Trajectory Animator Java Applet

John E. Weglian

In partial fulfillment of the degree of Master of Science in Aerospace Engineering

Georgia Institute of Technology
Atlanta, GA

December 2001

Table of Contents

List of Figures

List of Tables

I. Introduction

Low thrust trajectories allow for very complicated, programmable thrust profiles to shape the orbit. Also it is often desirable or necessary to only apply thrust at a certain point in a trajectory such as at periapsis. The Trajectory Animator allows the user to program complicated thrust profiles based on the position in the orbit (specified by true anomaly or mean anomaly), and then see how the orbit develops over time.

Trajectory Animator is a Java applet that can be run from any windows browser that supports Java. The HTML code for the applet will prompt a user to automatically download the Java plug-in if it is not already installed on the user's machine.

II. Description

A. Main Window

The main window in the Trajectory Animator consists of several parts. The top section contains the main control buttons to start/pause, define the initial orbit, define the thrust profile, define the vehicle parameters, and to restart the animation/calculations. The middle section contains zoom and speed controls as well as a button to show or hide the history of the orbit, and a button to smooth the track data (at the cost of slower performance). The bottom section contains the information pertinent to the orbit and a status label that appears when the animation reaches an end point. The main window is shown in Figure 1. The orbital information displayed include the semi-major axis (a), the eccentricity (e), the orbital energy (E), the specific angular momentum (h), the orbital period, the total elapsed time, the apoapsis (ra), the periapsis (rp), the true anomaly, the mean anomaly, the magnitude of the position vector (r), the altitude (alt), the magnitude of the velocity vector (v), the mass of the vehicle, the magnitude of the acceleration, and the flight path angle (Phi).

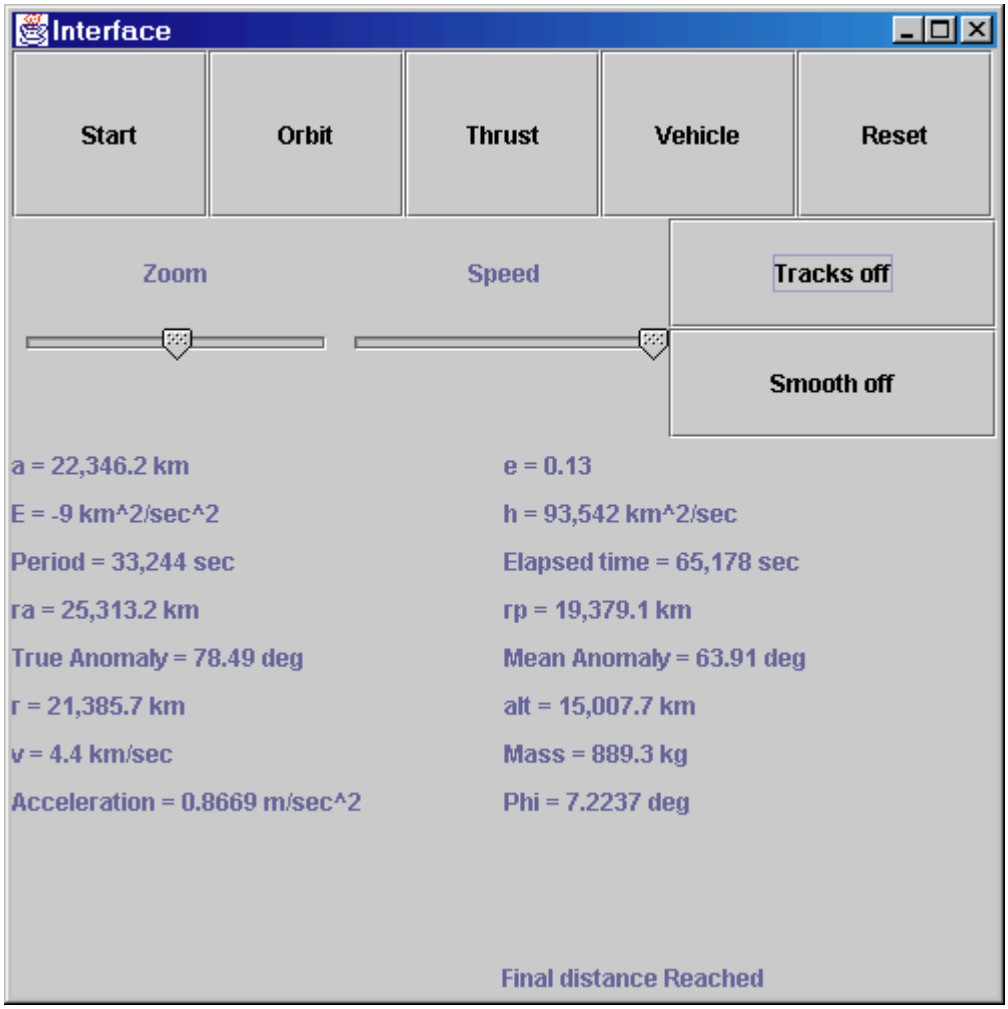

Figure 1: Main Window

B. Trajectory Animation Window

The trajectory animation window shows the sun or planet, the starting orbit in green, the final distance in red and the current position as a black circle. When the animation begins, the path that the spacecraft follows is also shown in black, unless the Tracks Off button is clicked to suppress them. A representative trajectory animation window is shown in Figure 2.

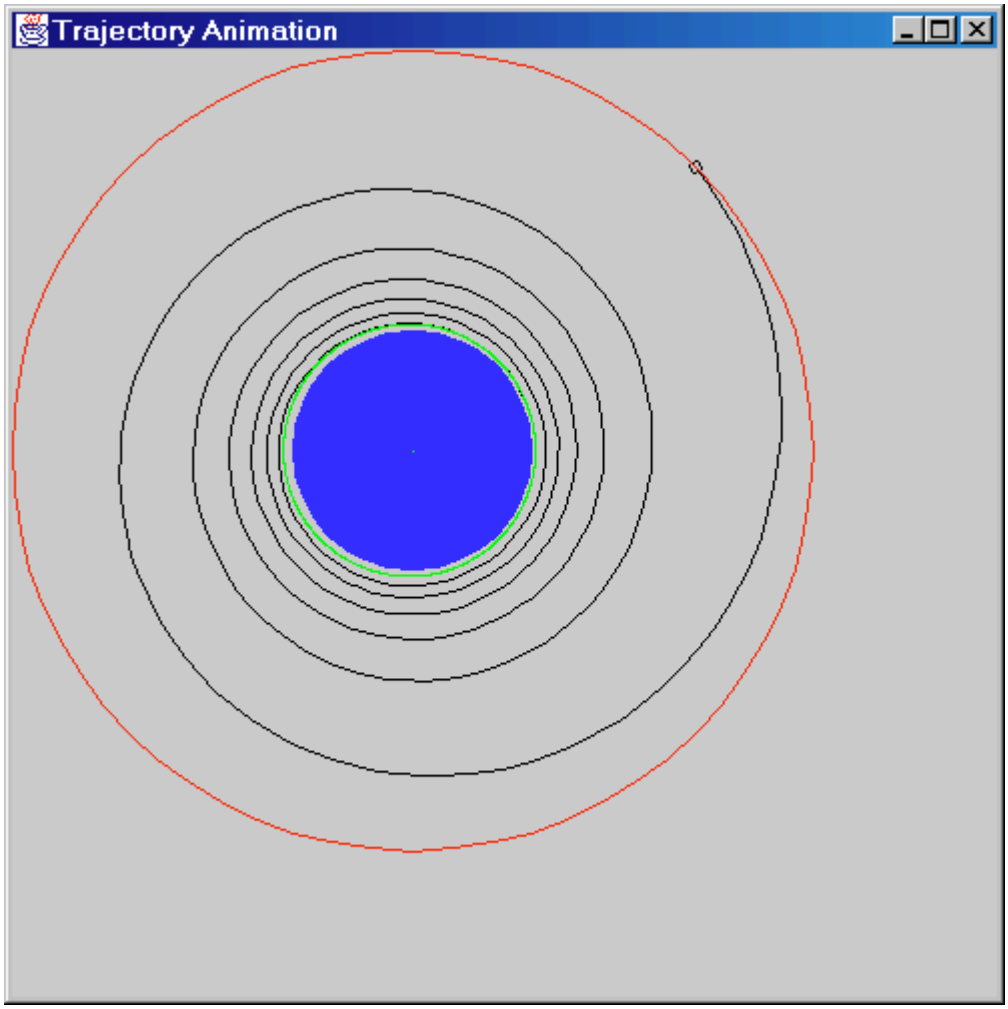

Figure 2: Trajectory Animation Window

C. Orbit Definition Window

The orbit definition window is used to define the initial orbital parameters and the final point at which the animation stops. A set of buttons allow the choice of the main body in the system. The choices are the sun and any of the planets. The unit of length can be changed from the default (meters). The initial orbit can be defined by supplying a pair of orbital parameters. The choices for the parameters are the semi-major axis and the eccentricity (a & e), the apoapsis and periapsis (ra & rp) or the apoapsis altitude and periapsis altitude (ha & hp). If the desired starting point in the orbit is not at a true anomaly of 0.0, it can be set to another value. The final position is entered as a radial

distance from the center of the planet/sun (a & e or ra & rp options) or as the final altitude (ha & hp option). The orbit definition window is shown in Figure 3.

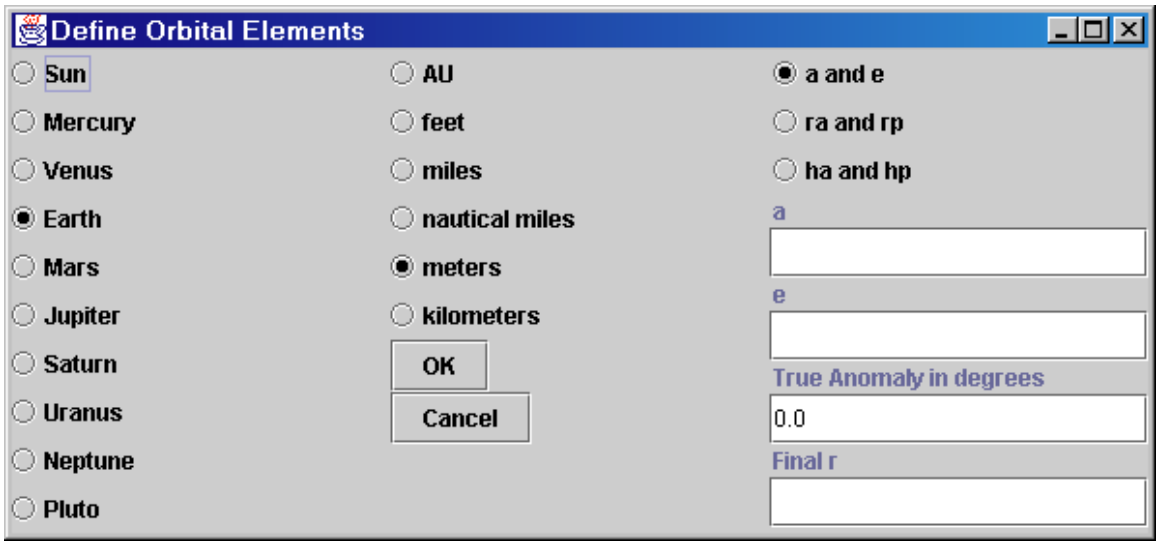

Figure 3: Orbit Definition Window

D. Thrust Definition Window

The thrust definition window is used to define the thrust profile over the course of the trajectory. The thrust profile is based on the true anomaly or mean anomaly of the spacecraft in orbit. Multiple thrust definitions can be pieced together so long as all 360 degrees are defined. The thrust direction and magnitude are linearly interpolated between a starting and ending value specified by the user with the range of true (mean) anomaly. The starting and ending thrust direction can be relative to the velocity or the position vector. The angle is measured counter-clockwise from the vector, so -5 degrees relative to the velocity vector points slightly away from the Earth in a circular orbit. Likewise, +90 degrees from the position vector (defined from the center of the orbital body to the spacecraft) is along the velocity vector for a circular orbit. The starting and ending magnitude of thrust can be specified in N, mN, or lbs. The thrust definition window is shown in Figure 4.

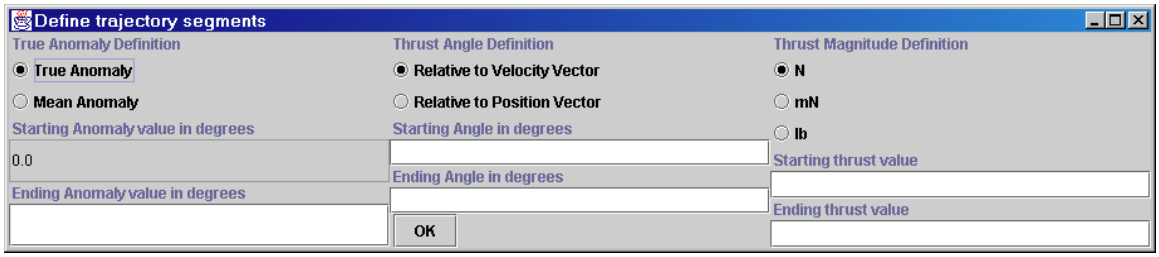

Figure 4: Thrust Definition Window

E. Vehicle Definition Window

The vehicle definition window defines the vehicle mass and the propulsion system's specific impulse (I_{SP}) in seconds. The vehicle mass can be defined in kg, lb_m, or slugs. The default values are 1000 kg and 3000 sec. The vehicle definition window is shown in Figure 5.

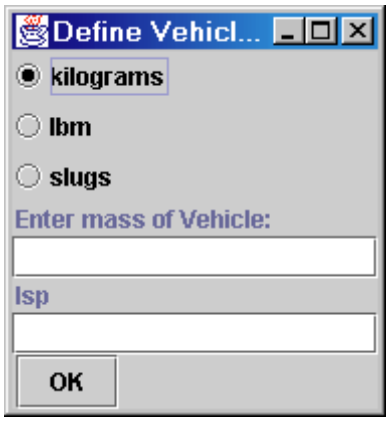

Figure 5: Vehicle Definition Window

III. Calculations

The program utilizes an iteration loop to calculate position, velocity, acceleration, and mass of the vehicle, as well as the orbital parameters and the time elapsed. This data is used to update the animation of the orbit and the orbital data listed in the main window.

A. Acceleration calculation

The acceleration due to gravity is calculated based on the distance from the center of the body using Equation 1 [1].

$$\vec{a} = -\frac{\mu}{r^2}\hat{r} \tag{1}$$

Thrust is determined from the user-defined thrust profile. True (mean) anomaly of the orbit is calculated and used to determine the applicable region of the thrust profile. A linear interpolation between the minimum and maximum thrust is used to determine the appropriate thrust magnitude. Similarly, the angle relative to the position vector or the velocity vector as specified by the user is determined from a linear interpolation between the minimum and maximum angle specified. The magnitude and angle are then used to determine the vector for thrust. Equation 2 shows this interpolation scheme for the angle using the true anomaly, θ, as the independent variable, and Equation 3 shows the interpolation for the thrust magnitude, T, using the mean anomaly, M, as the independent variable. Which variable is actually used as the independent variable is determined by the user's selection in the thrust definition window.

$$Angle = \frac{Angle_{final} - Angle_{initial}}{\theta_{final} - \theta_{initial}}(\theta - \theta_{initial}) + Angle_{initial} \tag{2}$$

$$T = \frac{T_{final} - T_{initial}}{M_{final} - M_{initial}}(M - M_{initial}) + T_{initial} \tag{3}$$

Newton's second law of motion is shown in Equation 4. Using the chain rule leads to Equation 5. Typically this is reduced to Equation 6 with the assumption that mass is constant. This is normally a very bad assumption to make for solving problems involving rockets where the expulsion of propellant (and therefore a change in mass) is used to produce the thrust. In this case, however, the time step used is so small that the instantaneous acceleration follows the result of Equation 6. This method is presented in

Reference 2. The mass of the vehicle will be calculated for the next time step, so the effect of the changing mass is taken into account by the numerical integration scheme.

$$\vec{T} = \frac{d(m\vec{v})}{dt} \tag{4}$$

$$\vec{T} = \frac{d(m)}{dt}\vec{v} + \frac{d(\vec{v})}{dt}m \tag{5}$$

$$\vec{T} = m\vec{a} \tag{6}$$

The total acceleration on the vehicle is the vector sum of the acceleration due to gravity and the acceleration from the thrust imparted by the vehicle's engines.

B. Velocity and Position Calculations

The velocity is calculated by integrating the acceleration as shown in Equation 7. The numerical approximation uses a finite time step, Δt, and approximates the relationship as shown in Equation 8.

$$\vec{v} = \int \vec{a}\,dt \tag{7}$$

$$\vec{v} \approx \vec{v}_0 + \vec{a}\Delta t \tag{8}$$

Similarly, the position is calculated by integrating the velocity as shown in Equation 9, and the numerical approximation is shown in Equation 10.

$$\vec{r} = \int \vec{v}\,dt \tag{9}$$

$$\vec{r} = \vec{r}_0 + \vec{v}\Delta t \tag{10}$$

C. Mass and Time Calculations

The mass flow rate of propellant expelled is determined from the specific impulse and the thrust as defined in Equation 11 where g_c is the gravitational acceleration at sea level on the Earth. The mass of the vehicle decreases as a function of time according to Equation 12. The numerical approximation of the final mass is given in Equation 13. Note that in these equations, $\dot{m}$ is the *propellant* mass flow rate and not the rate of change of mass of the vehicle. The rate of change of mass of the vehicle is the negative of this number.

$$I_{SP} = \frac{T}{\dot{m} g_c} \tag{11}$$

$$m = m_0 - \int \dot{m} dt \tag{12}$$

$$m = m_0 - \dot{m} \Delta t \tag{13}$$

The final time is simply the initial time plus the time step as shown in Equation 14.

$$t = t_0 + \Delta t \tag{14}$$

D. Determination of the Time Step

During any particular iteration, the time step is initially the time step from the previous iteration. The above numerical approximations are used to calculate the final position and velocity of the vehicle. They are then calculated again using half that time step two times. If there were no errors due to the approximations, the two solutions should come up with the same answer, since the total time elapsed for both cases is the same. Comparing the two solutions gives an estimate of the error in the calculations. If

this difference exceeds the tolerance ($1x10^{-8}$ times the smaller of the initial periapsis or final distance from the center of the body), the time step is reduced. If the difference is less than half of the tolerance, the time step is increased. In this way, the magnitude of the error is always kept relatively constant.

E. Determination of the Orbital Elements

The orbital elements are determined from the position and velocity vectors of the spacecraft and the gravitational parameter, μ, of the body which is being orbited. The orbital elements are not used to determine spacecraft position or motion, but they are used to determine the true and mean anomaly for the thrust profile. The orbital mechanics equations all come from Reference 1.

