Universidad de Investigación de Tecnología Experimental Yachay Tech

School of Mathematics and Computer sciences

Program : Undergraduate	Semester : VIII
Subject : HPC	Professor : Saravana Prakash T
Date : 22/05/2019	Time:
Semester: Jan-May-2019	
Student's Surname, Name : BRYAN PATRICIO CHACHALO GOMEZ	
Student Reg.No:1004147961	

COLLISION SIMULATOR

Linear Momentum and Impulse

In modeling a collision between two objects, momentum is a way to characterize the state of an object in motion. The linear momentum, p, of an object is simply the mass of the object, m, multiplied by its velocity, v.

$$p = mv \tag{1}$$

When two object strikes a wall, it will change the direction of its flight and therefore its velocity components will change as well.

$$\vec{F} = \vec{p_1} - \vec{p_0} = m(\vec{v_1} - \vec{v_0}) \tag{2}$$

According to Equation (2), the collision of a moving object with something solid such as a wall change in velocity is the result of a linear impulse of force acting on the object due to the collision. The time of the collision, or dt, is generally very small, so according to Equation (6.6), in order for the impulse to be large enough to significantly change the momentum of the object, the force acting on the object must be very large. The force due to collision is known as an impulsive force. The magnitude of the impulsive force is usually so much larger than any other forces (gravity, drag, etc.) acting on the object during the collision, that all other forces can be ignored during the collision.

Two-Body Linear Collisions

Figure 1 shows a general two-body collision in the x-y plane. The objects have some initial velocities v_1 and v_2 and masses m_1 and m_2.

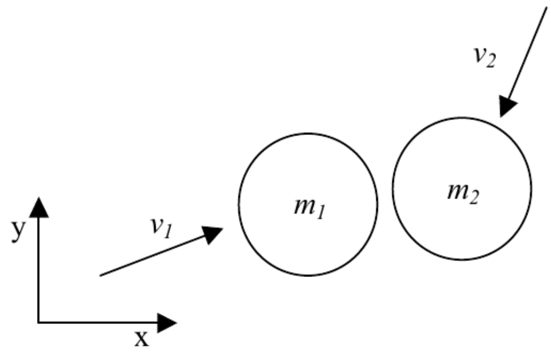

Figura 1: Schematic of a two-body linear collision

When they collide, the two objects will experience an impulse of force due to the collision. The magnitude of the impulse will be equal for both objects but will act in opposing directions. The geometric line along which the impulse acts is called the line of action for the collision. The line of action of the collision is a line drawn normal, or perpendicular, to the tangential plane at the point of collision. For the collision of the two spheres shown in Figure 1, the line of action is a line drawn through the center of the spheres that goes through the point of contact. To develop the equations that determine post-collision velocity, we will consider the collision of two spheres such that the line of action of the collision is parallel to the x-axis as shown in Figure 1. We will also assume that the impulsive force is significantly greater than all other forces acting on the colliding objects. For the duration of the collision, all other forces acting on the objects can be ignored. There is also assumed to be no friction between the two objects.

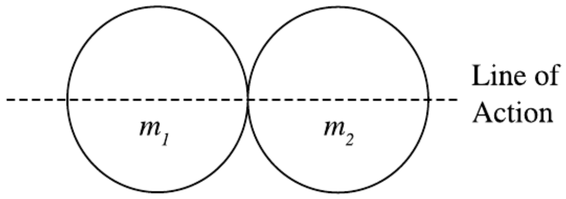

Figura 2: A collision causes an impulse of force to act along the line of action.

The linear impulse of force caused by the collision changes the velocity of the objects. The post-collision velocity components for the first object can be determined from Equation 2. Since the line of action for the collision is parallel to the x-axis, the linear impulse in the y-direction is equal to zero.
Below there are presented expressions can be obtained for the post-collision velocities along the line of action.

$$v_{2x}^{'} = \frac{(1+e)m_1}{m_1+m_2} + \frac{m_2 - em_1}{m_1+m_2}v_{2x} \tag{3}$$

$$v_{1x}^{'} = \frac{m_1 - em_2}{m_1+m_2}v_{1x} + \frac{(1+e)m_1}{m_1+m_2}v_{2x} \tag{4}$$

We can see from Equations 3,4 that the post-collision velocities along the line of action of the collision are a function of the pre-collision velocities along the line of action, the masses of the two objects, and the coefficient of restitution. The velocities in the y-direction, perpen- dicular to the line of action of the collision, are unaffected by the collision.

General two-dimensional collision

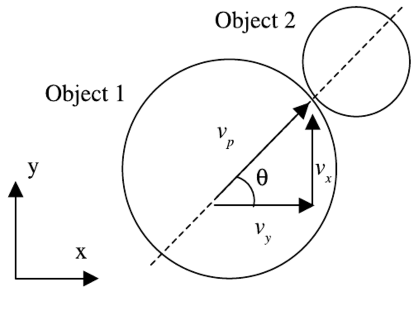

Figura 3: A general two-dimensional collision

Object 1 has pre-collision velocity components in the x- and y-directions equal to v_x and v_y. In order to analyze the collision, the velocity along the line of action, v_p, must be determined. Once v_p has been calculated, the post collision velocities along the line of action can be calculated according to Equations 3,4. The velocity along the line of action can be computed from the trigonometric relation shown in Equation 5.

