Simulation of Planetary Bodies in the Universe (N-Body)

New Mexico Supercomputing Challenge

Final Report

Team 21

Desert Academy

Team Members

Salvador Brandi

Damian Browne

Cameron Mathis

Project Mentors

Jeffrey Mathis

Jocelyne Comstock

Carolyn Brandi

Executive Summary

In Newtonian mechanics, the N-body problem is the problem of of predicting the motion of astral bodies with gravity. Study of the N-body problem involves ideas of classical mechanics such as Newton's law of universal gravitation, inertia, mass and velocity of astral bodies, Kepler's laws of planetary motion and the inverse-square law. Our solar system is an N-body system. N-body simulation is the simulation of astral bodies under gravity, using laws of classical mechanics to define how the astral bodies move.

The goal of our project is to model the N-body problem in NetLogo. Code defining how astral bodies interact implements the inverse square law and a gravitational constant to calculate gravitational force between them. Our project has not generated numerical data yet because, our model of should be finished before testing is done and we feel ours isn't there yet.

Problem

For a very long time physicists have toyed with the idea that with enough random instances of particles following physical rules, usually the law of universal gravitation, inertia, and collisions, the chance of a random orbit could be high, similar to how our own universe started. Our team researched the N-body problem and sought to recreate it, using netlogo. We wondered, how many trials could it take to create a near-perfect orbit? What variables could we manipulate to get us there faster? This model shows our take on the N-body simulation using the inverse square law and force and velocity variables.

Method

Our model was made entirely in NetLogo due to its strong agent based approach. The source code is available in Appendix B at the end of this report. We also used a two dimensional setup as opposed to a three dimensional approach.

On the setup screen are sliders that can be manipulated to slightly change the model's behaviors. The "number" slider defines the number of turtles spawned when the "Setup" button is pressed. "Base_mass" defines the starting mass of "planets" (the default breed), and "stars,"."Stars" are stationary "planets" with one hundred times the "base_mass." "Path_fade" defines how quickly paths, which are created by "planets" as they move, will fade. The "Star?" switch defines whether there will be "stars" in the simulation.

The model is only displayed in NetLogo's "World" but, we use turtles-own variables "xc" and "yc," which are used for a turtle's true coordinates. D_sqrd is the square of the distance between two planets, used in the inverse square law. Another set of turtles-own variables are "fx" and "fy" which determine the x and y component of the force vector and is defined by the inverse square law:

```
set fx ((cos (atan (c_yc - yc) (c_xc - xc))) * (base_mass / d_sqrd))
set fy ((sin (atan (c_yc - yc) (c_xc - xc))) * (base_mass / d_sqrd))
```

"Vx" and "vy" define the velocity of a planet based on its force and the gravitational constant given above:

```
set vx (vx + (fx * g))
set vy (vy + (fy * g))
```

Our global variables consist of: “c_xc” and “c_yc,” “g,” and “w_var.” “c_xc” and “c_yc” determine the x and y coordinates of the “closest-planet” which is determined by: “let closest-planet min-one-of turtles [distance self].” “G” is the gravitational constant being 6.67E-11 in our universe, but we have defined it as .5. “W_var” defines who to ask for determining the force of gravity. This variable is a replacement to “c_xc” and “c_yc” but, whether this is more accurate than “c_xc” and “c_yc,” is unclear.

Originally we approached the N-body problem by only using Newton’s Law of Universal Gravitation, which only made all the planets clump up into a ball, which is not entirely accurate on the grand scheme of things. At the time of publishing this paper, we have not conquered the problem we have when two planets “collide.” In most cases, when two planets have the same x and y coordinate they “slingshot” off the screen, never to be seen again.

Verification and Validation

Even though our model is not entirely accurate, it recreates with graphical simplicity and mathematical correctness of the N-body simulation. Through many many trials, we realize that normal orbits are incredibly complex and hard to obtain through any normal means, and causes us to conclude that our own solar system is an incredible anomaly of the universe, and verifies why the universe existed for so long without star-planet systems or large galaxies.

As the aforementioned conclusion was the one we expected, The model was successful in serving its proper purpose, and has helped us to better understand information we know, leading our team to believe that our project was very helpful in achieving our desired effect, and modeling what wa required.

