Problem: To develop a dynamic, expressive set of interactions for particles/drones capable of many transformations in a real-time updating environment.

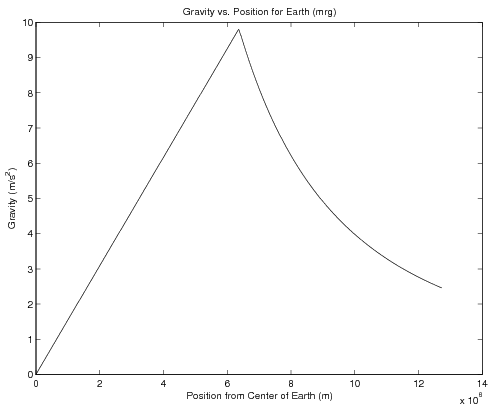
Natural interactions; new degree of interesting control; live updating; can later incorporate simulations of drone flight and fluid dynamics

Initial suggestion was to simulate a gravitational field, as I have a fairly comprehensive understanding of it and non-physicists have a good idea of what it looks like; Has simple ability for tight or large orbit, different eccentricities.

In order to make it random and more representative of a ‘field,’ particles would be given random initial positions and velocities. This would later need to be restricted in order to avoid particles immediately reaching escape velocity.

We started in 2D using an animation script that plotted the position of the next time step and deleted the plot of the position (XX) time steps ago. The positions were plotted as asterisks, and chains of asterisks began to look like ‘caterpillars,’ hence the nickname.

First problem involved an uncontrollable slingshot effect. Gravity source was initially modelled as a point particle such that the force acting upon the caterpillars was equal to F=-mag/r^2 at any distance. This posed a problem because the acceleration would then approach infinity as the radius approached 0. In a real physical system, this would be acceptable as time steps are infinitely small and the ‘infinite’ force would not act on the object for very long. However, in our model, time steps were finite, and thus by the time the object passed the origin, its infinite acceleration would move it far out of frame. In order to account for this, the gravity source was modeled as a planet of constant density (similar to that of the Earth). This solved the problem as the particle no longer felt any force at r=0; The particle experiences linearly increasing acceleration until it reaches r=radPlanet, at which point the planet can be treated as a point source with F=-mag/r^2.



The second problem that arose was observed after the code was initially presented. After several hundred time steps, the particles would seem to move faster and faster with each pass of the origin until they eventually flew out of frame. This was strange occurrence, as the system seemed perfectly even and only considered conservative forces. The problem was discovered in the time delay of the forces. Initially, a calculation of force wouldn’t affect the position of the object until two time steps later (as demonstrated below).

|  |  |  |  |
| --- | --- | --- | --- |
| **k** (time step) | 1 | 2 | 3 |
| **a** (acceleration) | a1 = F(x1) | a2 = F(x2) | a3 = F(x3) |
| **v** (velocity) | v1 = rand() | v2 = v1 + a1\*Δt | v3 = v2 + a2\*Δt |
| **x** (position) | x1 = rand() | x2 = x1 + v1\*Δt | x3 = x2 + v2\* Δt |

Where F(x) is the force calculation at a position x; Δt is the length of time step.

The calculations were initially set up this way because it follows closely to the classical description of movement. a1, v1, and x1 would all represent the instantaneous values at any time t=1. However, also in real life, the time steps are infinitesimal and calculations follow differential equations instead of additive functions (this is impossible to mathematically model according to the three-body problem). A problem then arises as follows.

If one follows the flow of information about a position x, it is clear to see that the acceleration it causes does not again affect position until two time steps later.

|  |  |  |  |
| --- | --- | --- | --- |
| **k** (time step) | 1 | 2 | 3 |
| **a** (acceleration) | a1 = F(x1) | a2 = F(x2) | a3 = F(x3) |
| **v** (velocity) | v1 = rand() | v2 = v1 + a1\*Δt | v3 = v2 + a2\*Δt |
| **x** (position) | x1 = rand() | x2 = x1 + v1\*Δt | x3 = x2 + v2\* Δt |

Where 🡪 denotes the ‘affect’ relation.

This causes a clear problem when the object passes x=0 and its acceleration is reversed.

x=0

The circles represent positions at sequential time steps (denoted by the numbers). The arrows represent the direction of acceleration that is acting on the position at the time. We can see that at t=3, the object is still accelerating past the point of origin. This continues until t=5 when the acceleration vector finally (after two steps) affects the position. This delay means that, at t=3 and t=4, the particle is gaining excess acceleration for every pass of the origin, thus slowly building up the velocity and kinetic energy of the particle and, as a result, allowing it to reach farther distances from the origin until it moves out of frame.

According to the specifics of the problem, our solution is limited. We cannot have a change in acceleration immediately affect the position at the same time step. This is because the position is necessary to calculate the magnitude of acceleration and thus cannot directly affect itself. Therefore, the smallest delay we can design is to have the acceleration affect the following time step. We can do this by having the acceleration affect the velocity within the same step, and subsequently updating the next position with this velocity.

Solution #1

|  |  |  |  |
| --- | --- | --- | --- |
| **k** (time step) | 1 | 2 | 3 |
| **a** (acceleration) | a1 = F(x1) | a2 = F(x2) | a3 = F(x3) |
| **v** (velocity) | v1 = rand() + a1\*Δt | v2 = v1 + a2\*Δt | v3 = v2 + a3\*Δt |
| **x** (position) | x1 = rand() | x2 = x1 + v1\*Δt | x3 = x2 + v2\* Δt |

The second solution I proposed was to estimate the next position according to the current velocity, use this to calculate the next acceleration, and have all the values update within the same time step.