The energy of the orbit, E, is calculated from the magnitude of the velocity vector and the magnitude of the position vector as shown in Equation 15.

$$E = \frac{1}{2}v^2 - \frac{\mu}{r} \tag{15}$$

The specific angular momentum vector, h, is calculated by taking the cross product of the position and velocity vectors as shown in Equation 16. Only the magnitude of h is used in the calculations.

$$\vec{h} = \vec{r} \times \vec{v} \tag{16}$$

The flight path angle, ϕ, is determined from Equation 17. Since the cosine is symmetric about the y-axis, the arc-cosine only determines the value from 0 to 180 degrees. The sign of ϕ is the same as the sign as $\vec{r} \bullet \vec{v}$.

$$h = rv\cos\phi \tag{17}$$

The semi-major axis, a, is determined from Equation 18.

$$a = -\frac{\mu}{2E} \tag{18}$$

The semi-latus rectum, p, of the orbit is found by Equation 19.

$$p = \frac{h^2}{\mu} \tag{19}$$

The eccentricity, e, can be found then from Equation 20. Numerical inaccuracies can cause p/a to be greater than 1.0, even though this is a non-physical result. If this is the case, e is set to 0.0.

$$e = \sqrt{1 - \frac{p}{a}} \tag{20}$$

The periapsis, r_p, and apoapsis, r_a, are found from Equations 21 and 22, respectively.

$$r_p = a(1 - e) \tag{21}$$

$$r_a = a(1 + e) \tag{22}$$

From the definition of specific angular momentum, the velocity at periapsis, v_p, and the velocity at apoapsis, v_a, can be found from equations 23 and 24.

$$v_p = \frac{h}{r_p} \tag{23}$$

$$v_a = \frac{h}{r_a} \tag{24}$$

The true anomaly, θ, is determined by rearranging equation 25 to solve for θ as shown in Equation 26. If $\vec{r} \bullet \vec{v}$ is negative, then the sign of θ is negative. Due to numerical inaccuracies, if the argument of the arc cosine in Equation 26 is greater than 1.0, θ is set to 0 degrees, and if it is less than –1.0, θ is set to 180 degrees. Note that θ is not defined for a circular orbit, so for any orbit with a sufficiently small e, θ is set to 0.

$$r = \frac{p}{1 + e\cos\theta} \tag{25}$$

$$\theta = \cos^{-1}\frac{\frac{p}{r} - 1}{e} \tag{26}$$

The period is calculated by Equation 27.

$$Period = \frac{2\pi}{\sqrt{\mu}} a^{\frac{3}{2}} \tag{27}$$

The calculation of mean anomaly, M, depends on the type of orbit. The mean anomaly, similar to the true anomaly, is not defined for a circular orbit, and is therefore set to 0 in this case. The equations for calculating mean anomaly are taken from Reference 3. For an elliptical orbit where $0 < e < 1$, M is given by Equation 28 where the parameter u is given by Equation 29. For a hyperbolic orbit, where $e > 1$, M is given by Equation 30 where the parameter u is given by Equation 31.

$$M = u - e\sin u \tag{28}$$

$$u = \cos^{-1}\frac{e + \cos\theta}{1 + e\cos\theta} \tag{29}$$

$$M = e\sinh u - u \tag{30}$$

$$u = \cosh^{-1}\frac{e + \cos\theta}{1 + e\cos\theta} \tag{31}$$

IV. Java Programming

A. Object Oriented Programming

The Java programming language uses object-oriented programming. The code is written using multiple *classes*, which define how the data is used and manipulated. An *object* is a collection of data belonging to a given class. For example, one class used in the Trajectory Animator is the MyVector class. (Java has it's own Vector class which is not related to 2D vectors) In this class, there are class *methods* that can manipulate the data contained in the MyVector object. Position and velocity are two objects of the MyVector class used in the Trajectory Animator. The MyVector class contains methods that can be used to calculate the dot product, cross product, or addition of two MyVector objects as well as several other useful functions. This style of programming allows the programmer to create a library of useful tools (the classes) that are easy to manipulate to produce the desired outcome. The source code for all of the Java classes is presented in Appendix A.

B. Animation

Java makes graphical user interfaces extremely user friendly, including animation. In order to animate the orbits, it was necessary to convert the orbital coordinates to pixels on the screen. The geometric modeling techniques, a series of affine transformations, are given in Reference 4. First, the center of the orbital picture is moved to the origin. Then the coordinates are scaled to the proper dimension (which can later be changed by a

scaling factor set by the user with the zoom slider bar). Next the picture is translated so that the center of the picture is in the center of the window, which is how it is displayed. The same transformations are applied to the orbital body, the initial orbit, and the final position.

V. Verification

A. Orbital Progression

Simply animating an orbit without any thrust applied demonstrates the accuracy of the orbital progression. The green orbit in the trajectory animation window is determined by plotting 1000 points of the orbit based on the initial orbital parameters. For orbits that are not highly elliptical, the error is typically so small that it is hard to distinguish any difference between the two paths. The change in the orbital parameters in the main window is a better indication of the error. If there were no error, the orbital parameters would remain constant. Typically, the specific angular momentum does not change at all. For the case of a = 15,000 km about the Earth and e = 0.3, the Energy is 1% off after 7 complete orbits.

B. Thrust Application

Thrust can be applied in any magnitude desired. Typically thrust is considered to be "high thrust" which for calculations can be assumed to be nearly instantaneous or "low thrust" where the thrust is applied continuously over a long period of time, usually with a much more efficient (higher I_{SP}) engine. The trajectory animator can simulate both types of thrust profiles, although defining the true anomaly limits for a high thrust case is not intuitive.

1. High Thrust

As a test case, a sample orbit was created with a = 8000 km and e = 0.1. The main window after this setup is shown in Figure 6. The orbital elements shown agree with those calculated using Equations 13 through 25.

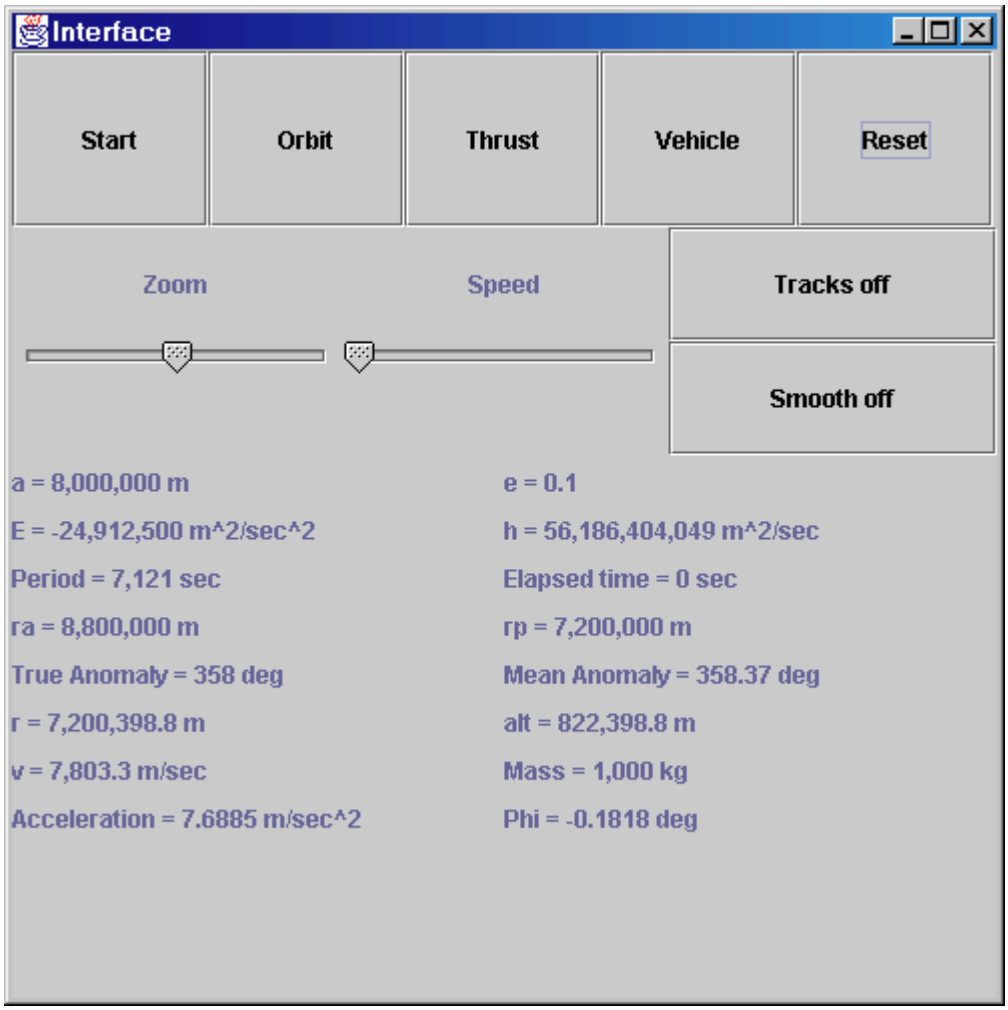

Figure 6: High Thrust Test Case Initial Status

A thrust of 10,000 N from an engine with an I_{SP} of 460 sec is used in this test case. The engine is on from a true anomaly of 358 to 002 degrees. (Note that in running the program, since the thrust definition is always started at 0 degrees true or mean anomaly, this is set in 3 steps: 0 to 2 degrees with 10,000 N of thrust, 2 to 358 degrees

with zero thrust, and 358 to 360 degrees with 10,000 N of thrust. The orbit is started at a true anomaly of 358 degrees. The thrust angle is defined as 0 degrees relative to the velocity vector.) The results of this simulated burn are shown in Figure 7. The total elapsed time is only 60 seconds, so this is a good approximation of an instantaneous burn for ΔV calculations. The calculated change in velocity over this time is 8448.7 m/sec – 7803.3 m/sec or 645.4 m/sec. The calculated change in mass is from 1000 kg to 866.3 kg. The ratio of initial mass to final mass is 1.154. Using Equation 32, the rocket equation, the calculated ratio is 1.154, which is equivalent to the calculated value.

$$\frac{m_i}{m_f} = e^{\frac{\Delta V}{g_c I_{SP}}} \tag{32}$$

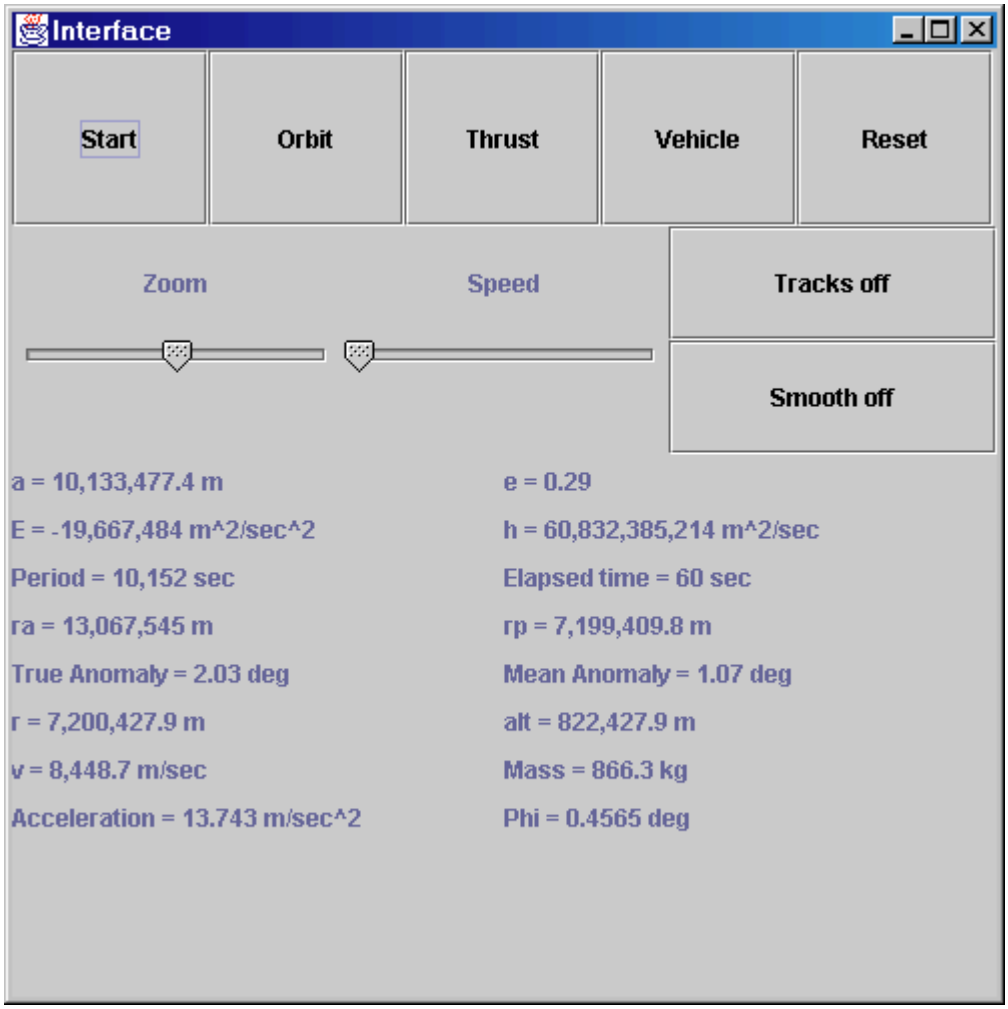

Figure 7: High Thrust Test Case Final Status

If the above test case is repeated using a very high I_{SP} ($1x10^9$ sec), after a 60 second burn time, the velocity is 8404 sec (see Figure 8). This is equivalent to a 600.7 m/sec ΔV. The predicted ΔV using Equation 33 is 600 m/sec. Equation 33 is simply the application of Equation 8, substituting thrust/mass for acceleration. In this case, since the mass is constant, it is permissible to use this equation without the instantaneous acceleration assumption using very small time steps.

$$\Delta V = \frac{\Delta t * T}{m} \tag{33}$$

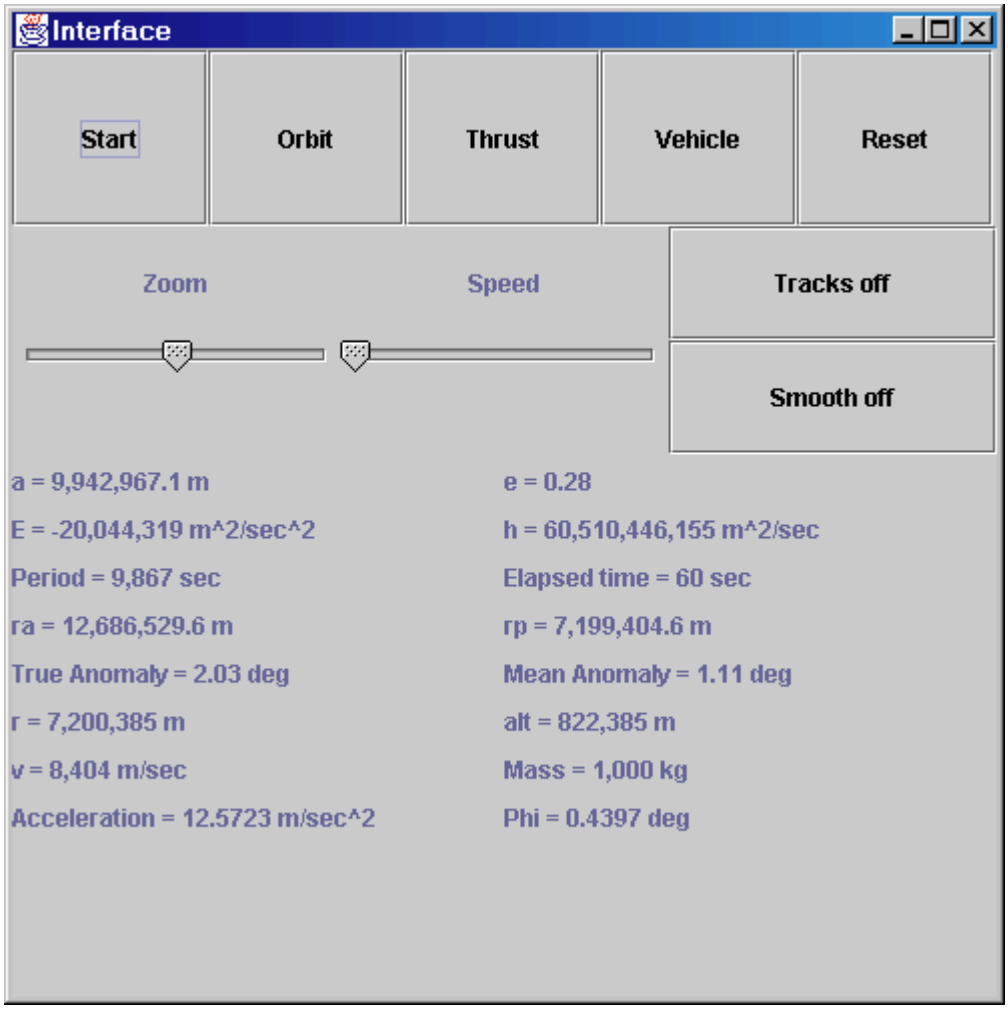

Figure 8: High Thrust, Constant Spacecraft Mass Test Case

2. Low Thrust

The output of the Trajectory Animator was compared to two 4th order Runge-Kutta simulations of a low thrust trajectory [5,6]. These simulations included the oblateness of the Earth, which the Trajectory Animator does not. In the simulations, a 10 kg spacecraft uses an engine with 1 N of thrust at a specific impulse of 3000 seconds. The simulation required the spacecraft to move from a 7015 km circular orbit to a final distance of 42235 km. In the simulation, the thrust was vectored 5 degrees clockwise from the velocity vector for the first 8068 seconds, and then was vectored 2 degrees

clockwise from the velocity vector for the remainder of the flight. In the simulation, the elapsed time was 45,875 seconds and the final mass of the spacecraft was 8.44 kg. Figure 9 shows a plot of this solution (1 DU = 6378 km).

This condition was simulated using the Trajectory Animator. The initial conditions are shown in Figure 10. The thrust was defined to be 1 N, -5 degrees relative to the velocity vector for all 360 degrees of the true anomaly. Once 8068 seconds had passed, the simulation was paused, and the thrust redefined with the only change being thrust applied -2 degrees from the velocity vector (Trajectory Animator references angles counter-clockwise).

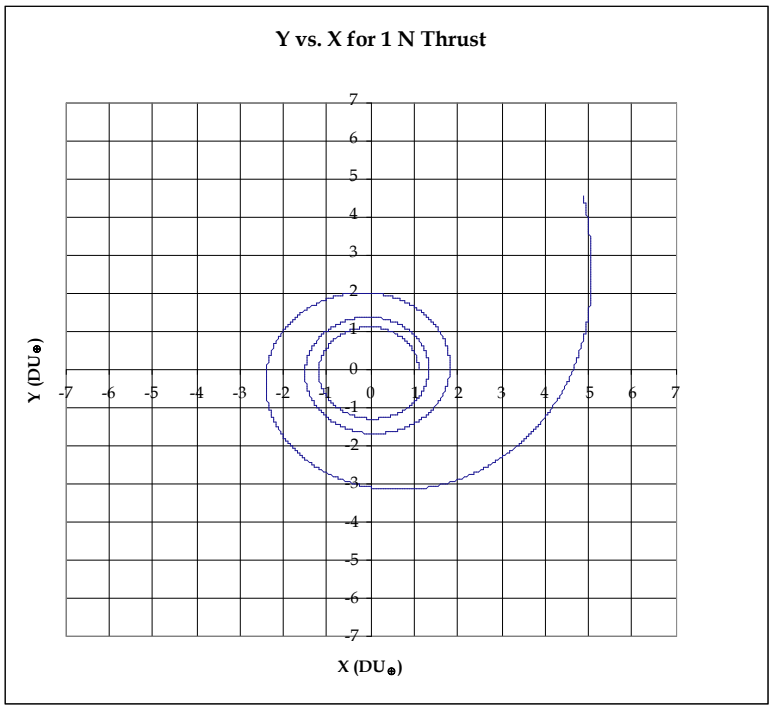

Figure 9: 4th Order Runge-Kutta Solution

The final status of the orbital propagation is shown in Figure 11, and the animation of the orbit is shown in Figure 12. The animation is almost identical to that in Figure 9. A comparison of the results is shown in Table 1. As can be seen, the results are very close. The difference in calculated time was only 591 seconds, which is only 1.3% of the Runge-Kutta solution value of 45,409 sec.