Analysis and Conclusion

Our model is a partially accurate representation of the N-body simulation that has representations of how astral bodies with gravity interact, but is missing factors. The tendency of planets to attract slowly at first and then very quickly as they near suggests correct implementation of the inverse square law to make planets gravitate towards each other, and the way they interact in this matter is reasonable. They also have the tendency to accelerate away from one another at an unreasonable rate upon collision, which is an emergent pattern that was unpredicted or is a result of error in code. Additionally, planets approaching the sun accelerate rapidly as earlier mentioned, after their apparent contact with the sun, which suggests As orbit hasn't been observed in the model, fundamental problems related to how the planets interact may be present, or orbit is an emergent pattern observed only when full detail of the n-body problem has been accounted for and our model is not extensive enough.

Reviewing the current model opens windows for future study. The next thing to do with the model is to further research how orbit fits into the n-body problem, and to look at other models in NetLogo that simulate these ideas so that we may identify what ours is missing and why we have not seen orbit. More research into real n-body systems could be done to further check our model to identify what additions are needed. Additionally the model could be expanded by accounting for the event of a collision, where two planets might break into pieces or combine masses, or any combination of potential outcomes. The model could also be taken in a different direction by modeling our solar system using the current model's code. Our model serves as basics that could be used for additions or further models.

Most Significant Achievement

Throughout this year's project, we struggled with many code based as well as team based problems. Our team faced hurdles, most notably math beyond our level, code languages we were not familiar with, and ideas we could not fully understand. However, we remedied these situations using the varying amount of resources we had on hand, such as consulting others, and putting the necessary time into it ourselves. One of the largest things we accomplished this year was implementing multiple equations and overall physical mechanics accurately into our code. It required our team's time, much brainstorming and more than a few meetings in to achieve

Beyond even this, one of our biggest achievement was learning how to coordinate work, stay on task, and just generally work as a team. Throughout this year, we learned the hard way, wasting valuable time and putting us behind. But because of this, our team learned the value of working together and doing things well, each team member helping one another with their own, individual hurdles.

Acknowledgments

Although this was largely a project independent from outside help, it's important to acknowledge the advice and expertise of our project advisors, Mr. Jeff Mathis and Ms. Jocelyne Comstock. We also thank Julia Fjeldsted, Mariam Browne, and Carolyn Brandi; their help organizing meetings between the group outside of school was invaluable and essential to getting the project finished. Additionally, Ms. Comstock is our physics teacher, and we thank her not only for physics class introducing to us concepts utilized in our model but for her advice on implementing these concepts. We also want to thank our friend Kara Fischer, who was a member of our team last year and continued to give genius insight and thoughtful advice throughout our project this year that helped to navigate through mental blocks.

Appendix A: References

- Board, John A. Jr (1999), Humphres, Christopher W., Lambert, Christophe G., Rankin, William T., and Toukmaji, Abdulnour Y., *Ewald and Multipole Methods for Periodic N-Body Problems*, in the book **Computational Molecular Dynamics: Challenges, Methods, Ideas** by editors Deuflhard, Peter, Hermans, Jan, Leimkuhler, Benedict, Mark, Alan E., Reich, Sebastian, and Skeel, Robert D., Springer Berlin Heidelberg, pp. 459–471, http://dx.doi.org/10.1007/978-3-642-58360-5_27, ISBN 978-3-540-63242-9.
- Chenciner, Alain (2007), *Three body problem*, Scholarpedia, 2(10):2111, doi:10.4249/scholarpedia.2111