Solution #2

|  |  |  |  |
| --- | --- | --- | --- |
| **k** (time step) | 1 | 2 | 3 |
| **a** (acceleration) | a1 = F(x1) | a2 = F(x’2) | a3 = F(x’3) |
| **v** (velocity) | v1 = rand() + a1\*Δt | v2 = v1 + a2\*Δt | v3 = v2 + a3\*Δt |
| **x’** (est. position) |  | x’2 = x1 + v1\*Δt | x’3 = x2 + v2\*Δt |
| **x** (position) | x1 = rand() | x2 = x1 + v2\*Δt | x3 = x2 + v3\* Δt |

Both of these solutions were written up and tested, and an interesting relationship between the two was found. Solution #1 was a very good model, allowing the particles to reach stable orbits for many time steps. As the length of time step was exaggerated, a problem similar to the original arose, with the particles slowly gaining KE and drifting out of frame. However, the magnitude of increase was very insignificant in comparison. Solution #2 had the opposite problem. Over time, the particles would gradually lose KE and fall into the gravitational well at the center. This is probably as a result of the underestimation of position upon the particles’ approaches to the gravity source. In whatever regard, what is interesting about the two solutions is that, as the length of time step decreases, their problematic effects decrease and they model the same results!

Despite the fact that both solutions worked much better than the original and even though Solution #2 would theoretically be a better model to real life, we chose Solution #1 as it was more accurate than Solution #2 at higher time step lengths (on the level of step=0.01 at which real-time animation will look realistic and interesting). We also postulated ways to counteract the gradual KE gain, one suggestion of which involved a friction field that turned on at a certain distance from the origin, forcing the particles to lose energy if they gained enough to reach that point.

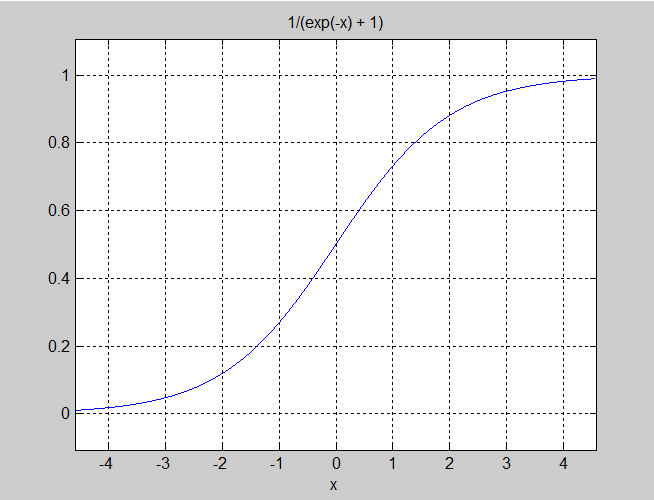
In response to the KE problems, I wanted to design a mechanism to keep the particles within the field of view of the graph to confirm that they were gradually gaining speed. The ‘Bounce’ function was designed to have the particles ricochet at the boundaries of the plot. This was done simply by reversing their x or y velocity if their position was outside of a certain range. This would ensure that the objects would be moving at the same speed (with the same KE), just in another direction. This would eventually lead, in addition to the KE problem, particles bouncing throughout the plot in linear motion, with little regard to the gravity well at the center. They would have gathered enough energy to overcome the gravitational acceleration and maintain their linear velocity.

This ‘Bounce’ function would later pose problems, especially when it was integrated with the ‘particleDeflect’ function. If a particle extended beyond the boundaries of the plot and had a negative acceleration (which occurs whenever the object moves away from the origin), its next calculated position would not be within the plot’s boundaries (less velocity and thus less distance step). This would, according to the script, signal another velocity reversal, forcing the object to bounce back and forth with constantly reversing velocity outside of the plot boundaries for the rest of the trial. This would later be resolved by specifying that the object needed a velocity vector directed out of the plot in order to be subject to velocity reversal.

Around this time in the project, we began to look at applications beyond that of the drone control/performance. One suggestion detailed using Kinect recordings to control direct and expressive features of the visualization in real time. In order to demonstrate this, I designed a feature that would move the origin of the graph at a set time in order to see how the particles would react. It worked very well and, according to the nature of the algorithm, many other features could simply be changed mid-run of the script and the particles would immediately respond. As the origin moved, however, the KE of particles would randomly increase or decrease, so some other feature may be needed later in the project to keep them under control (dynamic friction field).

We also began to look at different representations and field forces to see if they would be useful in different expressions. The next simple model suggested was that of an ideal spring (F=-kx). It could use the same distance calculation as the gravitational model and also naturally eliminate the ‘slingshot’ effect, as the acceleration approached 0 at the origin. When this was coded, an interesting interaction occurred in which the particles reached Simple Harmonic Motion as is theoretically predicted by physical models. This means they follow the same path of motion, continuously oscillating between two extreme positions in a sinusoidal fashion with respect to time. Since the oscillations were fairly constant, the visualizations became fairly boring after a while, as opposed to the orbits of the gravitational model which were generally chaotic in nature.

This prompted my mentors to ask if these were representative of real springs, which of course, they are not. Real springs will stretch and compress irregularly, and even break at long distances. They are similar in modeling to the function F(x) = n/(m+e^(-cx+h)).



This is because real springs are known to follow Hooke’s law at small displacements, but distort and stretch at their extremes.