Figure 10: 1 N Low Thrust Test Case Initial Settings

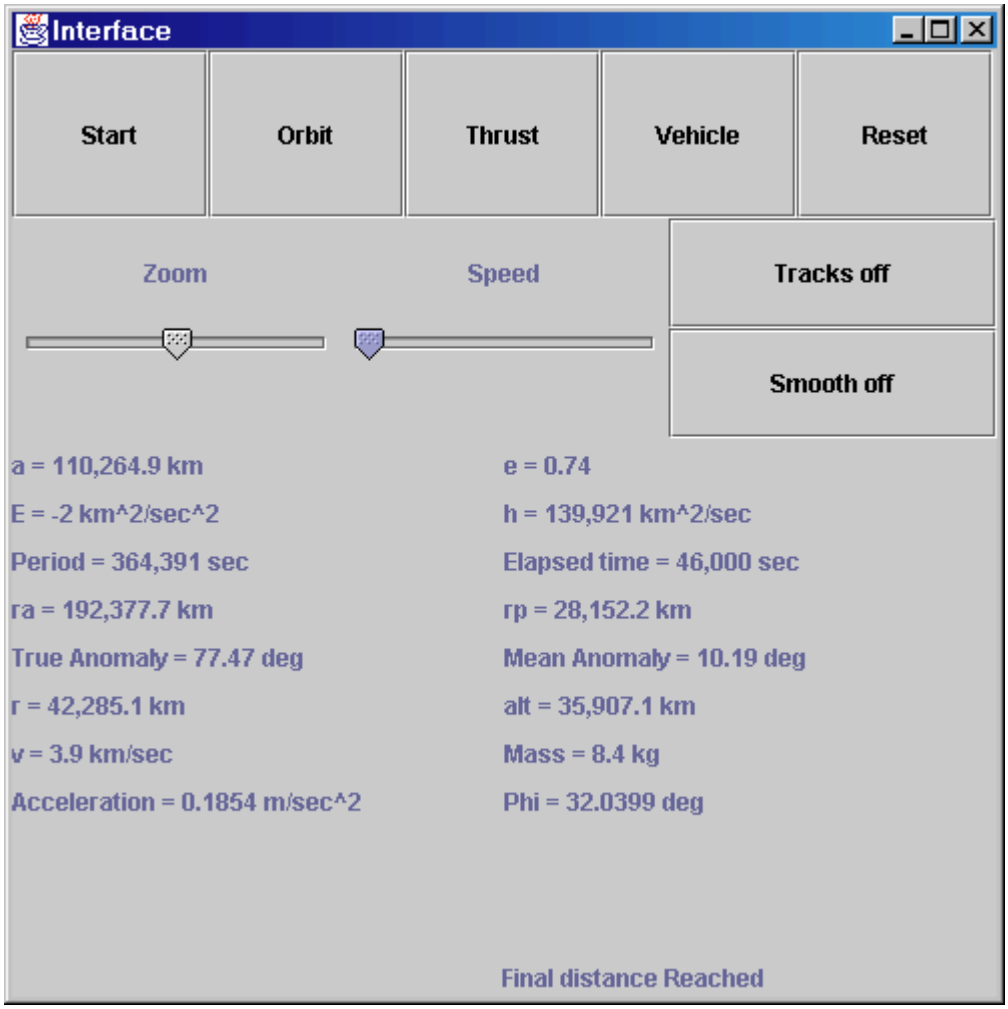

Figure 11: 1 N Low Thrust Test Case Final Settings

Table 1: Comparison Between Runge-Kutta Code and Trajectory Animator, 1 N Thrust

	4^{th} Order Runge-Kutta	Trajectory Animator
Final Mass	8.43 kg	8.4 kg
Final Eccentricity	0.76	0.74
Propagation Time	45,409 sec	46,000 sec

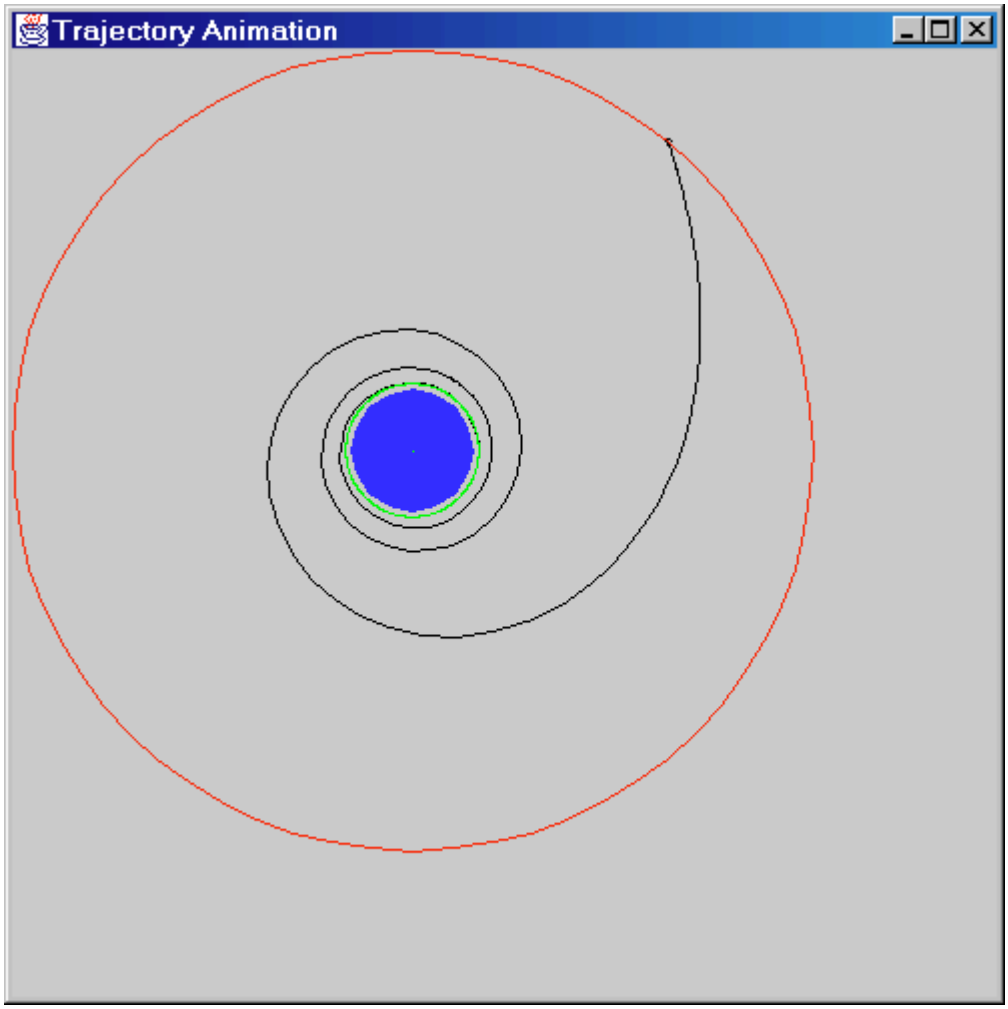

Figure 12: 1 N Low Thrust Test Case Animation Window

Another case using the same conditions except 0.01 N of thrust was also run for comparison. The comparison of these results to the higher order solution is presented in Table 2. In this case, the difference in the time of flight is 3718 seconds, which is only 0.1 % of the Runge-Kutta solution of 4,150,840 sec. The final orbital parameters are shown in Figure 13, and the animation is shown in Figure 14. In order to run this case more quickly, the data was not smoothed, and this can be seen in Figure 14, but it does not affect the accuracy of the calculations.

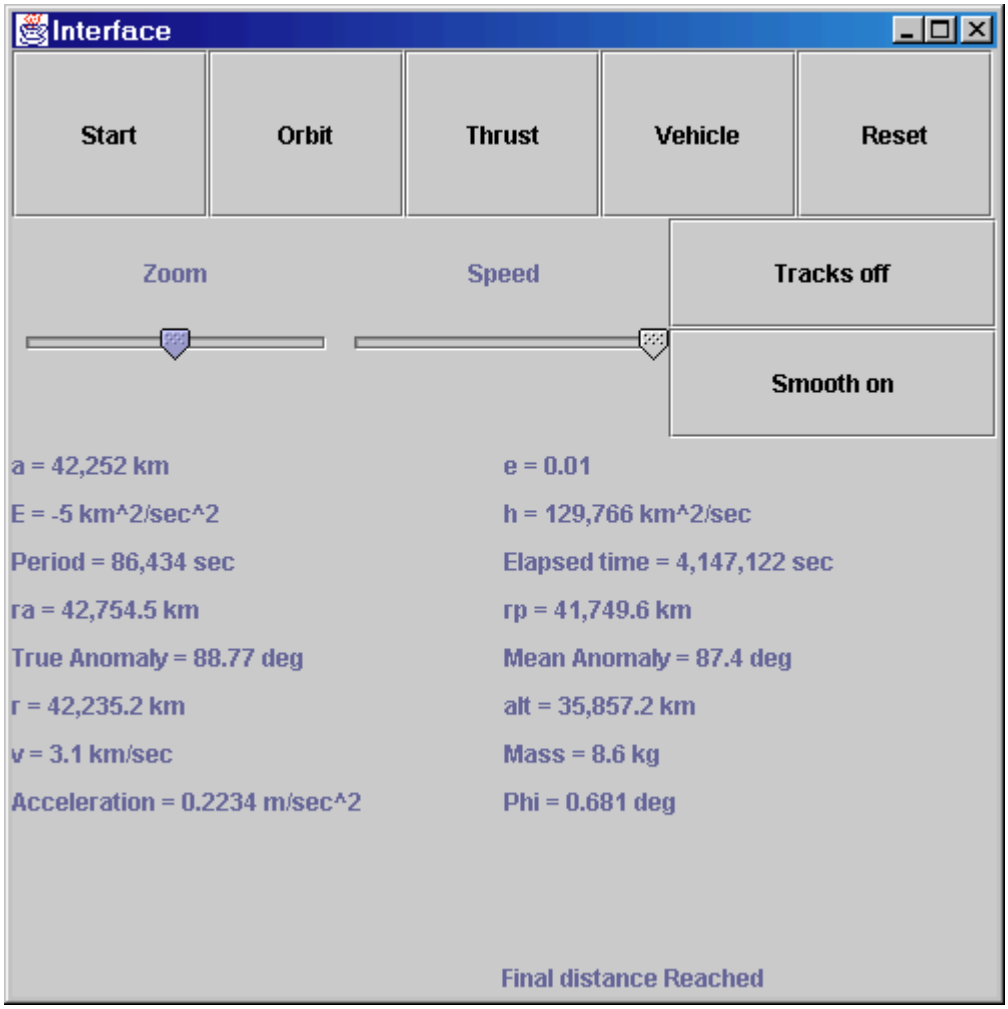

Figure 13: 0.01 N Low Thrust Test Case, Final Results

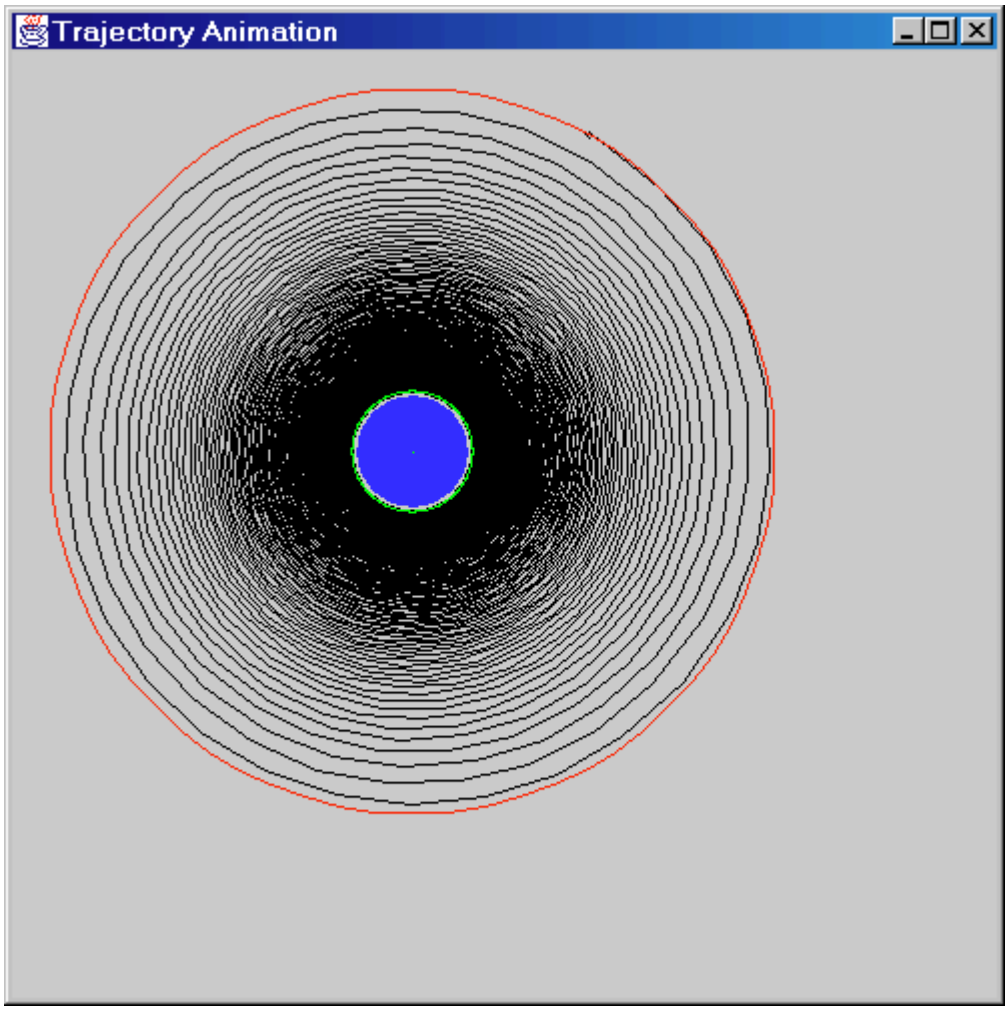

Figure 14: 0.01 N Low Thrust Test Case Animation Window

Table 2: Comparison Between Runge-Kutta Code and Trajectory Animator, 0.01 N Thrust

	4^{th} Order Runge-Kutta	Trajectory Animator
Final Mass	8.58 kg	8.6 kg
Final Eccentricity	0.01	0.01
Propagation Time	4,150,840 sec	4,147,122

VI. Limitations

The Trajectory Animator works best for orbits that are not highly elliptical. All orbits will see errors near the apoapsis where the acceleration due to gravity is smaller. This error becomes pronounced for orbits with an eccentricity greater than about 0.6.

Small numerical errors coupled with the small acceleration can accumulate and make a significant change to the orbit. This is seen as the apoapsis in subsequent orbits is reduced without any thrust applied. If an orbit is animated without any thrust applied, the user can see after several orbits the magnitude of the error in the orbit.

For thrust definition, the true or mean anomaly must be specified between 0 and 360 degrees. For an elliptical orbit, this does not pose a problem, as all points on the ellipse are contained in this range. For a hyperbolic orbit, however, the mean anomaly extends from negative infinity to positive infinity, and the true anomaly is bounded by the asymptotes, so it does not encompass all of the 360 degrees. If a hyperbolic orbit is used, the thrust should be defined with true anomaly, and in the thrust definition, values must be supplied for the values of true anomaly that will never be reached, even though the program will not make use of them.

VII. References

1. Bate, Roger R., Mueller, Donald D., White, Jerry E., *Fundamentals of Astrodynamics*, New York, NY, Dover Publications, Inc., 1971.
2. Sutton, George P., *Rocket Propulsion Elements An Introduction to the Engineering of Rockets*, John Wiley & Sons, Inc., 1986.
3. Hale, Francis J., *Introduction to Space Flight*, Prentice-Hall, 1994.
4. Mortenson, M. E., *Geometric Modeling, 2nd Ed.*, John Wiley & Sons, 1997.
5. Nelson, Doug, "A Program to Propagate a Low Thrust Trajectory Using Cowell's Method", 2000.
6. St. Germain, Brad, FORTRAN Low-thrust trajectory code

