Documentation for
"Sim Solar System"
©1992 by Dean Dauger
December 6, 1992

Introduction

My goal in writing this program was to create a tool that could help people with an interest in physics model, understand, and visualize objects under the influence of a force law proportional to the distance between the objects raised to any power. The most obvious phenomenon to model is gravitation, specifically Newtonian gravitation. This is perhaps one of the most interesting and modeled areas of physics in the three centuries since its discovery. Some programs, looking at a subset of its possibilities, which still is by no means small, consider only two-dimensional cases. Others attempt to model three dimensions, but it commonly becomes difficult to visualize what is going on if all that is provided are projected coordinates, i.e., top view, side view, other side view, etc. I consider that an unnecessary limitation and attempt to address these problems. I certainly do not want anything to get in the way of the point of the program: to understand how these mechanics work and what they do by knowing what is happening in the solution.

General Layout

Most of the program is understood by doing and playing around with the controls. The main steps in creating a new "solar system" here are as follows. The program needs to know how many and what kinds of objects are to fly around, where they are and where are they going when they start, and what kind of universe they live in. I have tried to make this as straightforward as possible without denying numerous possibilities.

Starting Up - The Actors

After opening the application, three windows appear. The largest, appropriately, is the main display window. Here is where the show is, where all may see the action. The program automatically calculates the window's initial size based on the hardware screen size. The second is called a control window, situated on the lower right of the main display window. Here various statistics about the program when running appear

and a few items may be controlled using the mouse from here. More on that later. The last is a window containing the object list.

In the object list window, quick facts about the initial values of the objects are laid out in a table. Initially, there are no objects present. Clicking on the top line creates an entry with null information about the new object. You may have up to 14 different objects like this one. This window displays only the objects' colors, masses, and initial speeds for the purpose of generally recognizing which is which. Clicking on an existing entry will tell the program to open up a dialog box so that you may specify things about the object. You may delete all the objects by selecting Clear under the Edit menu or pressing ⌘-delete.

In the object dialog box, you may specify the object's mass, initial position, initial velocity, color, and radius. For color, you may specify the RGB components or click on the color box to the right of the numbers and use the Apple Color Picker. Color helps in easy identification of the object while it is flying about the screen. It is usually not immediately obvious why I have a radius specification, but it will be important in the visualization of the objects described later. Also, they specify the definition of whether or not two objects have "crashed" into each other. The object's mass is also specified here. The arrangement of the text edit boxes is my closest, simplest approximation of scientific notation. Its range should cover any reasonable real system. But let us say you know a mass in pounds and not kilograms. Just click on the units to the right of the number and it will change to pounds. The number will change accordingly, representing the same mass. Now you may enter it in in pounds. The program will convert it to and store it in kilograms when you click Okay. Position and velocity may be specified in three dimensions, but I have found that specifying these coordinates in xyz is too difficult, simply because most systems that we try to simulate have objects in circular or elliptical orbits. So position and velocity are specified in spherical coordinates. In this case it appears as altitude-azimuth-radius. This makes it simple to specify a two dimensional system in polar coordinates. The azimuth-radius coordinates follow the mathematical polar coordinate convention. Altitude is angle up from this plane, just like telescope direction coordinates. (And see, I have avoided the physics/math q j convention conflicts!) The program actually stores all positions and velocities in xyz. It automatically converts everything whenever you see the numbers. Also, just like the mass units, the units next to position and velocity change as well. The distance units cycle through meters, kilometers, feet, and miles. The velocity units have the same cycling distance units along with the per time units cycling through seconds, minutes, and hours. This is so that you don't have to always convert to meters per second. Just use miles per hour,

or feet per minute, etc. From here just click Okay, and the program will convert and store all the numbers into memory. All values stored and computed throughout the program are in standard mks units. You may take advantage of the feature, or they may be completely ignored in favor of an ideal case.

