detectCollision
    This method takes a position as input and returns true if the position is within the volume of the planet, and false otherwise. This is needed for collision detection.
  - gravAcc
    As shown in the Theory section, the acceleration some distance away from a body is as in equation 6. This method does that calculation for a given position, and returns the resulting acceleration.
- Rocket
  This object implements a rocket, based off of the characteristics of a Saturn V. Thus, it contains the necessary constants for the three different stages of the Saturn V. The most important methods implemented here are
  - timeStepBurn
    This method is similar to the timeStep method earlier, except in this case we want the rockets to be firing during the time period given. Thus, this method calculates the force exerted on the rocket by the thrust of the engines and adds that to the outside acceleration given in the input, and uses the MassObject method to perform an Euler approximation for the position and the velocity. Meanwhile, this method uses an Euler approximation for the mass.
  - rotate
    This method changes the orientation of the rocket in the simulation plane by one degree, left or right depending on the input boolean.

The rest of the code for the simulation mainly performs tasks related to displaying the GUI and handling the controls, which are as shown in Table 1, which is attached.
The main classes also handle calling the appropriate methods for the objects at the right time, and passing results between them.

5.2 Running the simulation

Since this simulation is written as a Java applet, it can be run by opening one of the .html files included.

5.3 Simulation Code

The complete set of Java and HTML code required to run the applets has been attached to this paper.

6 Analysis

Unfortunately, there are no quantitative results for this experiment. While the simulation could be modified to output numerical results and data about the orbits induced, since each is the product of a very specific set of human instructions to the rocket during take off, such results are unlikely to be reproducible or particularly interesting, so I shall stick with qualitative results. I discuss each in the two systems below, split by the specific experiment in question.

6.1 Earth and Moon

In this simulation, the rocket starts on the Earth. The moon is set up to orbit the Earth as in it's actual orbit, which is slightly elliptical, and this behavior has been observed in the simulation. Unfortunately, due to the distance between the Earth and the Moon, at a scale where both bodies are visible, the Moon is only a couple pixels, and the Earth is not many more, so observing anything at that scale would be difficult.

Stable elliptical orbits are easily achievable by hand. The general method used consists of allowing the first stage of the rocket to burn up to a kilometer or two altitude, then pitching the rocket in the desired direction with a small angle to the horizontal, generally about thirty degrees. Towards the end of stage 2 and the beginning of stage 3 the rocket is pitched more towards the direction of the desired orbit.

The first two attached figures show this. Figure 1 shows the rocket shortly after takeoff, and Figure 2 shows the rocket after takeoff and a subsequent burn in orbit, which has resulted in a large elliptical orbit. The reason that the orbit does not appear to be perfectly elliptical in this system is unknown, as more elliptical orbits have been observed in other, unrecorded trials. Figure 3 then shows this system approximately 40 orbits later. We can see that approximation errors have resulted in a significant drift of the orbit over time. I think that this behavior may be related to large values for the time multiplier, as huge errors result at extremely large values of the multiplier such as 1024 or 2048.

Figure 4 shows this same system shortly after takeoff on a different run. The scale has been increased enough that the moon is barely visible at the bottom edge of the simulation, above the text.

6.2 Binary Planets

The system was extended to a binary planet, in which two planets, each roughly half the mass and half the volume of the Earth, orbit their center of mass. A sample orbit in this system is attached as Figure 5. Stable orbits are extremely difficult to achieve, as the rocket is easily ejected from the system. Another barrier is that the initial conditions for the two planets are limited in accuracy, which has resulted in slightly elliptical orbits for each, which, in the attached screen shot, led to the collision. The center of mass of the system has moved. We do, however, observe a very expected elliptical orbit far from the planets.

In Figure 6, we can see an rocket which has achieved sufficient velocity to escape from the system.

We can easily hypothesize that odd orbits will result if the rocket goes between the two planets of the binary system. We can see such an orbit in Figure 7. Escape from the system is possible after passing between the planets, but is trickier than a normal escape orbit as in Figure 6.

7 Conclusion

I have successfully designed and implemented a simulation of a Saturn V rocket. Numerical approximations are done in real time, which means that control instructions to the rocket can be input in real time, making the rocket human controllable with an additional applet GUI implementation.

The code for this project has all been written in Java. This had the benefit of making the GUI implementation easier, which was a significant bonus. Unfortunately, Java is not very computationally friendly. The simulation has only been run so far on fast computers, on which it has utilized a small but significant percentage of the CPU - between 10 and 30%, dependent on the time multiplier. For this reason, an implementation in C may be better suited for general use.

As shown in Figure 3 and discussed in the Analysis, significant orbit drift occurs over long periods of time with a high time multiplier, such as one in the thousands. Obviously, this results from the Euler approximations done in the simulation, but the exact cause has not been found.

While unable to mimic the NASA Apollo missions, which would be interesting to do in a simulation, I have been able to achieve interesting orbits in both systems. In the Earth system, elliptical orbits are easy to obtain, and characteristics of these orbits are observable. In the binary system, stable orbits are extremely difficult to achieve, such that I have not

managed to do so yet. However, odd and unconventional orbits are easy and interesting to observe in this system.

Another feature of this simulation is that it is easily extendible to other systems, so that in the future this base code can be used to simulate other interesting systems.

Table 1: Simulation Controls

Action	Key
Toggle Engines On	w or i
Toggle Engines Off	s or k
Turn 'left'	a or j
Turn 'right'	d or l
Increase time multiplier	x
Decrease time multiplier	z
Increase pixel scale	m
Decrease pixel scale	n

Table 2: Simulation Constants

Name	Value
$\frac{dM}{dt}$, stage 1 (kg/s)	13,583
v_e, stage 1 (m/s)	2989
$\frac{dM}{dt}$, stage 2 (kg/s)	1186
v_e, stages 2&3 (m/s)	4177
$\frac{dM}{dt}$, stage 3 (kg/s)	198