Computational Methods Fall 2017 Final Project Outline

Goal: Few/many-body Newtonian Gravity Simulation + Visualization

I. Simulation (Python)

a. Basic Goal: Simulate a few-body system (using the solar system as example)

1. Start by defining planetary bodies along with their initial three-dimensional positions and velocities IN A DICTIONARY FOR EACH BODY

2. Repeatedly step time forward, at each step recalculating positions and velocities in rectangular coordinates of each body (Only counting gravitational influence from the Sun and Jupiter).

i. for EACH body: Calculate distance/mass from Sun and Jupiter, plug in to 2nd order eqns. of motion, step forward using Velocity Verlet Method

ii. record results for each step in a DICTIONARY FOR EACH BODY, containing:

A. time(list)

B. mass(single value)

C-E. XYZ position(each a list)

F-H. XYZ velocity(each a list)

I. acceleration magnitude(list)

3. After "enough" steps (will require testing for stability of system), final output will be the matrix of times/positions/velocities

II. Visualization (Maya Animation and Modeling Software) -- ***THIS GOAL IS COMPLETE***

a. Maya's coding script (MEL) provides bridge from python code to animation

1. Need to write a python code which can take the output of the simulation (the times/positions/velocities dictionaries), uses the times and positions to write a MEL script to place points at their given positions at certain frames

2. Code will also need to "key" bodies/frames (connect bodies in one frame to themselves in other frames to animate them through time - the animation process should render the velocities unnecessary)

3. Maya animates the movement of the system and outputs an mp4 file to present to the class

III. Stretch Goals

a. A more general few-body simulator

1. Incorporate gravitational influence from ALL other bodies, not just the Sun and Jupiter - longer eqns. of motion, more distance calculations.

2. Introduce other bodies - asteroids, comets

3. Apply to a more arbitrary few-body system (bodies with starting masses/positions/velocities different from that in our system)

b. The jump to many-body

1. Adaptive time stepping seems to destabilize Verlet method. Possible to implement by switching to other method for N-body, where cyclical behavior would be less likely?

2. Explore a way to implement a Barnes-Hut tree (or something similar) so that this program doesn't take another semester to run

3. Scale factor for super optimistic GR implementation. *Sounds* simple, but how to include?

4. Regularization. Looks horrible.