The Rules

Now that we know what to act on, the program needs to know what these things do to each other. This is where we need to look at the Universal Constants. Look under the *Options* menu, drag down to the *Edit Universe* submenu, and select *Universal Constants*. A faster way is to type ⬚-U (hold down the ⬚ key and type U). (Don't you just love playing god?) Here we can edit the most important constants of this simulated universe: G and ß. Every object in this universe obeys two laws:

$$\vec{F} = m\vec{a} \quad \text{and} \quad \vec{F} = -GmMr^{\beta}\,\hat{r}$$

The first is the well known Newton's second law. The second is a generalization of his gravitational law. m is the mass we are dealing with. M is some other mass. r is the magnitude of $\vec{r}$, the vector difference the positions of the masses. ß may be any real number. The default values of G and ß are 6.672 x 10^{-11} and -2 respectively, corresponding to Newtonian gravitation. (No, unfortunately you cannot change the units of G. (too many possibilities, like hogshead per slug-fortnight squared......:))

One check box to the right of the r dependence specification is called Crash Check. This turns on or off crash checking in the program. This is one place where that radius specification of the objects comes in. In this universe, we can say whether objects will crash when they get too close, or if the are all ideal point masses that can almost pass through each other. Computation speed usually is not lost too much. When the program finds a crash, it plays an unpleasant crashing sound. It doesn't do anything else when it sees a crash. The numbers will still compute even when the singularities of the masses become very close. However, it is a good way to tell if something went wrong. It becomes very likely that numerical errors will occur from this point on. Speaking of which.......

Also here is where you specify attributes of the numerical method. The most important is step size, meaning Δt here. This will convert units when you click on them, between seconds, minutes, hours, and days. Presently implemented are four standard numerical methods for your enjoyment: Euler's method (yep, that old one), fourth order Runge-Kutta, Adams-Bashforth four step method, and Adams-Bashforth four step method

with Moulton predictor-corrector method. Pick your choice; there are compromises on speed and accuracy. The default is 4ORK. :)

Visualization - The Stage and Seating

After placing these little beasties in their place and telling them what rules they follow, we need to see them. You may wish to start their motion now. Just select Run Orbit or Fast Loop under the Options menu, or press □-R or □-F, respectively. These two are the same except that Fast Loop disallows all background tasks. This may run faster on some systems, but the Run Orbit mode is generally more useful in real operation. Of course now what happens? Well the objects could fly around each other, or they could crash into each other, or they can fly off the screen, or start off the screen entirely. It becomes very important to control where we see the system from . This is an issue I have not seen adequately dealt with elsewhere. For example, if the objects go flying off the screen, it is a bit of a tedious task to rescale the image size so that we may see everything again. Let us look at two areas of control of the visual display. The first is another dialog box; the second is during runtime.

Again under the Options menu, within the Edit Universe submenu, is the Eye Attributes selection. This is more easily accessed by pressing □-E. The Viewing Attributes dialog box will appear. During runtime, this program always provides an image of the system calculated in true perspective. No simple projection of xy,yz,xz coordinates or "move up if its farther back" here. I find that far too difficult to see for the purposes of this program, simply because that is not what we do in real life. I try to make use of the natural skills we have developed between our eyes and brain.

The simulated eye in the program is situated in the space of the system. Before computation begins, all object positions and velocities are realigned so that the center of mass is at the origin and the velocity of the center of mass is zero. The eye is always pointed straight at this origin. The position of the eye may be specified in the Viewing Attributes dialog. Again, the eye position is specified in the altitude-azimuth-radius coordinates as for all the objects. Imagine we are adrift in space above these objects. The main display window is a portal out of some craft we are in. There will be an angle width that we can see through the portal. This is specified by the Viewing Angle attribute, from the left side of the window to the right. 45 to 60 degrees is the standard normal lens for most SLRs. You usually would want to leave this number alone. The Viewing window width and height specify the coordinate size of the window on the screen. Trail length indicates how long the trail of computations we should see on the screen before the program erases the dots. A length of 1 or 2 makes it look like a real object. Longer

can give you more visual velocity information. Specifying -1 will tell the program not to erase any trails. The Display Sizes check box will tell the program too calculate and display sizes of the objects in perspective. This is where the radius information of the objects become very important. With this checked, and appropriate radii in the object specifications, the objects displayed will be drawn as small squares whose size will depend on the distance from the eye and the specified radius. When dealing with three dimensional display, I make use of the natural skills of visual clues. An object coming closer will appear to expand in size. Going away will make its appearance shrink. I do this to the best of the computation speed and screen resolution of the hardware. Just watching the action explains it all. We should look at what happens during runtime, and the purpose of the rest of the controls in the Viewing Attributes box becomes clear.