Appendix A: Source Code

```
/*
 * Interface.java
 *
 * Created on September 14, 2001, 4:07 PM
 */


/**
 *
 * @author  John E. Weglian
 * @version 1.0
 */

import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JComboBox;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;
import javax.swing.JTextField;
import javax.swing.BoxLayout;
import javax.swing.JApplet;
import javax.swing.Timer;
import javax.swing.JSlider;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
import javax.swing.SwingConstants;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.geom.GeneralPath;
import java.awt.geom.AffineTransform;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Shape;

import java.util.ArrayList;

import java.text.NumberFormat;

public class Interface extends JApplet implements
ActionListener
{
    private     JPanel              _orbitParametersPanel
= null;
    private     JPanel              _vectorPanel
= null;
    private     JPanel              _buttonPanel
= null;
    private     JPanel              _sliderPanel
= null;

    private     JFrame              _orbitFrame
= null;
    private     JFrame              _trajectoryFrame
= null;
    private     JFrame              _vehicleFrame
= null;

    private     JLabel              _aLabel
= null;
    private     JLabel              _eLabel
= null;
    private     JLabel              _trueAnomalyLabel
= null;
    private     JLabel              _meanAnomalyLabel
= null;
    private     JLabel              _energyLabel
= null;
    private     JLabel              _hLabel
= null;
    private     JLabel              _raLabel
= null;
    private     JLabel              _rpLabel
= null;
    private     JLabel              _periodLabel
= null;
    private     JLabel              _aorLabel
= null;
    private     JLabel              _eorLabel
= null;
    private     JLabel              _errorMessage
= null;
    private     JLabel              _trueAnomalyValueLabel
= null;
```

```
    private     JLabel              _finalRLabel
= null;
    private     JLabel              _positionLabel
= null;
    private     JLabel              _massLabel
= null;
    private     JLabel              _IspLabel
= null;
    private     JLabel              _defineVehicleErrorMessage
= null;
    private     JLabel              _massValueLabel
= null;
    private     JLabel              _statusLabel
= null;
    private     JLabel              _zoomLabel
= null;
    private     JLabel              _speedLabel
= null;
    private     JLabel              _timeLabel
= null;
    private     JLabel              _velocityLabel
= null;
    private     JLabel              _altitudeLabel
= null;
    private     JLabel              _accelerationLabel
= null;
    private     JLabel              _phiLabel
= null;

    private     JButton             _pauseButton
= null;
    private     JButton             _defineOrbitButton
= null;
    private     JButton             _defineThrustButton
= null;
    private     JButton             _restartButton
= null;
    private     JButton             _defineOrbitOKButton
= null;
    private     JButton             _defineOrbitCancelButton
= null;
    private     JButton             _defineVehicleButton
= null;
    private     JButton             _defineVehicleOKButton
= null;
    private     JButton             _trackButton
= null;
    private     JButton             _smoothButton
= null;

    private     JRadioButton        _aeButton
= null;
    private     JRadioButton        _rarpButton
= null;
    private     JRadioButton        _hahpButton
= null;
    private     JRadioButton        _sunButton
= null;
    private     JRadioButton        _mercuryButton
= null;
    private     JRadioButton        _venusButton
= null;
    private     JRadioButton        _earthButton
= null;
    private     JRadioButton        _marsButton
= null;
    private     JRadioButton        _jupiterButton
= null;
    private     JRadioButton        _saturnButton
= null;
    private     JRadioButton        _uranusButton
= null;
    private     JRadioButton        _neptuneButton
= null;
    private     JRadioButton        _plutoButton
= null;
    private     JRadioButton        _ftButton
= null;
    private     JRadioButton        _miButton
= null;
    private     JRadioButton        _nmButton
= null;
    private     JRadioButton        _mButton
= null;
    private     JRadioButton        _AUButton
= null;
    private     JRadioButton        _kmButton
= null;
    private     JRadioButton        _kgButton
= null;
    private     JRadioButton        _lbmButton
= null;
    private     JRadioButton        _slugButton
= null;
```

```
    private     JTextField          _aorTextField
= null;
    private     JTextField          _eorTextField
= null;
    private     JTextField          _trueAnomalyTextField
= null;
    private     JTextField          _finalRTextField
= null;
    private     JTextField          _massTextField
= null;
    private     JTextField          _IspTextField
= null;

    private     JSlider             _speedSlider
= null;
    private     JSlider             _zoomSlider
= null;

    private     GeneralPath         _trajectoryPath
= null;
    private     GeneralPath         _plotPath
= null;
    private     GeneralPath         _bodyPath
= null;
    private     GeneralPath         _initialOrbitPath
= null;
    private     GeneralPath         _finalOrbitPath
= null;
    private     GeneralPath         _satellite
= null;
    private     GeneralPath         _initBodyPath
= null;
    private     GeneralPath         _initInitialOrbitPath
= null;
    private     GeneralPath         _initFinalOrbitPath
= null;

    private     AffineTransform     _transform
= null;

    private     Timer               _timer
= null;

    private     Orbit               _currentOrbit
= null;

    private     Orbit               _initialOrbit
= null;

    private     Color               _bodyColor
= Color.blue;

    private     NumberFormat        _format
= NumberFormat.getInstance();

    private     int                 _dimension
= 400;
    private     int                 _textBoxLength
= 10;
    private     int                 _body
= 3;
    private     int                 _selection
= 0;
    private     int                 _delay
= 500;
    private     int                 _errorStatus
= 0;

    private     double              _scaleFactor
= 1.0;
    private     double              _aorValue
= 0.0;
    private     double              _eorValue
= 0.0;
    private     double              _factor
= 1.0;
    private     double              _trueAnomalyValue
= 0.0;
    private     double              _finalRValue
= 0.0;
    private     double              _centerX
= 0.0;
    private     double              _centerY
= 0.0;
    private     double              _max
= 0.0;
    private     double              _massFactor
= 1.0;
    private     double              _massValue
= 0.0;
    private     double              _IspValue
= 0.0;
```

```
    private     double              _initialMass
= 0.0;
    private     double              _initialMax
= 0.0;

    private     boolean             _paused
= true;
    private     boolean             _aorSet
= false;
    private     boolean             _eorSet
= false;
    private     boolean             _trueAnomalySet
= false;
    private     boolean             _finalRSet
= false;
    private     boolean             _massSet
= false;
    private     boolean             _IspSet
= false;
    private     boolean             _plotTracks
= true;
    private     boolean             _smooth
= true;

    private     String              _lengthUnitString
= new String (" m");
    private     String              _massUnitString
= new String (" kg");

    private     Trajectory          _trajectory
= null;

    private     PlotPanel           _trajectoryPanel
= null;

    private static final double [] _radius = {696000.0e3,
2487.0e3, 6187.0e3,
    6378.0e3, 3380.0e3, 71370.0e3, 60400.0e3, 23530.0e3,
22320.0e3, 7016.0e3};

    /** Creates new Interface */
    public Interface()
    {
    }

    public Interface (Container contentPane)
    {
        setDefaults();
        _trajectoryPanel = createTrajectoryPanel();
        _orbitParametersPanel = createOrbitParametersPanel ();
        _vectorPanel = createVectorPanel ();
        _buttonPanel = createButtonPanel ();
        _sliderPanel = createSliderPanel ();

        contentPane.setLayout(new BoxLayout(contentPane,
BoxLayout.Y_AXIS));
        contentPane.add(_buttonPanel);
        contentPane.add(_sliderPanel);
        contentPane.add(_orbitParametersPanel);
        contentPane.add(_vectorPanel);
        _statusLabel = new JLabel ("");
        contentPane.add(_statusLabel);
        _trajectoryFrame = new JFrame ("Trajectory Animation");
        _trajectoryPanel.setDoubleBuffered(true);

_trajectoryFrame.getContentPane().add(_trajectoryPanel);
        _trajectoryFrame.setSize(500,500);
        _trajectoryFrame.show();
        setupTimer();
    }

    public void init ()
    {
        Container contentPane = getContentPane();
        setDefaults();
        _trajectoryPanel = createTrajectoryPanel();
        _orbitParametersPanel = createOrbitParametersPanel ();
        _vectorPanel = createVectorPanel ();
        _buttonPanel = createButtonPanel ();
        _sliderPanel = createSliderPanel ();

        contentPane.setLayout(new BoxLayout(contentPane,
BoxLayout.Y_AXIS));
        contentPane.add(_buttonPanel);
        contentPane.add(_sliderPanel);
        contentPane.add(_orbitParametersPanel);
        contentPane.add(_vectorPanel);
        _statusLabel = new JLabel ("");
        contentPane.add(_statusLabel);
        _trajectoryFrame = new JFrame ("Trajectory Animation");
        _trajectoryPanel.setDoubleBuffered(true);

_trajectoryFrame.getContentPane().add(_trajectoryPanel);
        _trajectoryFrame.setSize(500,500);
```

```
        _trajectoryFrame.show();
        setupTimer();
    }

    private void setupTimer()
    {
        _timer = new Timer (_delay, this);
        _timer.setInitialDelay(0);
        _timer.setCoalesce(true);
    }

    private void setDefaults ()
    {
        Interpolation defaultSegment = new Interpolation (true,
true, 0.0, 360.0, 0.0, 0.0, 0.0, 0.0);
        ArrayList segment = new ArrayList();
        segment.add (defaultSegment);
        _trajectory = new Trajectory (3, 6778.0e3, 7000.0e3,
segment);
        _currentOrbit = _trajectory.getOrbit();
        _initialOrbit = _currentOrbit;
        _bodyPath = _trajectory.getBodyPath();
        _initialOrbitPath = _trajectory.getInitialOrbitPath();
        _finalOrbitPath = _trajectory.getFinalOrbitPath();
        _trajectory.initializePath();
        _trajectoryPath = _trajectory.getPath();
        _satellite = _trajectory.getSatellite();
        scale();
        _initialMass = 1000.0;
        _trajectory.setMass (_initialMass);
        _trajectory.setIsp (3000.0);
    }

    private JPanel createSliderPanel()
    {
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(2, 3));
        _zoomLabel = new JLabel("Zoom");

_zoomLabel.setHorizontalAlignment(SwingConstants.CENTER);
        panel.add(_zoomLabel);
        _speedLabel = new JLabel("Speed");

_speedLabel.setHorizontalAlignment(SwingConstants.CENTER);
        panel.add(_speedLabel);
        _trackButton = new JButton ("Tracks off");
        _trackButton.addActionListener (new ActionListener ()
        {
            public void actionPerformed (ActionEvent e)
            {
                if (_plotTracks)
                {
                    _plotTracks = false;
                    _trajectoryPanel.repaint();
                    _trackButton.setText("Tracks on");
                }
                else
                {
                    _plotTracks = true;
                    _trajectoryPanel.repaint();
                    _trackButton.setText("Tracks off");
                }
            }
        });
        panel.add(_trackButton);

        _zoomSlider = new JSlider (0, 100, 50);
        _zoomSlider.addChangeListener (new ChangeListener ()
        {
            public void stateChanged (ChangeEvent e)
            {
                double value =
Math.exp(0.1*(double)_zoomSlider.getValue());

                _scaleFactor = Math.exp(5.0)/value;
                newScale();
            }
        });
        panel.add(_zoomSlider);

        _speedSlider = new JSlider (0, 1000, 500);
        _speedSlider.addChangeListener (new ChangeListener ()
        {
            public void stateChanged (ChangeEvent e)
            {
                int value = _speedSlider.getValue();
                _delay = 1001 - value;
                _timer.setDelay(_delay);
                if (!_smooth)
                    _trajectory.setLimit(value/10);
                else
                {
                    if (value < 250)
                        _trajectory.setLimit(value/10);
```

```
                else
                    _trajectory.setLimit(25);

            }
        }
    });
    panel.add(_speedSlider);

    _smoothButton = new JButton ("Smooth off");
    _smoothButton.addActionListener (new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            if (_smooth)
            {
                _smooth = false;
                int value = _speedSlider.getValue();
                if (value < 250)
                    _trajectory.setLimit(value/10);
                else
                    _trajectory.setLimit(25);
                _trajectoryPanel.repaint();
                _smoothButton.setText("Smooth off");
            }
            else
            {
                _smooth = true;
                int value = _speedSlider.getValue();
                _trajectory.setLimit(value/10);
                _trajectoryPanel.repaint();
                _smoothButton.setText("Smooth on");
            }
        }
    });
    panel.add(_smoothButton);

    return panel;
}

private PlotPanel createTrajectoryPanel ()
{
    PlotPanel panel = new PlotPanel ();
    return panel;
}

private JPanel createOrbitParametersPanel ()
{
        JPanel panel = new JPanel ();
        panel.setLayout(new GridLayout (8,2));
        double r = _currentOrbit.getPosition().getMagnitude();
        double v = _currentOrbit.getVelocity().getMagnitude();
        double a =
_trajectory.getAcceleration().getMagnitude();
        _format.setMaximumFractionDigits(0);
        _energyLabel = new JLabel ("E = "
        +
_format.format(_currentOrbit.getEnergy()/(_factor*_factor))
        + _lengthUnitString + "^2/sec^2");
        _hLabel = new JLabel ("h = "
        +
_format.format(_currentOrbit.getH()/(_factor*_factor))
        + _lengthUnitString + "^2/sec");
        _periodLabel = new JLabel ("Period = "
        +  _format.format(_currentOrbit.getPeriod()) + " sec");
        _timeLabel = new JLabel ("Elapsed time = "
        + _format.format(_trajectory.getTime()) + " sec");
        _format.setMaximumFractionDigits(1);
        _raLabel = new JLabel ("ra = "
        +  _format.format(_currentOrbit.getRa()/_factor)
        + _lengthUnitString);
        _rpLabel = new JLabel ("rp = "
        +  _format.format(_currentOrbit.getRp()/_factor)
        + _lengthUnitString);
        _aLabel = new JLabel ("a = "
        + _format.format(_currentOrbit.getA()/_factor)
        + _lengthUnitString);
        _massLabel = new JLabel("Mass = "
        +  _format.format(_trajectory.getMass()/_massFactor)
        + _massUnitString);
        _positionLabel = new JLabel("r = "
        +  _format.format(r/_factor) + _lengthUnitString);
        _velocityLabel = new JLabel("v = "
        + _format.format(v/_factor)
        + _lengthUnitString + "/sec");
        _altitudeLabel = new JLabel ("alt = "
        + _format.format((r-_radius[_body])/_factor)
        + _lengthUnitString);

        _format.setMaximumFractionDigits(2);
        _eLabel = new JLabel ("e = "
        + _format.format(_currentOrbit.getE()));
        _phiLabel = new JLabel ("phi = "
        + _format.format(_currentOrbit.getPhi())
        + " deg");
```

```
        _trueAnomalyLabel = new JLabel ("True Anomaly = "
        + _format.format(_currentOrbit.getTrueAnomaly())
        + " deg");
        _meanAnomalyLabel = new JLabel ("Mean Anomaly = "
        + _format.format(_currentOrbit.getMeanAnomaly())
        + " deg");
        _format.setMaximumFractionDigits(4);
        _accelerationLabel = new JLabel ("Acceleration = "
        +_format.format(a) + " m/sec^2");
        _phiLabel = new JLabel("Phi = "
        +_format.format(Math.toDegrees(_currentOrbit.getPhi()))
        + " deg");

        panel.add(_aLabel);
        panel.add(_eLabel);
        panel.add(_energyLabel);
        panel.add(_hLabel);
        panel.add(_periodLabel);
        panel.add(_timeLabel);
        panel.add(_raLabel);
        panel.add(_rpLabel);
        panel.add(_trueAnomalyLabel);
        panel.add(_meanAnomalyLabel);
        panel.add(_positionLabel);
        panel.add(_altitudeLabel);
        panel.add(_velocityLabel);
        panel.add(_massLabel);
        panel.add(_accelerationLabel);
        panel.add(_phiLabel);
        return panel;
    }

    public void updateOrbit ()
    {
        _currentOrbit = _trajectory.getOrbit();
        double r = _currentOrbit.getPosition().getMagnitude();
        double v = _currentOrbit.getVelocity().getMagnitude();
        double a =
_trajectory.getAcceleration().getMagnitude();
        _format.setMaximumFractionDigits(0);
        _energyLabel.setText ("E = "
        +
_format.format(_currentOrbit.getEnergy()/(_factor*_factor))
        + _lengthUnitString + "^2/sec^2");
        _hLabel.setText ("h = "
        +
_format.format(_currentOrbit.getH()/(_factor*_factor))
        + _lengthUnitString + "^2/sec");
        _periodLabel.setText ("Period = "
        + _format.format(_currentOrbit.getPeriod()) + " sec");
        _timeLabel.setText ("Elapsed time = "
        +_format.format(_trajectory.getTime())
        + " sec");
        _format.setMaximumFractionDigits(1);
        _aLabel.setText ("a = "
        + _format.format(_currentOrbit.getA()/_factor)
        + _lengthUnitString);
        _raLabel.setText ("ra = "
        + _format.format(_currentOrbit.getRa()/_factor)
        + _lengthUnitString);
        _rpLabel.setText ("rp = "
        + _format.format(_currentOrbit.getRp()/_factor)
        + _lengthUnitString);
        _positionLabel.setText("r = "
        + _format.format(r/_factor)
        + _lengthUnitString);
        _massLabel.setText("Mass = "
        + _format.format(_trajectory.getMass()/_massFactor)
        + _massUnitString);
        _velocityLabel.setText("v = "
        + _format.format(v/_factor)
        + _lengthUnitString + "/sec");
        _altitudeLabel.setText ("alt = "
        + _format.format((r-_radius[_body])/_factor)
        + _lengthUnitString);
        _format.setMaximumFractionDigits(2);
        _eLabel.setText ("e = " +
_format.format(_currentOrbit.getE()));
        _phiLabel.setText ("phi = " +
_format.format(_currentOrbit.getPhi())
        + " deg");
        _trueAnomalyLabel.setText ("True Anomaly = "
        + _format.format(_currentOrbit.getTrueAnomaly())
        + " deg");
        _meanAnomalyLabel.setText ("Mean Anomaly = "
        + _format.format(_currentOrbit.getMeanAnomaly())
        + " deg");
        _format.setMaximumFractionDigits(4);
        _accelerationLabel.setText("Acceleration = "
        +_format.format(a) + " m/sec^2");
        _phiLabel.setText("Phi = "
        +_format.format(Math.toDegrees(_currentOrbit.getPhi()))
        + " deg");
```

```
    }

    private JPanel createVectorPanel ()
    {
        JPanel panel = new JPanel ();


        return panel;
    }

    private JPanel createButtonPanel ()
    {
        JPanel panel = new JPanel ();
        panel.setLayout(new GridLayout(1,4));
        _pauseButton = new JButton ("Start");
        _pauseButton.addActionListener (new ActionListener ()
        {
            public void actionPerformed (ActionEvent e)
            {
                pause();
            }
        });
        panel.add(_pauseButton);

        _defineOrbitButton = new JButton ("Orbit");
        _defineOrbitButton.setToolTipText("Define the initial
orbital parameters");
        _defineOrbitButton.addActionListener (new
ActionListener ()
        {
            public void actionPerformed (ActionEvent e)
            {
                if (!_paused)
                    pause();
                defineOrbit();
            }
        });
        panel.add(_defineOrbitButton);

        _defineThrustButton = new JButton ("Thrust");
        _defineThrustButton.setToolTipText("Define the thrust
profile");
        _defineThrustButton.addActionListener (new
ActionListener ()
        {
            public void actionPerformed (ActionEvent e)
            {
                if (!_paused)
                    pause();
                SetUp newSetUp = new SetUp ();
                newSetUp.setUpSegments(_trajectory, 0.0);
            }
        });
        panel.add(_defineThrustButton);

        _defineVehicleButton = new JButton ("Vehicle");
        _defineVehicleButton.setToolTipText("Define the vehicle
parameters");
        _defineVehicleButton.addActionListener (new
ActionListener ()
        {
            public void actionPerformed (ActionEvent e)
            {
                if (!_paused)
                    pause();
                defineVehicle();
            }
        });
        panel.add(_defineVehicleButton);

        _restartButton = new JButton ("Reset");
        _restartButton.addActionListener (new ActionListener ()
        {
            public void actionPerformed (ActionEvent e)
            {
                if (!_paused)
                    pause();
