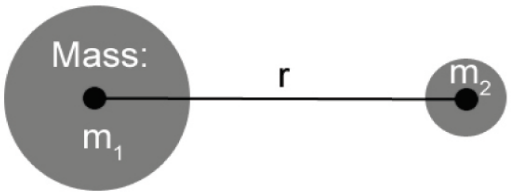
About NBody Simulations

Jack Carrozzo
http://www.crepinc.com/

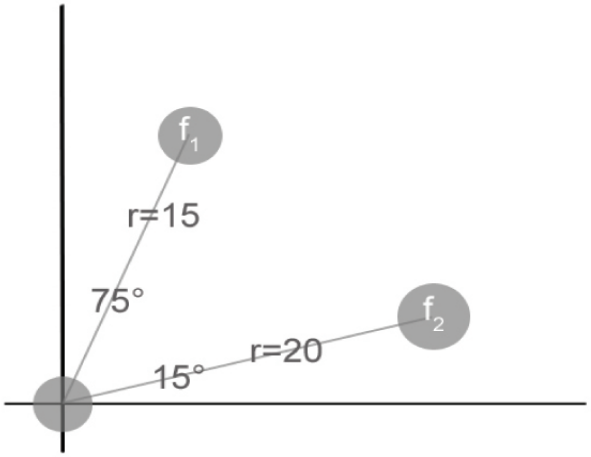
Assuming you haven't been living under water for your entire life, you've probably noticed gravity. It's quite helpful really, in that planets don't simply fall from the sky, your water stays in your glass, etc. But the best part is, it's highly predictable.

Any "body", or "solid mass" as it were, creates a force upon everything else in the universe. Generally a given body you might see around is too small to make much of a different to anything, but the force is there nonetheless. Thus, we can find the force on any body by simply evaluating the equation for gravity for each object it interacts with, then sum the forces.

The force in one dimensional is: $F = \frac{Gmn}{r^2}$ where G is the gravitational constant, **m** and **n** are the masses of the two objects, and **r** is the distance between them.



Remember, this is only one-dimensional. If we have more than two bodies, we need to think of the forces as vectors; that is, 20 units of force at 30 degrees is only 10 units in the **X** direction. So, in order to convert from vector or polar form to rectangular, we simply need to multiply the distance **r** by the cosine or sine of the angle: $\vec{F} = Rcos(\theta)\mathbf{i} + Rsin(\theta)\mathbf{j}$.



Here we have two force vectors. To add them, we simply sum the forces in the **X** direction and the forces in the **Y** direction:
$\vec{F}_{sum} = \vec{F_1} + \vec{F_2} = (R_1cos(\theta_1) + R_2cos(\theta_2))\mathbf{i} + (R_1sin(\theta_1) + R_2sin(\theta_2))\mathbf{j}$
$(15cos(75) + 20cos(15))\mathbf{i} + (15sin(75) + 20sin(15))\mathbf{j} = 23.2\mathbf{i} + 19.66\mathbf{j}$

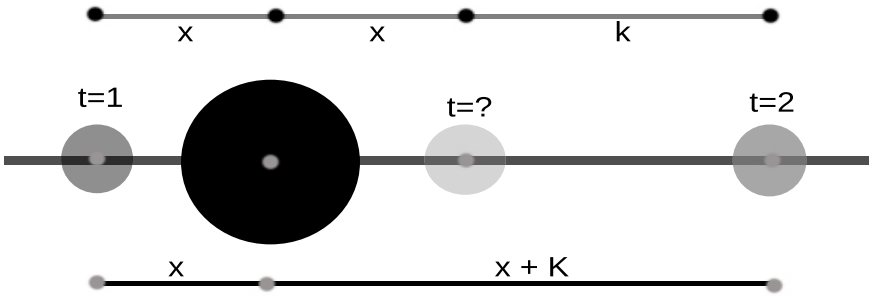
Now that we've found the forces on a body, we need to change it's speed and move it. If $F = MA$ (force equals mass time acceleration), then $A = \frac{F}{M}$. So far, we've found that $A_x = \frac{\Sigma F_x}{M} = \Sigma Rcos(\theta)$ and $A_x = \frac{\Sigma F_x}{M} = \Sigma Rcos(\theta)$, or "acceleration in the X and Y directions equal the sum of all forces in the x and y directions divided by the mass of the body the forces are acting upon".

We're getting somewhere now!

So assuming one has a stored list of their bodies and their properties in a format such as BodyPosX, BodyPosY, BodyVelocX, and BodyVelocY, it's a simple matter to itterate through and incrememt BodyVelocX and BodyVelocY by A_x and A_y. Congradulations! You've just succesfully evaluated and stepped an NBody evaluation. Simply repeat this and output the positions of each of the particles and you've written all you need to evaluate very basic arrays of bodies.

Have you noticed a potential issue yet? Try running the program, or looking at the video on my website. Everything comes together nicely... but where do they go? If you look closely, they are in fact shooting off in every direction at high speed. Is that what happens in reality? I think not.

Take another look at the force equation: $\frac{Gmn}{r^2}$. Yes, that's a fraction with a variable in the denominator. Thus, as the distance R between two bodies decreases, the force increases such that $\lim_{r\to 0} \frac{Gmn}{r^2} = \infty$. Think about this for a second. In real life, you can think of physics taking place as a "realtime simulation", or a simulation where is time between timesteps is zero: the forces are constantly evaluated and used to affect the body. But here, we must set how much time passes between each recomputation of our values. During that time, it is assumed that the particles and forces remain constant. That is of course not true, but when objects are more than a few units away, it does not show a noticable difference. Yet, as they get closer, the timestep needs to be smaller to be accurate. As the particles get closer than 1 unit, the force at one step will be so much that by the time the next step comes around, the particle is already too far away from the original body it reacted with, so the force does not slow the object down and counteract the very high force it got in the original timestep.



As you can see, since the distance on both sides of the larger body is not the same, the force will be greater on the left side. Thus, while the body (well, point-mass...) SHOULD go straight through the larger one and leave going the same speed at which it came, it willl actually be going much faster. That is why a calculus approch may be better for this sort of thing. However, in our case, we've already created quite an efficient NBody solver. Why not find a way to dampen the extra energy and keep the body around?

We have two options here: if a body comes within 2 units of another body, we can either "absorb" this other body, that is sum the two particle's masses and count them as one, or allow the second particle to pass through the first, and simply no evaluate the forces if the distance is very small, assuming the the particle will go straight through.

Why not both?

Personally, I chose a hybrid method wherein we choose between these two methods depending on the particle's incoming speed. If over a certain threshold, allow it to pass through, otherwise simply absorb it and sum the masses. You'll have to play with the threshholds a little bit, but you will eventually find one the suits your application.

I hope that by now I've given you a good taste of how NBody solvers work and how easily you can write your own. Of course, commercial solvers that run with MPI and use numerical methods to better aproximate large sets work better and faster, but hey, you wrote your own! My own code can be found on my website at: http://www.crepinc.com/projects/nbody for reference. I would love to hear any comments you may have!