When the program is running, it is very easy to have the objects fly out of view or their positions become confusing. During run time, even when no objects have been entered, try holding down the 0 key on the numeric keypad. A cube will appear, or at least its corners. For the first time, you may be viewing it face on. Holding down the 0 key with your thumb, try pressing the 4, 6, 2, or 8 keys. The cube should appear to whirl around depending on the key combination. This cube remains absolute relative to the system of objects and coordinate system. It is we who are flying around the system. Now let us say that the objects fly away or become very bunched together at the center. Just hold down the 0 key to see the cube again and tap the + or - keys on the right side of the keypad. The cube and system will zoom in and out accordingly. Obviously the cube could zoom out of view, so the program automatically adjusts the cube size while you are zooming in or out. You may also zoom in and out by holding down the 1 key and tapping the 2 or 8 keys. Zooming and rotating may also occur without the cube seen, i.e., without holding down the 0 key. Try playing around with this control. I find it a very intuitive way to see what is going on, especially for the three dimensional systems.

Looking back at the Viewing Attributes dialog, we can address what all the other options do. The Reset Dimensions ... to Screen calculates dimensions of the Viewing Window so that it almost fills the screen but leaves enough room for the Control Window. The Use file H&W box means that it will load height and width information from the files it loads. If the Rotation Speed Governor box is checked, the speed of the rotation will be governed to about 5 degrees per 60th of a second. This option is primarily for hardware such as a Quadra 700 or greater, where the cube could spin *too* fast. The Auto Zoom In and Auto Zoom Out check boxes mean exactly that. If the Auto Zoom Out box is checked, the program, when running an orbit, will automatically make the cube appear and zoom out if any object ventures out of the display window. If the Auto Zoom In box is checked,

the program will light the cube and zoom in if all the objects are within a rectangle one tenth the dimensions of the display window.

Wait a minute, there are two more, you say. Yep, they are In 2 Color 3-D and Eye radius. This is when we say "Put on your three-D glasses, now." Use the paper glasses with the red filter on the right eye and the cyan filter on the left eye. With the In 2 Color 3-D option on, the program simulates two eyes, each Eye Radius percent of the distance from the origin apart in the space. It is best to sit back in your chair while watching the screen. Try this option on while lighting and rotating the cube. The cube and the system of objects should appear just in front of the screen over your desk. It is very important not to get closer, for that will reduce the effect. Also best results are when you are viewing centered on the screen. For those of you that use corrective lenses, I put my glasses on first, then slip the 3-D filters and cardboard underneath the lenses.

What to do while the acrobatics are going on

There are a few nifty things I have added on to the program while it is running. In the control window, the program will display certain status information at regular intervals. The Δt/sec indicates approximately how many computations occur per second of operation. This can be a good benchmark for speed of computational method as well as hardware. The little picture of something like a stopwatch with a hand that goes around is, well, just some kind of thing I thought would look neat. On the next line, the text will say Running or Paused or be blank. If it is blank, no computations are occurring. Running means it is running through its calculation loop. Paused means it is running but all computations are paused. All display functions such as the cube, rotation, zooming in and out, and so on, will still be alive and active to your keypresses. The program is paused and unpaused by a tap on P, clear (on the numeric keypad), or pause (F15).

One thing that is mostly hidden is the stats option. Press S to activate or deactivate the stats display of any object you choose. The same can be done by clicking on the middle left of the Control Window. There an energy bar will appear, as well as numbers on the right of the bar. Choose an object by clicking near the object in the Display Window. The bar will take on the color of that object. The numbers on the right indicate velocity in meters per second, kinetic energy per unit mass, potential energy per unit mass, and total energy per unit mass in Joules per kg. Below the energy numbers is a number indicating the simulated elapsed time since initial conditions in hours. The energy numbers are independently selectable from the rest by clicking on them, as they take up a consider computation time. The entire energy bar selectable because it too can take up precious computation time.

The calculation of the energy stats are quite straightforward. The speed is merely the magnitude of the velocity vector of the object in question. The KE/m comes from $v^2/2$. U/m is calculated by effectively integrating the force function of each object from the selected object and superimposing the net effect. The resulting coefficients in front of the real integration is fully accounted for. E/m is merely the sum of KE/m and U/m. This translates to the energy bar. The format of the energy bar is as follows. Potential energy is considered first. In general for ß < -1, potential energy ≤ 0. This can be interpreted as a pit or well of energy, which this bar attempts to convey. Kinetic energy is always positive. This energy is read up from the bottom of the energy well. Then we can easily see that all objects have conservation of energy unless they are pulled by some other force or object. The bottom of the well may vary but the sum should be constant, unless we have a bad numerical method. For ß ≥ -1, potential energy is positive. The energy will read up from a zero energy line. The kinetic energy is again read up from the top of the potential energy, again indicating total energy graphically. Of course, these energies may range all over the place, so I have the program automatically adjust the energy scale when it becomes too bunched up or attempts to extend outside the window. It is possible to confirm examples of the virial theorem with this graph. It is easier to see when one achieves a circular orbit with an object.