                restart();
            }
        });
        panel.add(_restartButton);
        return panel;
    }

    private void defineVehicle()
    {
        _vehicleFrame = new JFrame ("Define Vehicle
Parameters");
        JPanel panel = new JPanel();
        panel.setLayout (new BoxLayout(panel,
BoxLayout.Y_AXIS));

        _kgButton = new JRadioButton ("kilograms", true);
        _kgButton.addActionListener (new ActionListener()
```

```
        {
            public void actionPerformed(ActionEvent event)
            {
                if (_massSet)
                    _massValue*=1.0/_factor;
                _massFactor = 1.0;
                _massUnitString = new String (" kg");
            }
        });
        _lbmButton = new JRadioButton ("lbm", false);
        _lbmButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                if (_massSet)
                    _massValue*=0.4536/_factor;
                _massFactor = 0.4536;
                _massUnitString = new String (" lbm");
            }
        });
        _slugButton = new JRadioButton ("slugs", false);
        _slugButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                if (_massSet)
                    _massValue*=14.60592/_factor;
                _massFactor = 14.60592;
                _massUnitString = new String (" slugs");
            }
        });
        ButtonGroup group = new ButtonGroup ();
        group.add(_kgButton);
        group.add(_lbmButton);
        group.add(_slugButton);
        panel.add(_kgButton);
        panel.add(_lbmButton);
        panel.add(_slugButton);

        _massValueLabel = new JLabel ("Enter mass of
Vehicle:");

        panel.add(_massValueLabel);

        _massTextField = new JTextField(_textBoxLength);
        _massTextField.setEditable (true);
        _massSet = false;
        _massTextField.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    String testString =
_massTextField.getText();
                    _massValue =
_massFactor*Double.valueOf(testString).doubleValue();
                    if (_massValue < 0.0)
                    {

_defineVehicleErrorMessage.setText("Mass must be greater than
0");
                        _massSet = false;
                        _massTextField.setText("");
                    }
                    else
                    {
                        _massSet = true;
                        _defineVehicleErrorMessage.setText("");
                        _IspTextField.requestFocus();
                        checkDefineVehicleOKStatus ();
                    }
                }
                catch (NumberFormatException error)
                {
                    _defineVehicleErrorMessage.setText("Enter a
number");
                    _massSet = false;
                }
            }
        });
        panel.add(_massTextField);

        _IspLabel = new JLabel ("Isp");
        panel.add(_IspLabel);

        _IspTextField = new JTextField(_textBoxLength);
        _IspTextField.setEditable (true);
        _IspSet = false;
        _IspTextField.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                try
```

```
                {
                    String testString =
_IspTextField.getText();
                    _IspValue =
Double.valueOf(testString).doubleValue();
                    _IspSet = true;
                    _defineVehicleErrorMessage.setText("");
                    _massTextField.requestFocus();
                    if (_IspValue < 0.0)
                    {
                        _IspSet = false;
                        _IspTextField.setText("");
                        _defineVehicleErrorMessage.setText("Isp
must be > 0");
                    }
                    else
                        checkDefineVehicleOKStatus ();

                }
                catch (NumberFormatException error)
                {
                    _defineVehicleErrorMessage.setText("Enter a
number");
                    _IspSet = false;
                }
            }
        });
        panel.add(_IspTextField);

        _defineVehicleErrorMessage = new JLabel ("");
        panel.add (_defineVehicleErrorMessage);

        _defineVehicleOKButton = new JButton ("OK");
        _defineVehicleOKButton.setEnabled (true);
        _defineVehicleOKButton.addActionListener (new
ActionListener ()
        {
            public void actionPerformed (ActionEvent e)
            {
                _massTextField.postActionEvent();
                _IspTextField.postActionEvent();
                if (_massSet && _IspSet)
                {
                    _initialMass = _massValue;
                    _trajectory.setMass(_initialMass);
                    _trajectory.setIsp(_IspValue);
                    _vehicleFrame.dispose();
                    updateOrbit();
                }
            }
        });
        panel.add(_defineVehicleOKButton);

        _vehicleFrame.getContentPane().add(panel);
        _vehicleFrame.pack();
        _vehicleFrame.show();

    }

    private void pause()
    {
        if (_paused)
        {
            _paused = false;
            _pauseButton.setText("Pause");
            startAnimation();
        }
        else
        {
            _paused = true;
            _pauseButton.setText("Start");
            stopAnimation();
        }
    }
    private void defineOrbit()
    {
        _orbitFrame = new JFrame ("Define Orbital Elements");
        JPanel bodyPanel = getBodyPanel ();
        JPanel unitPanel = getUnitPanel ();
        JPanel elementPanel = getElementPanel();
        Container contentPane = _orbitFrame.getContentPane();
        contentPane.setLayout( new GridLayout(1,3));

        contentPane.add(bodyPanel);
        contentPane.add (unitPanel);
        contentPane.add (elementPanel);
        _orbitFrame.pack();
        _orbitFrame.show();
    }

    private JPanel getElementPanel ()
    {
        JPanel panel = new JPanel ();
```

```
        panel.setLayout (new BoxLayout(panel,
BoxLayout.Y_AXIS));

        _aorSet = false;
        _eorSet = false;
        _trueAnomalySet = false;
        _finalRSet = false;
        _selection = 0;
        _aeButton = new JRadioButton ("a and e", true);
        _aeButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                _selection = 0;
                _aorTextField.setText ("");
                _eorTextField.setText ("");
                _aorSet = false;
                _eorSet = false;
                _aorLabel.setText("a");
                _eorLabel.setText("e");
                _finalRLabel.setText("Final r");
            }
        });
        _rarpButton = new JRadioButton ("ra and rp", false);
        _rarpButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                _selection = 1;
                _aorSet = false;
                _eorSet = false;
                _aorLabel.setText("ra");
                _eorLabel.setText("rp");
                _finalRLabel.setText("Final r");
            }
        });
        _hahpButton = new JRadioButton ("ha and hp", false);
        _hahpButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                _selection = 2;
                _aorSet = false;
                _eorSet = false;
                _aorLabel.setText("ha");
                _eorLabel.setText("hp");
                _finalRLabel.setText("Final h");
            }
        });
        ButtonGroup group = new ButtonGroup ();
        group.add (_aeButton);
        group.add (_rarpButton);
        group.add (_hahpButton);
        panel.add (_aeButton);
        panel.add (_rarpButton);
        panel.add (_hahpButton);
        _aorLabel = new JLabel ("a");
        _eorLabel = new JLabel ("e");
        _aorTextField = new JTextField (_textBoxLength);
        _aorTextField.setEditable (true);
        _aorTextField.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    String testString =
_aorTextField.getText();
                    _aorValue = _factor *
Double.valueOf(testString).doubleValue();
                    _errorMessage.setText("");
                    _aorSet = true;
                    _eorTextField.requestFocus();
                    if (_selection == 0)
                    {
                        if (_aorValue > 0.0 && _aorValue <
_radius[_body])
                        {
                            _aorSet = false;
                            _errorMessage.setText("a must be >
" + _radius[_body]/_factor);

                            _aorTextField.requestFocus();
                        }
                        if (_eorSet)
                        {
                            if (_eorValue > 1.0 && _aorValue >
0.0)
                            {
                                _aorSet = false;
                                _errorMessage.setText ("a must
be negative if e > 1");
```

```
                            _aorTextField.requestFocus();
                        }
                        else if (_eorValue < 1.0 &&
_aorValue < 0.0)
                        {
                            _aorSet = false;
                            _errorMessage.setText ("a must
be positive if e < 1");
                            _aorTextField.requestFocus();
                        }
                    }
                }
                else if (_selection == 1)
                {
                    if (_aorValue < _radius [_body])
                    {
                        _aorSet = false;
                        _errorMessage.setText ("ra must be
> " + _radius[_body]/_factor);
                        _aorTextField.requestFocus();
                    }
                    if (_eorSet && _aorValue < _eorValue)
                    {
                        _aorSet = false;
                        _errorMessage.setText("ra must be >
rp");
                    }
                }
                else
                {
                    if (_aorValue < 0.0)
                    {
                        _aorSet = false;
                        _errorMessage.setText("ha must be >
0");
                        _aorTextField.requestFocus();
                    }
                    if (_eorSet && _aorValue < _eorValue)
                    {
                        _aorSet = false;
                        _errorMessage.setText("ha must be >
hp");
                        _aorTextField.requestFocus();
                    }
                }
                checkOKStatus ();
            }
                catch (NumberFormatException error)
                {
                    _errorMessage.setText("Enter a number");
                    _aorSet = false;
                }
            }
        });
        panel.add(_aorLabel);
        panel.add(_aorTextField);

        _eorTextField = new JTextField (_textBoxLength);
        _eorTextField.setEditable (true);
        _eorTextField.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    String testString =
_eorTextField.getText();
                    _eorValue =
Double.valueOf(testString).doubleValue();
                    _errorMessage.setText("");
                    _eorSet = true;
                    _trueAnomalyTextField.requestFocus();
                    if (_selection == 0)
                    {
                        if (_eorValue < 0.0)
                        {
                            _eorSet = false;
                            _errorMessage.setText("e must be >
0");
                            _eorTextField.requestFocus();
                        }
                        if (_aorSet)
                        {
                            if (_aorValue > 0.0 && _eorValue >
1.0)
                            {
                                _eorSet = false;
                                _errorMessage.setText ("e must
be > 1 if a < 0");
                                _eorTextField.requestFocus();
                            }
                            else if (_eorValue < 1.0 &&
_aorValue < 0.0)
                            {
```

```
                                _eorSet = false;
                                _errorMessage.setText ("e must
be < 1 if a > 0");
                                _eorTextField.requestFocus();
                            }
                        }
                    }
                    else if (_selection == 1)
                    {
                        _eorValue *= _factor;
                        if (_eorValue < _radius [_body])
                        {
                            _eorSet = false;
                            _errorMessage.setText ("rp must be
> " + _radius[_body]/_factor);
                            _eorTextField.requestFocus();
                        }
                        if (_aorSet && _aorValue < _eorValue)
                        {
                            _eorSet = false;
                            _errorMessage.setText("rp must be <
ra");
                            _eorTextField.requestFocus();
                        }
                    }
                    else
                    {
                        _eorValue *= _factor;
                        if (_eorValue < 0.0)
                        {
                            _eorSet = false;
                            _errorMessage.setText("hp must be >
0");
                            _eorTextField.requestFocus();
                        }
                        if (_aorSet && _aorValue < _eorValue)
                        {
                            _eorSet = false;
                            _errorMessage.setText("hp must be <
ha");
                            _eorTextField.requestFocus();
                        }
                    }
                    checkOKStatus ();
                }
                catch (NumberFormatException error)
                {
                    _errorMessage.setText("Enter a number");
                    _eorSet = false;
                }
            }
        });
        panel.add(_eorLabel);
        panel.add(_eorTextField);

        _trueAnomalyValueLabel = new JLabel ("True Anomaly in
degrees");
        panel.add(_trueAnomalyValueLabel);

        _trueAnomalyTextField = new JTextField(_textBoxLength);
        _trueAnomalyTextField.setEditable (true);
        _trueAnomalyTextField.setText("0.0");
        _trueAnomalyValue = 0.0;
        _trueAnomalySet = true;
        _trueAnomalyTextField.addActionListener(new
ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    String testString =
_trueAnomalyTextField.getText();
                    _trueAnomalyValue =
Double.valueOf(testString).doubleValue();
                    while (_trueAnomalyValue < 0.0)
                        _trueAnomalyValue += 360.0;
                    while (_trueAnomalyValue > 360.0)
                        _trueAnomalyValue -= 360.0;
                    _trueAnomalySet = true;
                    _errorMessage.setText("");
                    _finalRTextField.requestFocus();
                    checkOKStatus ();
                }
                catch (NumberFormatException error)
                {
                    _errorMessage.setText("Enter a number");
                    _trueAnomalySet = false;
                }
            }
        });
        panel.add(_trueAnomalyTextField);

        _finalRLabel = new JLabel ("Final r");
```

```
        panel.add(_finalRLabel);

        _finalRTextField = new JTextField(_textBoxLength);
        _finalRTextField.setEditable (true);
        _finalRSet = false;
        _finalRTextField.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    String testString =
_finalRTextField.getText();
                    _finalRValue = _factor *
Double.valueOf(testString).doubleValue();
                    _finalRSet = true;
                    _errorMessage.setText("");
                    if (_selection !=2 && _finalRValue <
_radius[_body])
                    {
                        _finalRSet = false;
                        _errorMessage.setText("Final r must be
> " + _radius[_body]/_factor);
                    }
                    else if (_selection == 2 && _finalRValue <
0.0)
                    {
                        _finalRSet = false;
                        _errorMessage.setText("Final h must be
> 0");
                    }
                    else
                        checkOKStatus ();

                }
                catch (NumberFormatException error)
                {
                    _errorMessage.setText("Enter a number");
                    _finalRSet = false;
                }
            }
        });
        panel.add(_finalRTextField);

        _errorMessage = new JLabel ("");
        panel.add (_errorMessage);
        return panel;
    }

    private JPanel getUnitPanel ()
    {
        JPanel panel = new JPanel ();
        _factor = 1.0;
        panel.setLayout (new BoxLayout(panel,
BoxLayout.Y_AXIS));
        _AUButton = new JRadioButton ("AU", false);
        _AUButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                reset(1.4959965e11/_factor);
                _factor = 1.4959965e11;
                _lengthUnitString = new String (" AU");
            }
        });
        _ftButton = new JRadioButton ("feet", false);
        _ftButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                reset(0.3048/_factor);
                _factor = 0.3048;
                _lengthUnitString = new String (" ft");
            }
        });
        _miButton = new JRadioButton ("miles", false);
        _miButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                reset(1609.3/_factor);
                _factor = 1609.3;
                _lengthUnitString = new String (" mi");
            }
        });
        _nmButton = new JRadioButton ("nautical miles", false);
        _nmButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                reset(1852.0/_factor);
                _factor = 1852.0;
                _lengthUnitString = new String (" nm");
            }
```

```
        });
        _mButton = new JRadioButton ("meters", true);
        _mButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                reset(1.0/_factor);
                _factor = 1.0;
                _lengthUnitString = new String (" m");
            }
        });
        _kmButton = new JRadioButton ("kilometers", false);
        _kmButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                reset(1000.0/_factor);
                _factor = 1000.0;
                _lengthUnitString = new String (" km");
            }
        });
        ButtonGroup group = new ButtonGroup();
        group.add(_AUButton);
        group.add(_ftButton);
        group.add(_miButton);
        group.add(_nmButton);
        group.add(_mButton);
        group.add(_kmButton);

        panel.add(_AUButton);
        panel.add(_ftButton);
        panel.add(_miButton);
        panel.add(_nmButton);
        panel.add(_mButton);
        panel.add(_kmButton);

        JLabel temp = new JLabel ("");
        panel.add(temp);

        _defineOrbitOKButton = new JButton ("OK");
        _defineOrbitOKButton.setEnabled (true);
        _defineOrbitOKButton.addActionListener (new
ActionListener ()
        {
            public void actionPerformed (ActionEvent e)
            {
                _aorTextField.postActionEvent();
                _eorTextField.postActionEvent();
                _trueAnomalyTextField.postActionEvent();
                _finalRTextField.postActionEvent();
                if (_aorSet && _eorSet && _trueAnomalySet &&
_finalRSet)
                    finalizeOrbit ();
            }
        });
        panel.add(_defineOrbitOKButton);
        _defineOrbitCancelButton = new JButton ("Cancel");
        _defineOrbitCancelButton.addActionListener (new
ActionListener ()
        {
            public void actionPerformed (ActionEvent e)
            {
                _orbitFrame.dispose ();
            }
        });
        panel.add(_defineOrbitCancelButton);

        return panel;

    }

    private void finalizeOrbit ()
    {
        double a, e;
        if (_selection == 0)
        {
            _initialOrbit = new Orbit (_aorValue, _eorValue,
Math.toRadians(_trueAnomalyValue), _body);
        }
        else if (_selection == 1)
        {
            a = 0.5*(_aorValue + _eorValue);
            e = (_aorValue - _eorValue)/(_aorValue +
_eorValue);

            _initialOrbit = new Orbit (a, e,
Math.toRadians(_trueAnomalyValue), _body);
        }
        else
        {
            double ra = _aorValue + _radius[_body];
            double rp = _eorValue + _radius[_body];
            _finalRValue += _radius[_body];
            a = 0.5*(ra + rp);
```

```
            e = (ra - rp)/(ra + rp);
            _initialOrbit = new Orbit (a, e,
Math.toRadians(_trueAnomalyValue), _body);
        }
        _trajectory.setR2(_finalRValue);
        _trajectory.setOrbit (_initialOrbit);
        _bodyPath = _trajectory.getBodyPath();
        _initialOrbitPath = _trajectory.getInitialOrbitPath();
        _finalOrbitPath = _trajectory.getFinalOrbitPath();
        _trajectory.initializePath();
        _trajectoryPath = _trajectory.getPath();
        _satellite = _trajectory.getSatellite();
        scale();
        updateOrbit();
        _trajectoryPanel.repaint();
        _statusLabel.setText("");
        _orbitFrame.dispose();
    }

    private void reset(double value)
    {
        if (_aorSet)
            _aorValue *= value;
        if (_selection != 0 && _eorSet)
            _eorValue *= value;
        if (_finalRSet)
            _finalRValue *= _factor;
    }

    private void checkOKStatus ()
    {
        if (_aorSet && _eorSet && _trueAnomalySet &&
_finalRSet)
        {
