N-body final project

Alumno: Cerritos Lira Carlos
Profesor: M. en C. Carlos Gerardo Malanche Flores
Ayudante: Víctor Alfredo Milchorena Gonzáles
Física Computacional

December 13, 2019

N-body problem

In physics, the n-body problem is the problem of predicting the individual motions of a group of celestial objects interacting with each other gravitationally. Solving this problem has been motivated by the desire to understand the motions of the Sun, Moon, planets, and visible stars.

The classical physical problem can be informally stated as the following:
Given the quasi-steady orbital properties (instantaneous position, velocity and time) of a group of celestial bodies, predict their interactive forces; and consequently, predict their true orbital motions for all future times. [1]

1 How I solved the problem

1.1 Running the code

I wanted to run the code in Visual Studio, since it offers us a simple way of implementing libraries and sharing the project with others, any serious project nowdays is done in this framework. To run the code we need two libraries:

- opengl32: API for rendering 2D and 3D vector graphics.
- glfw: API for creating windows, contexts and surfaces, receiving input and events.

Since we are working on Visual Studio implementing this libraries is quite simple, we just have to put them on Additional Dependencies on our project propeties.

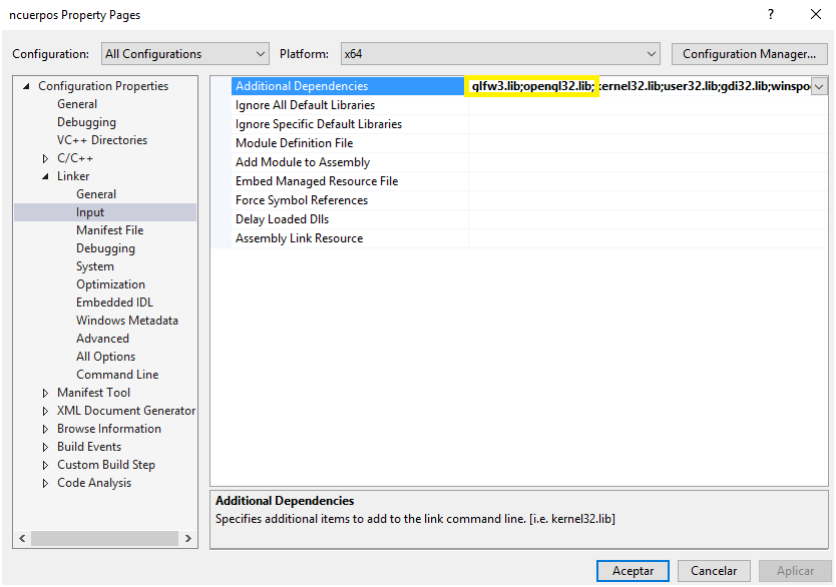

Figure 1: Libraries implementation in Visual Studio

One important note here, glfw will refresh $60 fps$ by defect when possible, if we want to see the simulation in real time, the frame rate must be stable, otherwise if we decide to save the image every time a frame passes we can later make a $60 fps$ video out of the images and the frame rate can be no stable.

1.2 Planets initial coditions

For planets initial conditions we want the following:

- Position:
  Random position between $[-1, 1] \times [-1, 1]$ using a uniform distribution.
- Velocity:
  With coordinates given by a gaussian distribution with $\mu = 0$, and $\sigma^2 = 0.1$
- Mass:
  Between 10^2 and 2×10^2

I used Mersenne Twister pseudorandom number generator, wich is the most widely used. In 2 we can see the simulation with this initial conditions.

Figure 2: Planets initial conditions, this is how the simulations just after start

1.3 Solving $\partial_t \vec{r} = \vec{v_0}$

For each planet we know $\vec{r}(t_0) = \vec{r_0}$, the analytical solution is:

$$\vec{r}(t) = \vec{r}(t_0) + \vec{v_0}(t - t_0)$$

Let's define:

$$t_n = t_0 + n\Delta t, \quad \vec{r_n} = \vec{r}(t_n)$$

We know:

$$\vec{r_0} = \vec{r}(t_0)$$

We can discretrize the problem in the following form:

$$\vec{r}_{n+1} = \vec{r_n} + \vec{v_0}\Delta t$$

In figure 3 we can see the orbit for a single planet wich follows this alghoritm.

1.4 Elastic collision with the boundary

We want to have elastic colition with the boundary, in this case a rectangle defined by:

$$R = [-x_b, x_b] \times [-y_b, y_b]$$

if $x \notin [-x_b, x_b]$ **then**
  set velocity to $(-v_x, v_y)$
end
if $y \notin [-y_b, y_b]$ **then**
  set velocity to $(v_x, -v_y)$
end

Algorithm 1: Elastic colition with boundary

here we are just reflecting the velocity vector respect to the vector paralel to the surface of colition. In figure 3 we can see the orbit for a single planet wich follows this alghoritm.