A fun thing you can do - The Craft

Wait a minute, you say, what about that thing Edit Craft and Autopilot? Here your questions are answered. Now you too can actively participate in the events unfolding before your eyes, making each run unique and truly your own. At our disposal is one more object, which I call the Craft. Set options for it by selecting under the Option menu, within the Edit Universe submenu, Edit Craft or press □-C. As you may see, it has almost all the options available to all the other objects. Almost, but more. Clicking the Use Craft check box selects whether or not to have the Craft participate in the events. Color selection is not available; the craft is almost always white. Again almost? There is a thrust edit text item here. What does this mean? you may ask. Heh heh heh...

This is one object that has thrusters on it. I got this idea about four years ago while visiting the Exploratorium in San Francisco. They had a computer hooked up to a monitor and a few buttons. They had a simulation of a satellite with thrusters going around a simulated Earth. After I played with that for a while (emphasis on *while*), I said I wanted this on my computer. So when I got home in Huntington Beach, I eventually programmed it on my Apple //e. The left and right arrow keys thrusted against and into the direction of motion, respectively. I have carried the idea here in this Macintosh

version, with the same buttons. The color of the craft will turn red when thrusting forward and orange when thrusting in reverse. In this version, we can change the amount of thrust per time step the keys command in the Edit Craft dialog box. With this it is possible to observe what happens to the energy and eccentricity of the orbits when I thrust the craft back and forth. I played around with it a lot back then, and gained much experience. However here the version is much better because it makes a sound whenever you turn the thrusters on!

You can really get to know what happens when you use those thrusters. Thrusting up will add energy to the craft, but the orbit on the opposite side will raise in altitude. Similarly, thrusting down on one side will cause the orbit to lose altitude on the other side. I know this well enough now so that it is fairly intuitive, but I think I can remember a time when it wasn't. (Hidden option: holding down option while pressing the arrow keys increases the thrust by ~10 times. Can be useful in a jam.)

One of the fun things you can do is thrust up and down to try to get near another massive moving object. There is the danger of getting sucked down and crashing into the object, but you can eventually get a gravitational assist and get flung far out of the system. Be careful when trying to slow down though. It becomes easier to drop to zero velocity as you get out there. If you do get zero velocity, you could drop right back into the central planet. Thrusting will only be radial in this case. You might be able to get a sideways pull if you can get close enough to an orbiting moon.

Through experience I have learned how to easily achieve a highly elliptical orbit or a circular one from almost any other orbit, but getting the computer itself to do that took a little longer. One option on the Exploratorium computer program that I saw was to flick a switch and it would turn on its autopilot option. That program would thrust up and down until it had a circular orbit. I tried to implement that on my Apple //e, but I did not figure it out until this year. Clicking on the word Autopilot in the Control Window or pressing A on the keyboard toggles the Autopilot option. This routine works under specific conditions. There must be some massive central force near or at the center of mass of the system, the force must be the inverse square law, and it must already be in some kind of elliptical orbit. Outside these bounds the routine may yield unpredictable results.

The way the Autopilot works is as follows. At the moment of toggling on, the program remembers the current radius of the craft from the center of mass as the target radius. When the radius and velocity are nearly perpendicular, the Autopilot begins to calculate. This is pretty much the way I would achieve a circular orbit by hand so far, but a human can see the entire ellipse. I had to find a different way for the computer. When

radius and velocity are perpendicular, the orbit is at apogee or perigee. Here we calculate the radius of the opposite side. Using previously derived relations of energy, angular momentum, semimajor axis, and eccentricity for elliptical orbits in a Newtonian gravitation field, I was able to derive an expression for the eccentricity as a function of the variables available at this point in the program under these special conditions. Basically, the program finds a value for the eccentricity and uses it to find the radius of the opposite side of the orbit. If this predicted radius is less than the target radius, it thrusts up; if the predicted radius is greater than the target radius, it thrusts down. After each time step and the position and velocity are close enough to perpendicular, it recalculates and predicts the opposite radius and makes the same judgement on thrusting. The same happens on the opposite side. This continuous checking occurs until the predicted and target radii are with in two percent of each other. The code will then try to maintain this circular orbit for as long as possible, continuously maintaining it with appropriate thrusting, to the best of its ability. Nearby orbiting moons may tend to throw it off, but it usually does a pretty good job. Try setting up a very elliptical orbit and press A when it is somewhere in the middle of the orbit and watch it go.