            _defineOrbitOKButton.setEnabled (true);
            _defineOrbitOKButton.requestFocus();
        }
        return;
    }

    private void checkDefineVehicleOKStatus ()
    {
        if (_massSet && _IspSet)
        {
            _defineVehicleOKButton.requestFocus();
        }
        return;
    }

    private JPanel getBodyPanel ()
    {
        JPanel panel = new JPanel ();
        panel.setLayout (new BoxLayout(panel,
BoxLayout.Y_AXIS));
        _body = 3;
        _bodyColor = Color.blue;
        _sunButton = new JRadioButton ("Sun", false);
        _sunButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                _body = 0;
                _bodyColor = Color.yellow;
                checkValues ();
            }
        });
        _mercuryButton = new JRadioButton ("Mercury", false);
        _mercuryButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                _body = 1;
                _bodyColor = Color.gray;
                checkValues ();
            }
        });

        _venusButton = new JRadioButton ("Venus", false);
        _venusButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                _body = 2;
                _bodyColor = Color.magenta;
                checkValues ();
            }
        });
        _earthButton = new JRadioButton ("Earth", true);
        _earthButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                _body = 3;
                _bodyColor = Color.blue;
```

```
        checkValues ();
    }
});
_marsButton = new JRadioButton ("Mars", false);
_marsButton.addActionListener (new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        _body = 4;
        _bodyColor = Color.red;
        checkValues ();
    }
});
_jupiterButton = new JRadioButton ("Jupiter", false);
_jupiterButton.addActionListener (new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        _body = 5;
        _bodyColor = Color.orange;
        checkValues ();
    }
});
_saturnButton = new JRadioButton ("Saturn", false);
_saturnButton.addActionListener (new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        _body = 6;
        _bodyColor = Color.cyan;
        checkValues ();
    }
});
_uranusButton = new JRadioButton ("Uranus", false);
_uranusButton.addActionListener (new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        _body = 7;
        _bodyColor = Color.cyan;
        checkValues ();
    }
});
_neptuneButton = new JRadioButton ("Neptune", false);
_neptuneButton.addActionListener (new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        _body = 8;
        _bodyColor = Color.cyan;
        checkValues ();
    }
});
_plutoButton = new JRadioButton ("Pluto", false);
_plutoButton.addActionListener (new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        _body = 9;
        _bodyColor = Color.lightGray;
        checkValues ();
    }
});
ButtonGroup group = new ButtonGroup ();
group.add(_sunButton);
group.add(_mercuryButton);
group.add(_venusButton);
group.add(_earthButton);
group.add(_marsButton);
group.add(_jupiterButton);
group.add(_saturnButton);
group.add(_uranusButton);
group.add(_neptuneButton);
group.add(_plutoButton);
panel.add(_sunButton);
panel.add(_mercuryButton);
panel.add(_venusButton);
panel.add(_earthButton);
panel.add(_marsButton);
panel.add(_jupiterButton);
panel.add(_saturnButton);
panel.add(_uranusButton);
panel.add(_neptuneButton);
panel.add(_plutoButton);
return panel;
}

private void checkValues ()
{
    if (_aorSet)
    {
        if (_selection == 0)
        {
```

```
                if (_aorValue > 0.0 && _aorValue <
_radius[_body])
                {
                    _aorSet = false;
                    _errorMessage.setText("a must be > " +
_radius[_body]/_factor);
                    _aorTextField.setText("");
                    _aorTextField.requestFocus();
                }

            }
            else if (_selection == 1)
            {
                if (_aorValue < _radius [_body])
                {
                    _aorSet = false;
                    _errorMessage.setText ("ra must be > " +
_radius[_body]/_factor);
                    _aorTextField.setText("");
                    _aorTextField.requestFocus();
                }

            }

        }
        if (_eorSet && _selection == 1)
        {
            if (_eorValue < _radius [_body])
            {
                _eorSet = false;
                _errorMessage.setText ("rp must be > " +
_radius[_body]/_factor);
                _eorTextField.setText("");
                _eorTextField.requestFocus();
            }

        }
        if (_finalRSet)
        {
            if (_finalRValue < _radius[_body])
            {
                _finalRSet = false;
                _finalRTextField.setText("");
                _errorMessage.setText("Final r must be > " +
_radius[_body]/_factor);
            }
```

```
        }
        _trajectory.setBody(_body);
        _initBodyPath = _trajectory.getBodyPath();
    }

    private void restart ()
    {
        _trajectory.setMass(_initialMass);
        _trajectory.setOrbit (_initialOrbit);
        _trajectory.initializePath();
        _bodyPath = _trajectory.getBodyPath();
        _initialOrbitPath = _trajectory.getInitialOrbitPath();
        _finalOrbitPath = _trajectory.getFinalOrbitPath();
        _trajectoryPath = _trajectory.getPath();
        _satellite = _trajectory.getSatellite();
        scale();
        _trajectoryPanel.repaint();
        updateOrbit();
        _statusLabel.setText("");

    }


    public void setTrajectory (GeneralPath newPath)
    {
        _trajectoryPath = newPath;
        scale();
    }

    private void newScale()
    {
        _bodyPath = (GeneralPath)_initBodyPath.clone();
        _initialOrbitPath
=(GeneralPath)_initInitialOrbitPath.clone();
        _finalOrbitPath =
(GeneralPath)_initFinalOrbitPath.clone();
        _plotPath = (GeneralPath)_trajectoryPath.clone();
        _max = _initialMax*_scaleFactor;

        double scale = ((double)_dimension)/_max;

        _transform = new AffineTransform();
        _transform.setToIdentity();
        _transform.scale(1.0, -1.0);
        _bodyPath.transform(_transform);
        _plotPath.transform(_transform);
        _initialOrbitPath.transform(_transform);
```

```
        _finalOrbitPath.transform(_transform);
        _satellite.transform(_transform);

        _transform.setToIdentity();
        _transform.translate(-_centerX, -_centerY);
        _bodyPath.transform(_transform);
        _plotPath.transform(_transform);
        _initialOrbitPath.transform(_transform);
        _finalOrbitPath.transform(_transform);
        _satellite.transform(_transform);

        _transform.setToIdentity();
        _transform.scale(scale, scale);
        _plotPath.transform(_transform);
        _bodyPath.transform(_transform);
        _initialOrbitPath.transform(_transform);
        _finalOrbitPath.transform(_transform);
        _satellite.transform(_transform);

        _transform.setToIdentity();
        _transform.translate(_max*scale*0.5, _max*scale*0.5);
        _plotPath.transform(_transform);
        _bodyPath.transform(_transform);
        _initialOrbitPath.transform(_transform);
        _finalOrbitPath.transform(_transform);
        _satellite.transform(_transform);

    }

    private void scale()
    {
        GeneralPath testPath = new
GeneralPath(_trajectoryPath);
        testPath.append(_bodyPath, false);
        testPath.append(_initialOrbitPath, false);
        testPath.append(_finalOrbitPath, false);
        _initBodyPath = (GeneralPath)_bodyPath.clone();
        _initInitialOrbitPath =
(GeneralPath)_initialOrbitPath.clone();
        _initFinalOrbitPath =
(GeneralPath)_finalOrbitPath.clone();
        double width = testPath.getBounds2D().getWidth();
        double height = testPath.getBounds2D().getHeight();
        _max = width;
        if (height > _max)
            _max = height;
        _initialMax = _max;
        _centerX = testPath.getBounds2D().getCenterX();
        _centerY = testPath.getBounds2D().getCenterY();
        double scale = ((double)_dimension)/_max;

        _plotPath = _trajectoryPath;
        _transform = new AffineTransform();

        _transform.setToIdentity();
        _transform.scale(1.0, -1.0);
        _bodyPath.transform(_transform);
        _plotPath.transform(_transform);
        _initialOrbitPath.transform(_transform);
        _finalOrbitPath.transform(_transform);
        _satellite.transform(_transform);

        _transform.setToIdentity();
        _transform.translate(-_centerX, -_centerY);
        _bodyPath.transform(_transform);
        _plotPath.transform(_transform);
        _initialOrbitPath.transform(_transform);
        _finalOrbitPath.transform(_transform);
        _satellite.transform(_transform);

        _transform.setToIdentity();
        _transform.scale(scale, scale);
        _plotPath.transform(_transform);
        _bodyPath.transform(_transform);
        _initialOrbitPath.transform(_transform);
        _finalOrbitPath.transform(_transform);
        _satellite.transform(_transform);

        _transform.setToIdentity();
        _transform.translate(_max*scale*0.5, _max*scale*0.5);
        _plotPath.transform(_transform);
        _bodyPath.transform(_transform);
        _initialOrbitPath.transform(_transform);
        _finalOrbitPath.transform(_transform);
        _satellite.transform(_transform);
    }

    private void rescale ()
    {
        _plotPath = (GeneralPath)_trajectoryPath.clone();
        _transform = new AffineTransform();

        _transform.setToIdentity();
        _transform.scale(1.0, -1.0);
```

```
    _plotPath.transform(_transform);
    _satellite.transform(_transform);

    _transform.setToIdentity();
    _transform.translate(-_centerX, -_centerY);
    _plotPath.transform(_transform);
    _satellite.transform(_transform);

    double scale = ((double)_dimension)/_max;
    _transform.setToIdentity();
    _transform.scale(scale, scale);
    _plotPath.transform(_transform);
    _satellite.transform(_transform);

    _transform.setToIdentity();
    _transform.translate(_max*scale*0.5, _max*scale*0.5);
    _plotPath.transform(_transform);
    _satellite.transform(_transform);

}

public void actionPerformed(ActionEvent p1)
{
    _errorStatus = _trajectory.update();
    _trajectoryPath = _trajectory.getPath();
    _satellite = _trajectory.getSatellite();
    updateOrbit ();
    drawOrbit ();
    drawDiagram ();
    if (_errorStatus == 0)
    {
        return;
    }
    else if (_errorStatus == 1)
    {
        pause();
        _statusLabel.setText("Final distance Reached");
    }
    else if (_errorStatus == 2)
    {
        pause();
        _statusLabel.setText("Impacted surface");
    }

    else if (_errorStatus == 3)
    {
        pause();
        _statusLabel.setText("Exhausted all mass");
    }
}

private void drawOrbit()
{
    rescale();
    _trajectoryPanel.repaint();
}

private void drawDiagram ()
{


}

public void start()
{
    startAnimation ();
}

public void stop ()
{
    stopAnimation();
}

public synchronized void startAnimation ()
{
    if (!_paused)
    {
        if (!_timer.isRunning())
            _timer.start();
    }
}

public synchronized void stopAnimation ()
{
    if (_timer.isRunning())
    {
        _timer.stop();
    }
}



class PlotPanel extends JPanel
{
```

```
        public PlotPanel ()
        {
            super ();
        }

        public void paintComponent (Graphics g)
        {
            super.paintComponent(g);
            Graphics2D g2 = (Graphics2D) g;
            if (_plotTracks)
            {
                g2.setPaint(Color.black);
                g2.draw(_plotPath);
            }
            g2.setPaint(Color.black);
            g2.draw(_satellite);
            g2.setPaint(_bodyColor);
            g2.draw(_bodyPath);
            g2.fill((Shape)_bodyPath);
            g2.setPaint(Color.green);
            g2.draw(_initialOrbitPath);
            g2.setPaint(Color.red);
            g2.draw(_finalOrbitPath);
        }
    }// class PlotPanel


} //Interface class
```

```
/*
 * Trajectory.java
 *
 * Created on April 27, 2001, 12:30 PM
 */


/**
 *
 * @author  John E. Weglian
 * @version 1.0
 */

import java.util.ArrayList;
import javax.swing.JOptionPane;
import java.awt.geom.Point2D;
import java.awt.geom.GeneralPath;
import java.awt.Container;
import javax.swing.JApplet;
import java.awt.geom.Ellipse2D;
import java.awt.Shape;
import javax.swing.JFrame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class Trajectory
{
    private MyVector                    _position;
    private MyVector                    _velocity;
    private MyVector                    _acceleration;
    private MyVector                    _thrustVector
= null;

    private double                      _initialR;
    private double                      _finalR;
    private double                      _direction      = 1.0;
    private double                      _delta          =
0.001;
    private double                      _mass;
    private double                      _Isp;
    private double                      _thrust         = 0.0;
    private double                      _time           = 0.0;
    private double                      _vEsc;
    private double                      _TOL;
    private double                      _r;

    private int                         _body;
    private int                         _limit          = 50;
    private int                         _errorStatus    = 0;

    private final double                _g0             = 9.81;
    private final double                _secondsPerDay =
86400.0;

    private Orbit                       _currentOrbit   = null;
    private ArrayList                   _segment        = null;
    private int                         _interpolationCount =
1;
    private int                         _currentInterpolation =
0;
    private Interpolation               _currentSegment = null;

    private GeneralPath                 _path           = null;
    private MyGeneralPath               _myPath         = null;
```

```
    private double                      _tolFactor      = 1e-8;
    private static final double [] _radius = {696000.0e3,
2487.0e3, 6187.0e3,
    6378.0e3, 3380.0e3, 71370.0e3, 60400.0e3, 23530.0e3,
22320.0e3, 7016.0e3};
    private final double [] _mu = {1.327e20, 2.168e13,
3.248e14, 3.986e14,
    4.297e13, 1.267e17, 3.791e16, 5.788e15, 6.832e15, 9.965e11,
4.903e12};

    private void initComponents() {
    }


    // Variables declaration - do not modify
    // End of variables declaration

    public Trajectory (int body, double R1, double R2,
ArrayList segment)
    {
        _initialR = R1;
        _finalR = R2;
        _TOL = _tolFactor*_initialR;
        if (R2 < R1)
        {
            _direction = -1.0;
            _TOL = _tolFactor*_finalR;
        }
        _body = body;
        _currentInterpolation = 0;
        _interpolationCount = segment.size();
        _segment = segment;
        _currentSegment =
(Interpolation)_segment.get(_currentInterpolation);
        _currentOrbit = new Orbit (_initialR, 0.0, 0, _body);
        _position = _currentOrbit.getPosition ();
        _velocity = _currentOrbit.getVelocity ();
        initializePath ();
        _r = 0.05*_position.getMagnitude();
        double r = _position.getMagnitude();
        _acceleration = (_position.unitVector()).times(-
1.0*_mu[_body]/(r*r));
}

    public Trajectory (int body, Orbit initialOrbit, double R2,
ArrayList segment)
    {
        _currentOrbit = initialOrbit;
        _finalR = R2;
        _initialR = _currentOrbit.getRp();
        _TOL = _tolFactor*_initialR;
        if (R2 < _initialR)
        {
            _direction = -1.0;
            _TOL = _tolFactor*_finalR;
        }
        _body = body;
        _currentInterpolation = 0;
        _interpolationCount = segment.size();
        _segment = segment;
        _currentSegment =
(Interpolation)_segment.get(_currentInterpolation);
        _position = _currentOrbit.getPosition ();
        _velocity = _currentOrbit.getVelocity ();
        _r = 0.05*_position.getMagnitude();
        initializePath ();
        double r = _position.getMagnitude();
        _acceleration = (_position.unitVector()).times(-
1.0*_mu[_body]/(r*r));
    }

    public void setMass (double newMass)
    {
        _mass = newMass;
    }

    public void setIsp (double newIsp)
    {
        _Isp = newIsp;
    }

    public double getMass ()
    {
        return _mass;
    }

    public int getBody ()
    {
        return _body;
    }

    public Orbit getOrbit ()
    {
```

```
    return _currentOrbit;
}

public int getErrorStatus()
{
    return _errorStatus;
}

public double getTime()
{
    return _time;
}

public MyVector getAcceleration()
{
    return _acceleration;
}

public void setLimit (int newLimit)
{
    _limit = newLimit;
}

public void setBody(int newBody)
{
    _body = newBody;
}

public void setR2 (double newR2)
{
    _finalR = newR2;
    if (_finalR < _currentOrbit.getRp())
    {
        _direction = -1.0;
        _TOL = _tolFactor*_finalR;
    }
    return;
}

public void setOrbit (Orbit newOrbit)
{
    _currentOrbit = newOrbit;
    _position = _currentOrbit.getPosition();
    _velocity = _currentOrbit.getVelocity();
    _r = 0.05*_position.getMagnitude();
    double rp = _currentOrbit.getRp();
    _TOL = _tolFactor*_initialR;
        if (_finalR < rp)
        {
            _direction = -1.0;
            _TOL = _tolFactor*_finalR;
        }
        else
            _TOL = _tolFactor*rp;
        return;
    }

    public void setSegment (ArrayList newSegment)
    {
        _segment = newSegment;
        _currentInterpolation = 0;
        _interpolationCount = _segment.size();
        _currentSegment =
(Interpolation)_segment.get(_currentInterpolation);
    }

    public GeneralPath getPath ()
    {
        _path = _myPath.getPath();
        return _path;
    }

    public MyVector getThrustVector()
    {
        return _thrustVector;
    }

    public void initializePath ()
    {
        _myPath = new MyGeneralPath();
        Point2D dataPoint = new Point2D.Double
(_position.getX(), _position.getY());
        _myPath.moveTo(dataPoint);
        _time = 0.0;
        _path = _myPath.getPath();
    }

    public GeneralPath getInitialOrbitPath ()
    {
        GeneralPath startOrbit = _currentOrbit.plotOrbit(1000);
        return startOrbit;
    }

    public GeneralPath getFinalOrbitPath ()
```

```
    {
        Ellipse2D endOrbit = new Ellipse2D.Double(-_finalR, -
_finalR,
                                          2.0*_finalR,
2.0*_finalR);
        GeneralPath path = new GeneralPath(endOrbit);
        return path;
    }

    public GeneralPath getBodyPath ()
    {
        Ellipse2D planet = new Ellipse2D.Double(-
_radius[_body], -_radius[_body],
                                         _radius[_body]*2.0,
_radius[_body]*2.0);
        GeneralPath path = new GeneralPath(planet);
        return path;
    }

    public GeneralPath getSatellite()
    {
        double x = _position.getX();
        double y = _position.getY();
        double x1 = x - _r;
        double x2 = x + _r;
        double y1 = y - _r;
        double y2 = y + _r;
        Ellipse2D circle = new Ellipse2D.Double(x1,y1,_r*2.0,
_r*2.0);
        GeneralPath satellite = new GeneralPath(circle);
        return satellite;
    }

    public int update()
    {
        MyGeneralPath newPath = new
MyGeneralPath((Shape)_myPath.getPath());

        double x = _position.getX();
        double y = _position.getY();

        int count = 0;
        Point2D dataPoint = new Point2D.Double (x, y);
        newPath.moveTo(dataPoint);
        double dist = _position.getMagnitude();
        for (count = 0; count < _limit && dist*_direction <
_finalR*_direction && dist > _radius[_body] &&
                _mass > 0.0; count ++)
        {
            doWork();
            dist = _position.getMagnitude();
        }
            x = _position.getX();
            y = _position.getY();
            dataPoint = new Point2D.Double (x, y);
            newPath.lineTo(dataPoint);
            _errorStatus = 0;
        if (dist*_direction >= _finalR*_direction)
            _errorStatus = 1;
        else if (dist <= _radius[_body])
            _errorStatus = 2;
        else if (_mass <= 0.0)
            _errorStatus = 3;
        _myPath.reset();
        _myPath = newPath;
        _path = (GeneralPath)_myPath.getPath().clone();
        return _errorStatus;
    }

    private void doWork ()
    {
        getCurrentSegment ();
        _delta = calcDelta(_delta);
        _time += _delta;
        _mass -= _thrust/(_Isp*_g0)*_delta;
        _currentOrbit = new Orbit (_position, _velocity,
_body);
    }

    private Interpolation getCurrentSegment ()
    {
        if (_currentSegment.withinBounds(_currentOrbit))
            return _currentSegment;
        else
        {
            do
            {
                _currentInterpolation++;
                if (_currentInterpolation ==
_interpolationCount)
                    _currentInterpolation = 0;
                _currentSegment =
(Interpolation)_segment.get(_currentInterpolation);
```

```
            } while
(!_currentSegment.withinBounds(_currentOrbit));
            return _currentSegment;
        }
    }


    private double calcDelta (double delta)
    {
        MyVector R1_2, V1_2, R2, V2, R1, V1, A1, A2;
        MyVector gravityAcceleration, engineAcceleration,
normal;
        double r, errorx, errory, max;
        Orbit newOrbit;
        final double mu = _mu[_body];
        do
        {
            r = _position.getMagnitude();
            gravityAcceleration =