Appendix B: Code

```
breed [planets planet]

breed [stars star]

turtles-own

[ fx     ;; x-component of force vector

  fy     ;; y-component of force vector

  vx     ;; x-component of velocity vector

  vy     ;; y-component of velocity vector

  xc     ;; real x-coordinate (in case particle leaves world)

  yc     ;; real y-coordinate (in case particle leaves world)

  d_sqrd ;; square of the distance to the defined turtle (c_xc/c_yc or w_var)

  t_b_mass ;; turtle's base_mass

]


 ;; run universal gravitation formula on each turtle for each turtle, make a
variable that adds one each time it loops, this variable cannot be the id of
self


globals

[ c_xc  ;; x-coordinate of "closest-particles'" base_mass

  c_yc  ;; y-coordinate of "closest-particles'" mas

  g     ;; Gravitational Constant to slow the acceleration

  w_var ;; variable determining who to ask for gravitation

]



to setup

  clear-all

  set g .5
  set w_var 1
  set-default-shape planets "circle"
  set-default-shape stars "circle"
  create-planets number
  [
    set size 10
    fd (random-float (max-pxcor - 6))
    set vx 0
    set vy 0
    set xc xcor
    set yc ycor
    set t_b_mass base_mass
    set size (3 * base_mass / (4 * pi)) ^ (1 / 3) * (10 / (3 / (4 * pi)) ^ (1
/ 3)) / 2 ;; accurate proportions of the size of spherical bodies and their
mass
  ]
  if Star? = true [ ;; a separate "experiment" to see if orbits are working
  create-stars 1 [
    set xcor -25
    set ycor -25
    sun_settings
  ]
  create-stars 1 [
  set xcor 50
  set ycor 0
  sun_settings
  ]
   create-stars 1 [
  set xcor 0
  set ycor 50
  sun_settings
  ]
]
  reset-ticks
end

to sun_settings
  set shape "circle"
  set t_b_mass base_mass * 100
  set size (3 * t_b_mass / (4 * pi)) ^ (1 / 3) * (10 / (3 / (4 * pi)) ^ (1 /
3)) / 2 ;; accurate proportions of the size of spherical bodies and their
mass
  set color yellow
end

to go
  ifelse w_var > number [
    set w_var 0
  ][
    set w_var w_var + 1
  ]
  ask planets [
;;  let closest-planet min-one-of turtles [
;;    distance self
;;  ]
if turtle w_var != nobody [
    set c_xc [xc] of turtle w_var
    set c_yc [yc] of turtle w_var
    gravitate
]
  ]
;;    fade-patches
    tick ;; might wanna change where this goes
end


to gravitate ;; Planet Procedure
  update_force
  update_velocity
  update_position
  update_collide
end


to update_force ;; Planet Procedure
  ;; Similar to 'distancexy', except using an unbounded plane.
  set d_sqrd (((xc - c_xc) * (xc - c_xc)) + ((yc - c_yc) * (yc - c_yc)))


  ;; prevents divide by zero
  ifelse (d_sqrd != 0)
  [
    ;; Calculate component forces using inverse square law
    set fx ((cos (atan (c_yc - yc) (c_xc - xc))) * (base_mass / d_sqrd))
    set fy ((sin (atan (c_yc - yc) (c_xc - xc))) * (base_mass / d_sqrd))
  ]
  [
    ;; if d_sqrd = 0, then it's at the base_mass, thus there's no force.
    set fx 0
    set fy 0
  ]
end

to update_velocity ;; planet Procedure
  ;; Now we update each particle's velocity, by taking the old velocity and
  ;; adding the force to it.
  set vx (vx + (fx * g))
  set vy (vy + (fy * g))
end

to update_position ;; planet Procedure
  set xc (xc + vx)
  set yc (yc + vy)

  ifelse patch-at (xc - xcor) (yc - ycor) != nobody
  [
    setxy xc yc
    if (path_fade != 100)
    [ ifelse (color = white)
      [ set pcolor red + 3 ]
      [ set pcolor color + 3 ]
    ]
  ]
  [ hide-turtle ]
end
```

```
to update_collide
  let closest-planet min-one-of turtles [
    distance self
  ]
  if c_xc - xc = 0 and c_yc - yc = 0 [
    print "Hello World"
  ]
 ;;  fade-patches
  ask patches with [pcolor != black]
  [ ifelse (path_fade = 100)
    [ set pcolor black ]
    [ if (path_fade != 0)
      [ fade ]
    ]
  ]
end


to fade ;; Patch Procedure
  let new-color pcolor - 8 * path_fade / 100
  ;; if the new-color is no longer the same shade then it's faded to black.
  ifelse (shade-of? pcolor new-color)
  [ set pcolor new-color ]
  [ set pcolor black ]
end
```