One of the more recent additions involves another computer controlled thrusting. Some time ago I set up some initial conditions that had an Earth at the center with two moons around it. I set up the craft in a parking orbit around the inner moon. I thought I could achieve a transfer orbit by hand. The first tries were encouraging. I was able to get the right thrust at the right time to get it out to about the right radius from the earth. Then I thought all it would be was a matter of timing to get it to go around the outer moon. I tried and tried, but I just got too tired. So I gave up for a while. The simulation was going by fast enough for it not to be boring or tedious, but to get it a the right time again and again was just too difficult. In a real system, the orbits would take hours to pass, so it would be much easier to get an exact thrust time relative to the system. A few days ago I implemented timed thrusts. Under the Edit Craft dialog box, you can edit a linked list of delta velocities and time indexes. The Δv is in meters per second. The time index is in number of computations since initial conditions. I saved a file set up where the thruster fired a forward Δv and injected the craft into a transfer orbit to the outer moon. When it got to the right location/time index, it fired a negative Δv to achieve a parking orbit around that moon. The filename is "TrnsfrOrbs - It Works!". I found the values by intuition and trial and error. I am envisioning a trip around the Galilean system I could try. Maybe simulate Galileo's voyage into the Jovian system to visit each of Galileo's moons. Wouldn't that be fun to try!

Other things I forgot to mention

If you set up an arrangement of bodies, initial conditions, timed thrusts, and the like, you can choose Save Universe from the File menu. I have over two dozen saved files with this program that you can base other experiments on. Just choose Load Universe. (I love playing god. heh heh)

Under the Action menu, there are presently three options. The first is Realign Inertial Reference Frame. This calculates the center of mass and its velocity and modifies all the position and velocity values to zero the center of mass and its velocity. Also invokable by pressing ⌘-I. It only processes the working values. The initial conditions you create are always stored in a separate area. The second menu item resets the system to initial conditions. It copies all the information from the separately stored area to the working area, then zeros the CM and velocity of the CM. The third item copies the present conditions to the initial conditions. This copies all the working data to the separate area. This does modify that other area, so be careful.

Press ⌘-A or select About under the apple menu to get a little about box with a nice picture in it. I created this picture as well as the application icons and file icons with my raytracer I began writing on and off a few years ago.

I wrote this program in THINK Pascal V4.0, a very nice environment. The legalisms say I have to say that portions of the compiled code are under their copyright, so there. The entire code is 3878 lines long or ~ 58 pages, so please forgive me that I don't have a hard copy. Most of it involves the Macintosh interface, loading and saving files, event managing, distribution of tasking, resource managing, and so on. I have written it so that it can run in the background under System 7.0.x. The code involved with solving and displaying the solutions is about 1600 lines. Known bugs: some weird things happen on the Quadras when I try to do things as simple as copy numbers from one area to another. I can get around by reselecting Reset to I.C.'s after I begin the Run Orbit loop. I haven't found any other problems with this version.

Credits

My thanks to those that gave me encouragement and watched this program grow. Specifically those that sort of "Beta tested" it: Wyatt Riley, Ben Weiss, and my father. They helped by giving me constructive criticism on things they did and did not like in the program. From time to time Wyatt and Ben said "cool!" as well. Also thanks to the Exploratorium in San Francisco, which planted the seed inside me to make my first orbiting program on my Apple //e. That one was very primitive compared to this one; it could only do two-dimensions and did motion calculations for only the craft and nothing

else. Also I had some inspiration from the Mechanical Universe series from CalTech. The television episode of the series entitled <u>Navigating in Space</u> is extraordinarily good at providing an intuitive understanding transfer orbits, getting out of parking orbits, and the like. It specifically gave me the idea of the energy bar. I also referenced the associated book to the series, listed below. I wish there were more programs and places like the Mechanical Universe and the Exploratorium, for they gave me some of the greatest enthusiasm for my interest in physics and science.

Reference

Olenick, Richard P., Apostol, Tom M., & Goodstein, David L., <u>The Mechanical Universe</u>, Cambridge University Press, New York, 1985.