(_position.unitVector()).times(-1.0*mu/(r*r));
            engineAcceleration =
_currentSegment.getThrust(_currentOrbit).times(1.0/_mass);
            _thrustVector = engineAcceleration.times(_mass);
            _thrust = _thrustVector.getMagnitude();

            A1 = gravityAcceleration.plus(engineAcceleration);
            V1 = _velocity.plus(A1.times(delta));
            R1 = _position.plus(V1.times(delta));
            V1_2 = _velocity.plus(A1.times(0.5*delta));
            R1_2 = _position.plus(V1.times(0.5*delta));
            r = R1_2.getMagnitude();
            newOrbit = new Orbit (R1_2, V1_2, _body);
            gravityAcceleration = (R1_2.unitVector()).times(-
1.0*mu/(r*r));
            engineAcceleration =
_currentSegment.getThrust(newOrbit).times(1.0/_mass);
            A2 = gravityAcceleration.plus(engineAcceleration);
            V2 = V1_2.plus(A2.times(delta*0.5));
            R2 = R1_2.plus(V2.times(delta*0.5));
            errorx = Math.abs(R2.getX() - R1.getX());
            errory = Math.abs(R2.getY() - R1.getY());

            if (errorx > errory)
                max = errorx;
            else
                max = errory;
            if (max > _TOL)
            {
                delta *= 0.5;
            }
            else if (max < _TOL*0.5)
            {
                delta *= 1.2;
            }

        } while (max > _TOL);
        _position = R1;
        _velocity = V1;
        _acceleration = A1;
        return delta;
    }// calcDelta
}// JApplet/*
```

```
/*
 * Orbit.java
 *
 * Created on May 16, 2001, 3:16 PM
 */


/**
 *
 * @author  John E. Weglian
 * @version 2.0
 */

import java.awt.geom.GeneralPath;
import java.awt.geom.Point2D;
import java.awt.Shape;

public class Orbit extends java.lang.Object
{

    private double                      _energy;
    private double                      _h;
    private double                      _phi;
    private double                      _a;
    private double                      _p;
    private double                      _e;
    private double                      _rp;
    private double                      _ra;
    private double                      _vp;
    private double                      _va;
    private double                      _period;
    private double                      _trueAnomaly;
```

```
    private double                      _u;
    private double                      _M;
    private double                      _time;
    private double                      _rotation;

    private int                         _body;

    private MyVector                    _position;
    private MyVector                    _velocity;

    private final double [] _mu = {1.327e20, 2.168e13,
3.248e14, 3.986e14, 4.297e13, 1.267e17, 3.791e16,
    5.788e15, 6.832e15, 9.965e11, 4.903e12};


    /** Creates new Orbit */
    public Orbit()
    {
    }

    public Orbit (MyVector position, MyVector velocity, int
body)
    {
        _body = body;
        double r = position.getMagnitude();
        double v = velocity.getMagnitude();
        _position = position;
        _velocity = velocity;
        _energy = v*v*0.5 - _mu[_body]/r;
        _h = _position.crossMagnitude(_velocity);
        double temp = _h/(r*v);
        if (temp > 1.0)
            _phi = 0.0;
        else if (temp < -1.0)
            _phi = Math.PI;
        else
            _phi = Math.acos(_h/(r*v));
        if (_position.dot(_velocity) < 0.0)
            _phi *= -1.0;
        _a = -1.0*_mu[body]*0.5/_energy;
        _p = _h*_h/_mu[body];
        if (_p/_a > 1.0)
            _e = 0.0;
        else
            _e = Math.sqrt(1.0 - _p/_a);
        _rp = _a*(1.0 - _e);
        _ra = _a*(1.0 + _e);
        _vp = _h/_rp;
        _va = _h/_ra;
        if (_e < 0.005)
            _trueAnomaly = 0.0;
        else
        {
            temp = 1.0/_e*(_p/r - 1.0);
            if (temp > 1.0)
                _trueAnomaly = 0.0;
            else if (temp < -1.0)
                _trueAnomaly = Math.PI;
            else
            {
                _trueAnomaly = Math.acos(temp);
                if (_position.dot(_velocity) < 0.0)
                    _trueAnomaly *= -1.0;
            }
        }
        _rotation = _position.getTheta() - _trueAnomaly;
        calcTime();
    }

    public Orbit (MyVector position, double v, double phi,
boolean cw, int body)
    {
        _position = position;
        _phi = phi;
        _body = body;
        double r = _position.getMagnitude();
        double angle;
        if (cw)
            angle = -Math.PI + _phi;
        else
            angle = Math.PI - _phi;
        _velocity = _position.unitVector().rotate(angle);
        _velocity.setMagnitude(v);
        _energy = v*v*0.5 - _mu[_body]/r;
        _h = _position.crossMagnitude(_velocity);
        _a = -1.0*_mu[body]*0.5/_energy;
        _p = _h*_h/_mu[body];
        if (_p/_a > 1.0)
            _e = 0.0;
        else
            _e = Math.sqrt(1.0 - _p/_a);
        _rp = _a*(1.0 - _e);
        _ra = _a*(1.0 + _e);
        _vp = _h/_rp;
```

```
        _va = _h/_ra;
        double temp = (_p/r - 1.0)/_e;
        if (temp > 1.0)
            _trueAnomaly = 0.0;
        else if (temp < -1.0)
            _trueAnomaly = Math.PI;
        else
        {
            _trueAnomaly = Math.acos(temp);
            if (_position.dot(_velocity) < 0.0)
                _trueAnomaly *= -1.0;
        }
        _rotation = _position.getTheta() - _trueAnomaly;
        calcTime();
    }

    public Orbit (double a, double e, double trueAnomaly, int
body, double rotation)
    {
        _rotation = rotation;
        _a = a;
        _e = e;
        _trueAnomaly = trueAnomaly;
        _body = body;
        _energy = -_mu[_body]*0.5/_a;
        _p = _a*(1.0 - _e*_e);
        _h = Math.sqrt(_p*_mu[_body]);
        _rp = _a*(1.0 - _e);
        _ra = _a*(1.0 + _e);
        _vp = _h/_rp;
        _va = _h/_ra;
        setVectors();
        calcTime();
    }

    public Orbit (double a, double e, double trueAnomaly, int
body)
    {
        this(a, e, trueAnomaly, body, 0.0);
    }

    public double getEnergy ()
    {
        return _energy;
    }

    public double getH ()
    {
        return _h;
    }

    public double getPhi ()
    {
        return _phi;
    }

    public double getA ()
    {
        return _a;
    }

    public double getP ()
    {
        return _p;
    }

    public double getE ()
    {
        return _e;
    }

    public double getRa ()
    {
        return _ra;
    }

    public double getRp ()
    {
        return _rp;
    }

    public double getVp ()
    {
        return _vp;
    }

    public double getVa ()
    {
        return _va;
    }

    public double getPeriod ()
    {
        if (_e < 1.0)
```

```
        return _period;
    else
        return -1.0;
}

public double getTrueAnomaly ()
{
    double answer = Math.toDegrees(_trueAnomaly);
    if (answer < 0.0)
        answer += 360.0;
    return answer;
}

public double getMeanAnomaly ()
{
    double answer = Math.toDegrees(_M);
    if (answer < 0.0)
        answer += 360.0;
    return answer;
}

public MyVector getPosition ()
{
    return _position;
}

public MyVector getVelocity ()
{
    return _velocity;
}

public double getTime ()
{
    return _time;
}

public double getRotation ()
{
    return _rotation;
}

private void solveEllipse ()
{
    double min, mid, max, test, eval;
    double r, v;
    double newTime = _time;
    double TOL = 0.000001;
    while (newTime < 0.0)
        newTime += _period;
    while (newTime > _period)
        newTime -= _period;
    _M = newTime*2.0*Math.PI/_period;
    if (_M < Math.PI)
    {
        min = 0.0;
        max = Math.PI;
    }
    else
    {
        min = Math.PI;
        max = 2.0*Math.PI;
    }
    do
    {
        mid = (min + max)*0.5;
        test = mid - _e*Math.sin(mid);
        if (test < _M)
            min = mid;
        else
            max = mid;
        eval = Math.abs(test - _M);
    } while (eval > TOL);
    _u = mid;
    double check = Math.cos(_u);
    if (_M < Math.PI)
    {
        min = 0.0;
        max = Math.PI;
    }
    else
    {
        min = -Math.PI;
        max = 0.0;
    }
    do
    {
        mid = (min + max)*0.5;
        test = (_e + Math.cos(mid))/(1.0 +
_e*Math.cos(mid));

        if ((_u < Math.PI && test > check) || (_u > Math.PI
&& test < check))
            min = mid;
        else
```

```
                max = mid;
            eval = Math.abs(test - check);
        } while (eval > TOL);
        _trueAnomaly = mid;
    }

    private void solveHyperbola ()
    {
        double min, mid, max, test, eval;
        double r, v;
        double TOL = 0.000001;
        _M = _time/Math.sqrt(-1.0*_a*_a*_a/_mu[_body]);
        if (_M == 0.0)
        {
            _u = 0.0;
            _trueAnomaly = 0.0;
            return;
        }
        double MTOL = _M*TOL;
        if (_time > 0.0)
        {
            min = 0.0;
            max = 10.0;
            while ((_e*MyMath.sinh(max) - max) < _M)
                max *=10.0;
        }
        else
        {
            max = 0.0;
            min = -10.0;
            while ((_e*MyMath.sinh(min) - min) > _M)
                min *=10.0;
        }

        do
        {
            mid = (min + max)*0.5;
            test = _e*MyMath.sinh(mid) - mid;

            if ((_M > 0.0 && test > _M) || (_M < 0.0 && test <
_M))
                max = mid;
            else
                min = mid;
            eval = Math.abs(test - _M);
        } while (eval > MTOL);
        _u = mid;
            double check = MyMath.cosh(_u);
            double UTOL = TOL*check;
            if (_M > 0.0)
            {
                min = 0.0;
                max = Math.acos(-1.0/_e);
            }
            else
            {
                min = -Math.acos(-1.0/_e);
                max = 0.0;
            }
            do
            {
                mid = (min + max)*0.5;
                test = (_e + Math.cos(mid))/(1.0 +
_e*Math.cos(mid));
                if ((test > check && _M > 0.0) || (test < check &&
_M < 0.0))
                    max = mid;
                else
                    min = mid;
                eval = Math.abs(test - check);
            } while (eval > UTOL && Math.abs(min-max) > TOL);
            _trueAnomaly = mid;
        }

        private void solveParabola()
        {
            double min, mid, max, test, eval;
            double TOL = 0.000001;
            if (_time < 0.0)
            {
                min = -Math.PI;
                max = 0.0;
            }
            else
            {
                min = 0.0;
                max = Math.PI;
            }
            do
            {
                mid = (min + max)*0.5;
                test = _h*_h*_h*0.5/(_mu[_body] *
_mu[_body])*(Math.tan(mid*0.5)
                + Math.pow(Math.tan(mid*0.5),3.0)/3.0);
```

```
            if ((test > _time && _time > 0.0) || (test < _time
&& _time < 0.0))
                max = mid;
            else
                min = mid;
            eval = Math.abs(test - _time);
        } while (eval > TOL);
        _trueAnomaly = mid;
    }

    private void setVectors ()
    {
        double r = _p/(1.0 + _e*Math.cos(_trueAnomaly));
        double v = Math.sqrt(2.0*(_energy + _mu[_body]/r));
        double temp = _h/(r*v);
        if (temp > 1.0)
            _phi = 0.0;
        else if (temp < -1.0)
            _phi = Math.PI;
        else
            _phi = Math.acos(_h/(r*v));
        if (_trueAnomaly < 0.0 || _trueAnomaly > Math.PI)
            _phi *= -1.0;
        _position = new MyVector (r, (_trueAnomaly +
_rotation), true);
        double theta = _trueAnomaly + Math.PI*0.5 - _phi +
_rotation;
        _velocity = new MyVector (v, theta, true);
    }

    private void calcTime ()
    {
        if (_e < 1.0)
        {
            _period =
2.0*Math.PI/Math.sqrt(_mu[_body])*Math.pow(_a, 1.5);
            _u = Math.acos((_e + Math.cos(_trueAnomaly))/(1.0 +
_e*Math.cos(_trueAnomaly)));
            if (_trueAnomaly < 0.0 || _trueAnomaly > Math.PI)
                _u = -_u;
            _M = _u - _e*Math.sin(_u);
            _time = _M*_period*0.5/Math.PI;
        }
        else if (_e > 1.0)
        {
            _u = MyMath.arcosh((_e +
Math.cos(_trueAnomaly))/(1.0 + _e*Math.cos(_trueAnomaly)));
            _M = _e*MyMath.sinh(_u) - _u;
            _time = Math.sqrt(-1.0*_a*_a*_a/_mu[_body])*_M;
        }
        else
        {
            _time =
_h*_h*_h*0.5/(_mu[_body]*_mu[_body])*(Math.tan(_trueAnomaly*0.5
)
            + Math.pow(Math.tan(_trueAnomaly*0.5),3.0)/3.0);
        }
    }

    public Orbit setTime(double time)
    {
        _time = time;
        if (_e < 1.0)
            solveEllipse();
        else if (_e > 1.0)  // Hyperbolic
            solveHyperbola();
        else
            solveParabola();
        setVectors();
        return this;
    }

    public Orbit addTime(double moreTime)
    {
        _time += moreTime;
        if (_e < 1.0)
            solveEllipse();
        else if (_e > 1.0)  // Hyperbolic
            solveHyperbola();
        else
            solveParabola();
        setVectors();
        return this;
    }

    public Orbit toDistance (double distance)
    {
        if (distance > _ra && _e < 1.0)
        {
            System.out.println("Error.  Distance > ra");
            System.out.println("ra = " + _ra);
            return null;
        }
        else if (distance < _rp)
```

```
        {
            System.out.println("Error.  Distance < rp");
            System.out.println("rp = " + _rp);
            return null;
        }
        else if (_e == 0.0)
        {
            System.out.println("Error.  Not defined for
circular orbit");
            return null;
        }
        double r = distance;
        double newTrueAnomaly;
        boolean flip = false;
        if ((_p/r - 1.0)/_e > 1.0)
            newTrueAnomaly = 0.0;
        else
            newTrueAnomaly = Math.acos((_p/r - 1.0)/_e);
        if (-newTrueAnomaly > _trueAnomaly)
            newTrueAnomaly *= -1.0;
        else if (_trueAnomaly > newTrueAnomaly && _e < 1.0)
            flip = true;
        _trueAnomaly = newTrueAnomaly;
        setVectors();
        calcTime();
        if (flip)
            _time += _period;
        return this;
    }

    public Orbit setTrueAnomaly (double newAnomaly)
    {
        _trueAnomaly = newAnomaly;
        setVectors();
        calcTime();
        return this;
    }

    public GeneralPath plotOrbit (int points)
    {
        double maxDist = 100.0*_rp;
        return plotOrbit(points, maxDist);
    }

    public GeneralPath plotOrbit (int points, GeneralPath path)
    {
        double maxDist = 100.0*_rp;
        return plotOrbit(points, maxDist, path);
    }

    public GeneralPath plotOrbit (int points, double maxDist)
    {
        GeneralPath path = new GeneralPath();
        return plotOrbit(points, maxDist, path);
    }

    public GeneralPath plotOrbit (int points, double maxDist,
GeneralPath path)
    {
        double oldTrueAnomaly = _trueAnomaly;
        double DELTA, max, min;
        MyGeneralPath myPath = new MyGeneralPath();
        Point2D dataPoint = new Point2D.Double(0.0, 0.0);
        myPath.moveTo (dataPoint);
        myPath.lineTo (dataPoint);
        if (_e < 1.0)
        {
            min = -Math.PI;
            max = Math.PI;
        }
        else
        {
            toDistance(maxDist);
            if (_trueAnomaly > 0.0)
                _trueAnomaly *= -1.0;
            min = _trueAnomaly;
            max = -min;
        }
        DELTA = (max - min)/((double)(points));
        _trueAnomaly = min;
        setVectors();
        dataPoint = new Point2D.Double (_position.getX(),
_position.getY());
        myPath.moveTo(dataPoint);

        for (_trueAnomaly = min; _trueAnomaly <= max;
_trueAnomaly += DELTA)
        {
            setVectors();
            dataPoint = new Point2D.Double (_position.getX(),
_position.getY());
            myPath.lineTo(dataPoint);
        }
        _trueAnomaly = oldTrueAnomaly;
```

```
        setVectors();
        calcTime();
        path.append((Shape)(myPath.getPath()), false);
        return path;
    }

    public Orbit deltaV (double velocityChange)
    {
        double v = _velocity.getMagnitude();
        System.out.println("v = " + v);
        v += velocityChange;
        System.out.println("v now = " + v);
        MyVector newVelocity = _velocity.unitVector();
        newVelocity.setMagnitude (v);
        Orbit result = new Orbit (_position, newVelocity,
_body);
        return result;
    }

    public Orbit deltaV (MyVector velocityChange)
    {
        MyVector newVelocity = _velocity.plus(velocityChange);
        Orbit result = new Orbit (_position, newVelocity,
_body);
        return result;
    }
}
```

```
/*
 * MyVector.java
 *
 * Created on April 27, 2001, 12:42 PM
 */


/**
 *
 * @author  John E. Weglian
 * @version 1.0
 */

import java.lang.Math;

public class MyVector
{
    private double                                  _x;
    private double                                  _y;
    private double                                  _magnitude;
    private double                                  _theta;


    /** Creates new MyVector */

    public MyVector ()
    {
    }

    public MyVector (double magnitude, double theta, boolean
polar)
    {
        if (polar)
        {
            _x = magnitude*Math.cos(theta);
            _y = magnitude*Math.sin(theta);
            _magnitude = magnitude;
            _theta = theta;
        }
        else
        {
            _x = magnitude;
            _y = theta;
            _magnitude = Math.sqrt(_x*_x + _y*_y);
            _theta = Math.atan2(_y, _x);
            if (_theta < 0.0)
                _theta += 2.0*Math.PI;
        }
    }

    public MyVector (double x, double y)
    {
        _x = x;
        _y = y;
        _magnitude = Math.sqrt(_x*_x + _y*_y);
        _theta = Math.atan2(_y, _x);
        if (_theta < 0.0)
            _theta += 2.0*Math.PI;
    }

    public double getMagnitude ()
    {
        return _magnitude;
    }

    public double getX ()
    {
```

```
    return _x;
}

public double getY ()
{
    return _y;
}

public double getTheta ()
{
    return _theta;
}

public double getDelta (MyVector b)
{
    return b.getTheta() - _theta;
}

public MyVector unitVector()
{
    double newX = Math.cos (_theta);
    double newY = Math.sin (_theta);
    MyVector newVector = new MyVector (newX, newY);
    return newVector;
}

public void setMagnitude (double magnitude)
{
    _x = magnitude*Math.cos(_theta);
    _y = magnitude*Math.sin(_theta);
    _magnitude = magnitude;
}

public void setTheta (double theta)
{
    _theta = theta;
    _x = _magnitude*Math.cos(_theta);
    _y = _magnitude*Math.sin(_theta);
}

public MyVector rotate (double delta)
{
    setTheta (_theta + delta);
    return this;
}

public MyVector plus (MyVector addVector)
    {
        double newX = _x + addVector.getX();
        double newY = _y + addVector.getY();
        MyVector newVector = new MyVector (newX, newY);
        return newVector;
    }

    public MyVector times (double mult)
    {
        double newX = _x*mult;
        double newY = _y*mult;
        MyVector newVector = new MyVector (newX, newY);
        return newVector;
    }

    public double crossMagnitude (MyVector b)
    {
        double delta = getDelta(b);
        return
Math.abs(_magnitude*b.getMagnitude()*Math.sin(delta));
    }

    public double dot (MyVector b)
    {
        double delta = getDelta (b);
        return _magnitude*b.getMagnitude()*Math.cos(delta);
    }
}
```

```
/*
 * MyGeneralPath.java
 *
 * Created on May 21, 2001, 2:28 PM
 */


/**
 *
 * @author  John E. Weglian
 * @version 1.0
 */

import java.awt.geom.GeneralPath;
import java.awt.geom.Point2D;
import java.awt.Shape;

public class MyGeneralPath
{
```

```
    GeneralPath _path = null;

    /** Creates new MyGeneralPath */
    public MyGeneralPath()
    {
        _path = new GeneralPath();
    }

    public MyGeneralPath (Shape s)
    {
        _path = new GeneralPath (s);
    }

    public GeneralPath lineTo (Point2D newPoint)
    {
        float x = (float)newPoint.getX();
        float y = (float)newPoint.getY();

        _path.lineTo(x, y);
        return _path;
    }

    public GeneralPath moveTo (Point2D newPoint)
    {
        float x = (float)newPoint.getX();
        float y = (float)newPoint.getY();
        _path.moveTo(x, y);
        return _path;
    }

    public GeneralPath getPath ()
    {
        return _path;
    }

    public GeneralPath append(Shape s, boolean connect)
    {
        _path.append(s, connect);
        return _path;
    }

    public void reset()
    {
        _path.reset();
    }
}
```

```
/*
 * Interpolation.java
 *
 * Created on August 31, 2001, 7:00 PM
 */


/**
 *
 * @author  John E. Weglian
 * @version 1.0
 */

import java.lang.Math;

public class Interpolation extends java.lang.Object
{

    private     boolean         _useTrueAnomaly             =
true;
    private     boolean         _velocityBased              =
true;

    private     double          _startAnomaly;
    private     double          _endAnomaly;
    private     double          _startAngle;
    private     double          _endAngle;
    private     double          _startMagnitude;
    private     double          _endMagnitude;
    private     double          _angleConstant;
    private     double          _magnitudeConstant;

    /** Creates new Interpolation */
    public Interpolation(boolean useTrueAnomaly, boolean
velocityBased, double startAnomaly, double endAnomaly,
                         double startAngle, double endAngle,