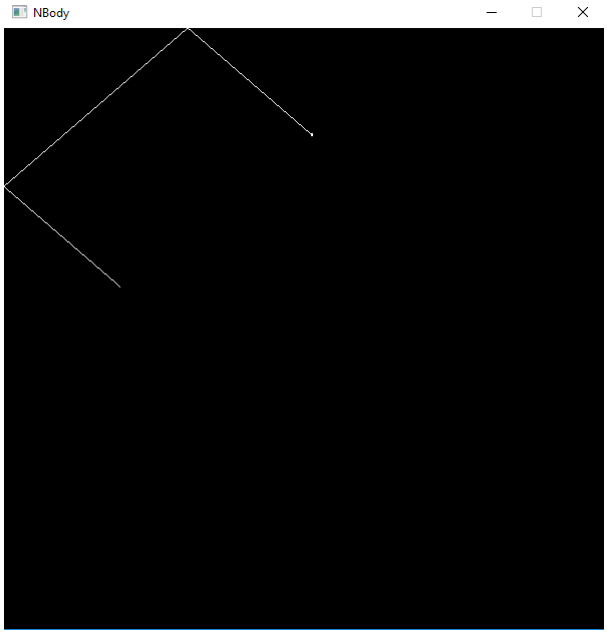

Figure 3: Orbit for a single planet, with initial velocity and elastic collision with the boundary

1.5 Computing forces between pairs

Here we need to compute force between pairs, let's say planet one has index i and planet two has index j, we want to find the force on planet i exerted by planet j, this is given by Newton's third law:

$$\vec{F}_{ij} = \frac{Gm_i m_j}{d_{ij}^3}\vec{d}_{ij}$$

where:

$$\vec{d}_{ij} = \vec{x}_j - \vec{x}_i$$

we have the relation:

$$\vec{F}_{ji} = -\vec{F}_{ij}$$

1.6 Computing the total force applied to each planet

The total force on each planet is just the sum of the forces:

$$\begin{aligned}\vec{F}_i &= \sum_{j=1, j\neq i} \vec{F}_{ij} \\ &= \sum_{j=1, j\neq i} \frac{Gm_i m_j}{d_{ij}^3}\vec{d}_{ij}\end{aligned}$$

since we want know to this for each planet the complexity of this operation is $O(n^2)$.

1.7 Solving $\mathbf{m}\partial_t^2\vec{x}(t) = \vec{F}$

This is the most important part of the problem, I used the leapfrog method wich is of second order and has low computational cost, this is the main problem here since this problem has complexity $O(n^2)$.
Suppose we want to solve the problem:

$$\frac{\partial^2}{\partial t^2}\vec{x}(t) = F(\vec{x}(t))$$

given a uniform partition of time:

$$t_n = t_0 + n\Delta t$$
$$\vec{x}_n = \vec{x}(t_n)$$

we have:

$$\vec{x}_{n+1} = \vec{x}_n + \vec{v}_n\Delta t + \frac{1}{2}\vec{a}_n\Delta t^2$$
$$\vec{v}_{n+1} = \vec{v}_n + \frac{1}{2}(\vec{a}_n + \vec{a}_{n+1})\Delta t$$
$$\vec{a}_{n+1} = F(\vec{x}_{n+1})$$

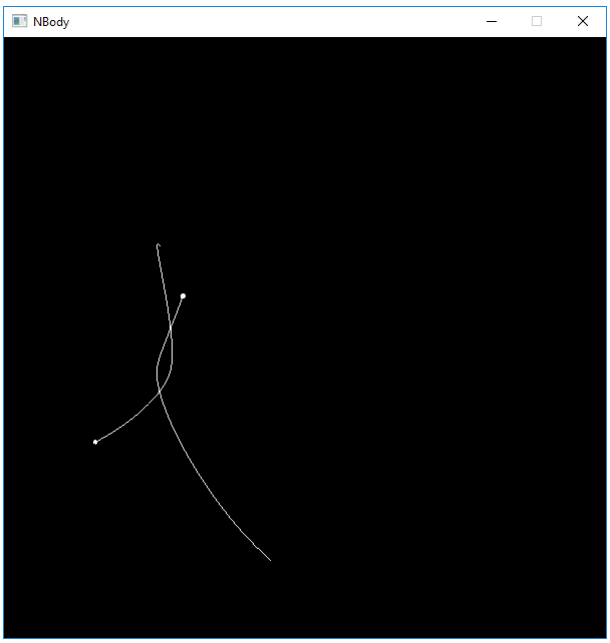

Figure 4: Two random planet's orbit given by leapfrog alghoritm

2 Running the simulation

One of the things I notice while running the simulation is the mass is not big enough for the gravitational force to be noticed, so I increased to be between $[10^5, 8 \times 10^5]$

Figure 5: Simulation just after start

Figure 6: Simulation after some time

3 Future work

- One way to see if our implementation of the alghoritm was done right, is by doing a coreography (setting the planets in such a way that the orbits are periodic), this can be done with 3 planets as there are known solution to this problem.
- The complexity of the algorithm can be reduced to $O(nlogn)$ using the Barnes–Hut algorithm.

References

[1] Wikipedia contributors. (2019, December 8). N-body problem. In Wikipedia, The Free Encyclopedia. Retrieved 09:44, December 13, 2019, from https://en.wikipedia.org/w/index.php?title=N-body_problemoldid=929761091