double startMagnitude, double endMagnitude)
    {
        _useTrueAnomaly = useTrueAnomaly;
        _velocityBased = velocityBased;
        _startAnomaly = startAnomaly;
        _endAnomaly = endAnomaly;
        _startAngle = startAngle;
        _endAngle = endAngle;
        _startMagnitude = startMagnitude;
        _endMagnitude = endMagnitude;
```

```
        _angleConstant = (_endAngle - _startAngle)/(_endAnomaly
- _startAnomaly);
        _magnitudeConstant = (_endMagnitude -
_startMagnitude)/(_endAnomaly - _startAnomaly);
    }

    public double getAngle(Orbit currentOrbit)
    {
        double x, y;
        if (_useTrueAnomaly)
            x = currentOrbit.getTrueAnomaly();
        else
            x = currentOrbit.getMeanAnomaly();
        y = (x-_startAnomaly)*_angleConstant + _startAngle;
        return y;
    }

    public double getMagnitude(Orbit currentOrbit)
    {
        double x, y;
        if (_useTrueAnomaly)
            x = currentOrbit.getTrueAnomaly();
        else
            x = currentOrbit.getMeanAnomaly();
        y = (x-_startAnomaly)*_magnitudeConstant +
_startMagnitude;
        return y;

    }

    public MyVector getThrust (Orbit currentOrbit)
    {
        double angle, magnitude;
        angle = Math.toRadians(getAngle(currentOrbit));
        if (_velocityBased)
            angle+=currentOrbit.getVelocity().getTheta();
        else
            angle+=currentOrbit.getPosition().getTheta();
        magnitude = getMagnitude(currentOrbit);
        MyVector thrust = new MyVector (magnitude, angle,
true);
        return thrust;
    }

    public double getEndAnomaly ()
    {
        return _endAnomaly;
    }

    public double getStartAnomaly ()
    {
        return _startAnomaly;
    }

    public boolean withinBounds (Orbit currentOrbit)
    {
        double currentAnomaly;
        if (_useTrueAnomaly)
            currentAnomaly = currentOrbit.getTrueAnomaly();
        else
            currentAnomaly = currentOrbit.getMeanAnomaly();
        if (currentAnomaly <= _endAnomaly && currentAnomaly >=
_startAnomaly)
            return true;
        else
            return false;
    }
}
```

```
/*
 * MyPlot.java
 *
 * Created on May 21, 2001, 3:31 PM
 */


/**
 *
 * @author  John E. Weglian
 * @version 1.0
 */

import java.awt.geom.AffineTransform;
import java.awt.geom.Point2D;
import java.awt.geom.GeneralPath;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.BasicStroke;

public class MyPlot
```

```
{
    private AffineTransform             _transform
= null;
    private GeneralPath                 _path
= null;
    private int                         _dimension
= 500;

    /** Creates new MyPlot */

    public MyPlot(GeneralPath path)
    {
       _path = path;
       scale ();
    }

    public void setPath (GeneralPath newPath)
    {
        _path = newPath;
    }

    public void scale()
    {
        _transform = new AffineTransform();
        _transform.setToIdentity();
        double x = _path.getBounds2D().getCenterX();
        double y = _path.getBounds2D().getCenterY();
        _transform.translate(-x, -y);
        _path.transform(_transform);
        double width = _path.getBounds2D().getWidth();
        double height = _path.getBounds2D().getHeight();
        double max = width;
        if (height > max)
            max = height;
        double scale = ((double)_dimension)/max;
        _transform.setToIdentity();
        _transform.scale(scale, scale);
        _path.transform(_transform);
        _transform.setToIdentity();
        _transform.translate(max*scale*0.5, max*scale*0.5);
        _path.transform(_transform);
    }

    public void plot (String title)
    {
        JFrame frame = new JFrame(title);
        PlotPanel panel = new PlotPanel();
        frame.getContentPane().add(panel);
        frame.setSize(_dimension + 40, _dimension + 60);
        frame.show();
    }

    class PlotPanel extends JPanel
    {
        public PlotPanel ()
        {
            super ();
        }

        public void paintComponent (Graphics g)
        {
            super.paintComponent(g);
            Graphics2D g2 = (Graphics2D) g;
            g2.setPaint(Color.black);
            g2.draw(_path);
        }
    }// class PlotPanel

}// class MyPlot
```

```
/*
 * SetUp.java
 *
 * Created on September 20, 2001, 3:26 PM
 */


/**
 *
 * @author  John E. Weglian
 * @version 1.0
 */
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JComboBox;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;
import javax.swing.JTextField;
import javax.swing.BoxLayout;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.text.NumberFormat;
```

```
import java.awt.Container;
import java.awt.GridLayout;

import java.util.ArrayList;


public class SetUp extends java.lang.Object
{
    private     JFrame          _segmentFrame
= null;

    private     JLabel          _anomalyLabel
= null;
    private     JLabel          _anomalyErrorMessage
= null;
    private     JLabel          _angleErrorMessage
= null;
    private     JLabel          _thrustErrorMessage
= null;
    private     JLabel          _startAnomalyLabel
= null;
    private     JLabel          _endAnomalyLabel
= null;
    private     JLabel          _startAngleLabel
= null;
    private     JLabel          _endAngleLabel
= null;
    private     JLabel          _startThrustLabel
= null;
    private     JLabel          _endThrustLabel
= null;

    private     JRadioButton    _trueAnomalyButton
= null;
    private     JRadioButton    _meanAnomalyButton
= null;
    private     JRadioButton    _velocityAngleButton
= null;
    private     JRadioButton    _positionAngleButton
= null;
    private     JRadioButton    _nUnitButton
= null;
    private     JRadioButton    _mnUnitButton
= null;
    private     JRadioButton    _lbUnitButton
= null;
    private     JTextField      _startAnomalyTextField
= null;
    private     JTextField      _endAnomalyTextField
= null;
    private     JTextField      _startAngleTextField
= null;
    private     JTextField      _endAngleTextField
= null;
    private     JTextField      _startThrustTextField
= null;
    private     JTextField      _endThrustTextField
= null;

    private     JButton         _OKButton
= null;

    private     double          _firstAnomaly
= 0.0;
    private     double          _initialAnomaly
= 0.0;
    private     double          _endAnomaly
= 0.0;
    private     double          _initialAngle
= 0.0;
    private     double          _endAngle
= 0.0;
    private     double          _initialThrust
= 0.0;
    private     double          _endThrust
= 0.0;
    private     double          _factor
= 1.0;

    private     int             _textBoxLength
= 20;
    private     int             _scaleIndex
= 0;
    private     int             _count
= 0;

    private     boolean         _initAnomalySet
= false;
    private     boolean         _useTrueAnomaly
= true;
    private     boolean         _endAnomalySet
= false;
```

```
    private     boolean         _initAngleSet
= false;
    private     boolean         _endAngleSet
= false;
    private     boolean         _initThrustSet
= false;
    private     boolean         _endThrustSet
= false;
    private     boolean         _velocityBased
= true;
    private     boolean         _segmentsSet
= false;

    private     double []       _scale
= {1.0, 0.001, 4.45};

    private     ArrayList       _segmentArrayList
= new ArrayList();

//    private     Interpolation []    _segment;

    private     Trajectory      _trajectory
= null;

    private     String          _errorMessage
= new String ("");

    /** Creates new Interface */
    public SetUp()
    {
        //        setUpSegments();
    }

    public void setUpSegments (Trajectory currentTrajectory,
double initialAnomaly)
    {
        _trajectory = currentTrajectory;
        _initialAnomaly = initialAnomaly;
        _segmentFrame = new JFrame ("Define trajectory
segments");
        JPanel anomalyPanel = getAnomalyPanel();
        JPanel anglePanel = getAnglePanel ();
        JPanel thrustPanel = getThrustPanel ();
        Container contentPane = _segmentFrame.getContentPane();
        contentPane.setLayout( new GridLayout(1,3));

        contentPane.add(anomalyPanel);
        contentPane.add (anglePanel);
        contentPane.add (thrustPanel);
        _segmentFrame.pack();
        _segmentFrame.show();
    }

    private JPanel getAnomalyPanel ()
    {
        JPanel anomalyPanel = new JPanel ();
        anomalyPanel.setLayout (new BoxLayout(anomalyPanel,
BoxLayout.Y_AXIS));
        _anomalyLabel = new JLabel ("True Anomaly Definition");
        anomalyPanel.add(_anomalyLabel);
        _trueAnomalyButton = new JRadioButton ("True Anomaly",
true);
        _trueAnomalyButton.addActionListener (new
ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                _useTrueAnomaly = true;
                _anomalyLabel.setText("True Anomaly
Definition");
            }
        });
        _meanAnomalyButton = new JRadioButton ("Mean Anomaly",
true);
        _meanAnomalyButton.addActionListener (new
ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                _useTrueAnomaly = false;
                _anomalyLabel.setText("Mean Anomaly
Definition");
            }
        });
        _trueAnomalyButton.setEnabled (!_initAnomalySet);
        _meanAnomalyButton.setEnabled (!_initAnomalySet);
        ButtonGroup group = new ButtonGroup ();
        group.add (_trueAnomalyButton);
        group.add (_meanAnomalyButton);
        anomalyPanel.add(_trueAnomalyButton);
        anomalyPanel.add(_meanAnomalyButton);

        _startAnomalyLabel = new JLabel ("Starting Anomaly
value in degrees");
```

```
        anomalyPanel.add(_startAnomalyLabel);

        String value = new String("");
        value = Double.toString(_initialAnomaly);
        _startAnomalyTextField = new
JTextField(_textBoxLength);
        _startAnomalyTextField.setText(value);
        _startAnomalyTextField.setEditable (false);
        _initAnomalySet = true;
        _startAnomalyTextField.addActionListener(new
ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    String testString =
_startAnomalyTextField.getText();
                    _initialAnomaly =
Double.valueOf(testString).doubleValue();
                    _anomalyErrorMessage.setText("");
                    _initAnomalySet = true;
                    _firstAnomaly = _initialAnomaly;
                    _endAnomalyTextField.requestFocus();
                }
                catch (NumberFormatException error)
                {
                    _anomalyErrorMessage.setText("Enter a
number");
                    _initAnomalySet = false;
                }
            }
        });
        anomalyPanel.add(_startAnomalyTextField);

        _endAnomalyLabel = new JLabel ("Ending Anomaly value in
degrees");
        anomalyPanel.add(_endAnomalyLabel);

        _endAnomalyTextField = new JTextField(_textBoxLength);
        _endAnomalyTextField.setEditable (true);
        _endAnomalyTextField.requestFocus();
        _endAnomalyTextField.addActionListener(new
ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    String testString =
_endAnomalyTextField.getText();
                    _endAnomaly =
Double.valueOf(testString).doubleValue();
                    _startAngleTextField.requestFocus();
                    if (_endAnomaly < _initialAnomaly)
                        _endAnomaly += 360.0;
                    if (_endAnomaly <= _initialAnomaly + 360.0)
                    {
                        _anomalyErrorMessage.setText("");
                        _endAnomalySet = true;
                    }
                    else
                    {
                        _anomalyErrorMessage.setText("Begining
and end anomaly must be within 360 degrees");
                        _endAnomalySet = false;
                    }
                }
                catch (NumberFormatException error)
                {
                    _anomalyErrorMessage.setText("Enter a
number");
                    _endAnomalySet = false;
                }
            }
        });
        anomalyPanel.add(_endAnomalyTextField);
        _anomalyErrorMessage = new JLabel("");
        anomalyPanel.add(_anomalyErrorMessage);
        return anomalyPanel;
    } //getAnomalyPanel

    private JPanel getAnglePanel ()
    {
        JPanel anglePanel = new JPanel ();
        anglePanel.setLayout (new BoxLayout(anglePanel,
BoxLayout.Y_AXIS));
        JLabel angleLabel = new JLabel ("Thrust Angle
Definition");
        anglePanel.add(angleLabel);
        _velocityAngleButton = new JRadioButton ("Relative to
Velocity Vector", true);
        _velocityAngleButton.addActionListener (new
ActionListener()
```

```
        {
            public void actionPerformed(ActionEvent event)
            {
                _velocityBased = true;
            }
        });
        _positionAngleButton = new JRadioButton ("Relative to
Position Vector", true);
        _positionAngleButton.addActionListener (new
ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                _velocityBased = false;
            }
        });

        ButtonGroup group = new ButtonGroup ();
        group.add (_velocityAngleButton);
        group.add (_positionAngleButton);
        anglePanel.add(_velocityAngleButton);
        anglePanel.add(_positionAngleButton);

        _startAngleLabel = new JLabel ("Starting Angle in
degrees");
        anglePanel.add(_startAngleLabel);

        _startAngleTextField = new JTextField(_textBoxLength);
        _startAngleTextField.setEditable (true);
        _startAngleTextField.addActionListener(new
ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    String testString =
_startAngleTextField.getText();
                    _initialAngle =
Double.valueOf(testString).doubleValue();
                    while (_initialAngle < -360.0)
                        _initialAngle += 360.0;
                    while (_initialAngle > 360.0)
                        _initialAngle -= 360.0;
                    _angleErrorMessage.setText("");
                    _initAngleSet = true;
                    _endAngleTextField.requestFocus();
                }
                catch (NumberFormatException error)
                {
                    _angleErrorMessage.setText("Enter a
number");
                    _initAngleSet = false;
                }
            }
        });
        anglePanel.add(_startAngleTextField);

        _endAngleLabel = new JLabel ("Ending Angle in
degrees");
        anglePanel.add(_endAngleLabel);

        _endAngleTextField = new JTextField(_textBoxLength);
        _endAngleTextField.setEditable (true);
        _endAngleTextField.addActionListener(new
ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    String testString =
_endAngleTextField.getText();
                    _endAngle =
Double.valueOf(testString).doubleValue();
                    while (_endAngle < -360.0)
                        _endAngle += 360.0;
                    while (_initialAngle > 360.0)
                        _endAngle -= 360.0;
                    _endAngleSet = true;
                    _startThrustTextField.requestFocus();
                }
                catch (NumberFormatException error)
                {
                    _angleErrorMessage.setText("Enter a
number");
                    _endAngleSet = false;
                }
            }
        });
        anglePanel.add(_endAngleTextField);
        _OKButton = new JButton ("OK");
        _OKButton.addActionListener (new ActionListener ()
        {
```

```
            public void actionPerformed (ActionEvent e)
            {
                _startAnomalyTextField.postActionEvent();
                _endAnomalyTextField.postActionEvent();
                _startAngleTextField.postActionEvent();
                _endAngleTextField.postActionEvent();
                _startThrustTextField.postActionEvent();
                _endThrustTextField.postActionEvent();
                if (_initAnomalySet && _endAnomalySet &&
_initAngleSet && _endAngleSet &&
                _initThrustSet && _endThrustSet)
                    defineSegment();
                else
                {
                    _errorMessage = "";
                    if (!_initAnomalySet)
                        _errorMessage = "Need Initial Anomaly
";
                    if (!_endAnomalySet)
                        _errorMessage += "Need Final Anomaly";

_anomalyErrorMessage.setText(_errorMessage);
                    _errorMessage = "";
                    if (!_initAngleSet)
                        _errorMessage = "Need Initial Angle  ";
                    if (!_endAngleSet)
                        _errorMessage += "Need Final Angle";
                    _angleErrorMessage.setText(_errorMessage);
                    _errorMessage = "";
                    if (!_initThrustSet)
                        _errorMessage = "Need Initial Thrust
";
                    if (!_endThrustSet)
                        _errorMessage += "Need Final Thrust";
                    _thrustErrorMessage.setText(_errorMessage);
                }
            }
        });

        _angleErrorMessage = new JLabel("");
        anglePanel.add(_angleErrorMessage);

        anglePanel.add(_OKButton);
        return anglePanel;
    } //getAnomalyPanel

    private JPanel getThrustPanel ()
    {
        JPanel thrustPanel = new JPanel ();
        thrustPanel.setLayout (new BoxLayout(thrustPanel,
BoxLayout.Y_AXIS));
        JLabel thrustLabel = new JLabel ("Thrust Magnitude
Definition");
        thrustPanel.add(thrustLabel);
        _nUnitButton = new JRadioButton ("N", true);
        _nUnitButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                if (_initThrustSet)
                {
                    _initialThrust *= 1.0/_factor;
                }
                if (_endThrustSet)
                {
                    _endThrust *= 1.0/_factor;
                }
                _factor = 1.0;
            }
        });
        _mnUnitButton = new JRadioButton ("mN", true);
        _mnUnitButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                if (_initThrustSet)
                {
                    _initialThrust *= 0.001/_factor;
                }
                if (_endThrustSet)
                {
                    _endThrust *= 0.001/_factor;
                }
                _factor = 0.001;
            }
        });
        _lbUnitButton = new JRadioButton ("lb", true);
        _lbUnitButton.addActionListener (new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                if (_initThrustSet)
                {
                    _initialThrust *= 4.44822/_factor;
```

```
                }
                if (_endThrustSet)
                {
                    _endThrust *= 4.44822/_factor;
                }
                _factor = 4.44822;
            }
        });

        ButtonGroup group = new ButtonGroup ();
        group.add (_nUnitButton);
        group.add (_mnUnitButton);
        group.add (_lbUnitButton);
        thrustPanel.add(_nUnitButton);
        thrustPanel.add(_mnUnitButton);
        thrustPanel.add(_lbUnitButton);

        _startThrustLabel = new JLabel ("Starting thrust
value");
        thrustPanel.add(_startThrustLabel);

        _startThrustTextField = new JTextField(_textBoxLength);
        _startThrustTextField.setEditable (true);
        _startThrustTextField.addActionListener(new
ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    String testString =
_startThrustTextField.getText();
                    _initialThrust = _factor *
Double.valueOf(testString).doubleValue();
                    _thrustErrorMessage.setText("");
                    _initThrustSet = true;
                    _endThrustTextField.requestFocus();
                }
                catch (NumberFormatException error)
                {
                    _thrustErrorMessage.setText("Enter a
number");
                    _initThrustSet = false;
                }
            }
        });
        thrustPanel.add(_startThrustTextField);

        _endThrustLabel = new JLabel ("Ending thrust value");
        thrustPanel.add(_endThrustLabel);

        _endThrustTextField = new JTextField(_textBoxLength);
        _endThrustTextField.setEditable (true);
        _endThrustTextField.addActionListener(new
ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    String testString =
_endThrustTextField.getText();
                    _endThrust = _factor *
Double.valueOf(testString).doubleValue();
                    _endThrustSet = true;
                    _OKButton.requestFocus();
                }
                catch (NumberFormatException error)
                {
                    _thrustErrorMessage.setText("Enter a
number");
                    _endThrustSet = false;
                }
            }
        });
        thrustPanel.add(_endThrustTextField);
        _thrustErrorMessage = new JLabel("");
        thrustPanel.add(_thrustErrorMessage);
        return thrustPanel;
    } //getAnomalyPanel

    private void defineSegment()
    {
        Interpolation newSegment = new Interpolation
(_useTrueAnomaly, _velocityBased, _initialAnomaly,
        _endAnomaly, _initialAngle, _endAngle, _initialThrust,
_endThrust);
        _segmentArrayList.add(newSegment);
        _count ++;
        _segmentFrame.dispose();
        if (_endAnomaly < _firstAnomaly + 360.0)
        {
            setUpSegments(_trajectory, _endAnomaly);
            return;
```

```
        }
        else
        {
            _trajectory.setSegment (_segmentArrayList);
            _segmentsSet = true;
        }
    }

} //SetUp class
```