$$v_p = v_x cos\theta + v_y sin\theta \tag{5}$$

Verlet Integration Method

The Verlet method is a second order sympletic integrator that computes position at the next time without the use of the velocity. The algorithm solves the position by utilizing two third-order Taylor series expansion for the position. To get a good approximations for the position δt needs to be sufficiently small.

$$x(t + \Delta t) = x(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \tag{6}$$

$$v(t + \Delta t) = v(t) + \frac{a(t) + a(t + \Delta t)}{2}\Delta t \tag{7}$$

Euler Integration Method

The Euler Method is the simplest explicit method for the numerical integration of ordinary differential equations. The method takes small finite time steps during which it approximate the slope of the function to be constant. The method on uses conditions at the current time to compute condition at later time.

$$v_{n+1} = v_n + g(t_n, x_n)\delta t \tag{8}$$

$$x_{n+1} = x_n + f(t_n, v_n)\delta t \tag{9}$$

Euler-Cromer Integration Method

The Euler-Cromer method, also known as the Sympletic Euler method is a modification of the Euler method described previously. This method make use of both the current state and the state at a later time to update the state. Euler-Cromer method is a first order sympletic integrator which does give more accurate results than the Euler method.

$$v_{n+1} = v_n + g(t_n, x_n)\delta t \tag{10}$$

$$x_{n+1} = x_n + f(t_n, v_{n+1})\delta t \tag{11}$$

This method differs from the standard Euler method in that it uses v_{n+1} in the x_{x+1} equation while the Euler method uses v_n.

COLLISION OF PARTICLES USING SERIAL, PARALLEL CODE USING MPI AND OPENMP

In this collision simulation there is not consider acceleration, leaving us equations 6,7 as follows:

$$x(t + \Delta t) = x(t) + v(t)\Delta t \tag{12}$$

$$v(t + \Delta t) = v(t) \tag{13}$$

The algorithm developed is as follows:

Algorithm 1 2D collision algorithm

```
1: For each particle in the system:
2: procedure 2D_COLLISION(a, b)    ▷ a,b particles with mass,(x,y) positions, ($v_x$,$v_y$) velocities
3:     Determine the line-of-action vector for the collision.
4:     Determine the velocity components along the line of action and normal to it.
5:     Compute the post-collision velocities from Equations (3,4).
6:     Rotate the post-collision velocities back to the original Cartesian coordinate system.
7: end procedure
8: procedure UPDATEPOSITIONS:
9:     For each particle in the system:
10:    Use integration method algorithm to calculate new positions and velocities of particles.
11: end procedure
```

The implementation was performed on a pc with linux environment, using openGL for graphic simulation. The graphical simulation was performed in a 4 cores 2.5 ghz with 8Gb of ram memory, and openGL 3.0 Mesa 19.0.3.

Figure 4 show how graphic simulation works.

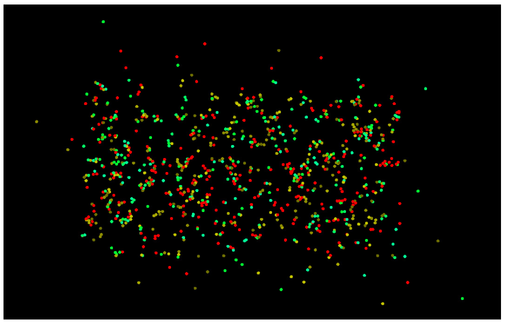

Figura 4: Simulation on openGL for 2000 particles

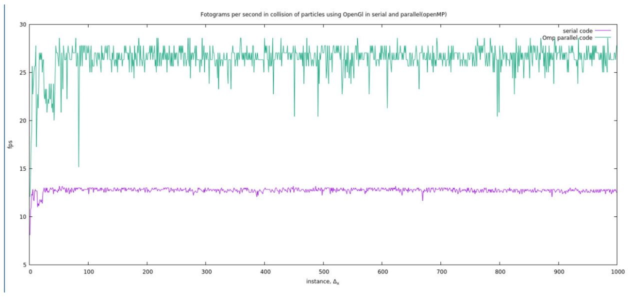

Figura 5: Fotograms per seconds in collision particles using serial code and openMP parallel code

Next, there are presented results of performance of serial, parallel code using openMP and parallel code using MPI directives for Verlet, Euler and Euler-Cromer methods of integration.

The code was compiled and executed in Quinde 1, using 2,4,8, and 12 cores.

No cores	seconds_mpi	seconds_omp	seconds_serial
2	9.62353	14.6191	33.8
4	3.70349	11.7414	33.8
8	2.42838	9.00153	33.8
12	2.15761	6.34023	33.8

Cuadro 1: Data of time performing for 400 cycles using Verlet algorithm

No cores	seconds_mpi	seconds_omp	seconds_serial
2	7.83	15.9122	27.91
4	2.36	9.14346	27.91
8	1.14	6.1153	27.91
12	0.79	4.37211	27.91

Cuadro 2: Data of time performing for 400 cycles using Euler algorithm

No cores	seconds_mpi	seconds_omp	seconds_serial
2	7.97	14.4296	28
4	2.48	8.30654	28
8	0.94	5.05995	28
12	0.6	3.78839	28

Cuadro 3: Data of time performing for 400 cycles using Euler-Cromer algorithm

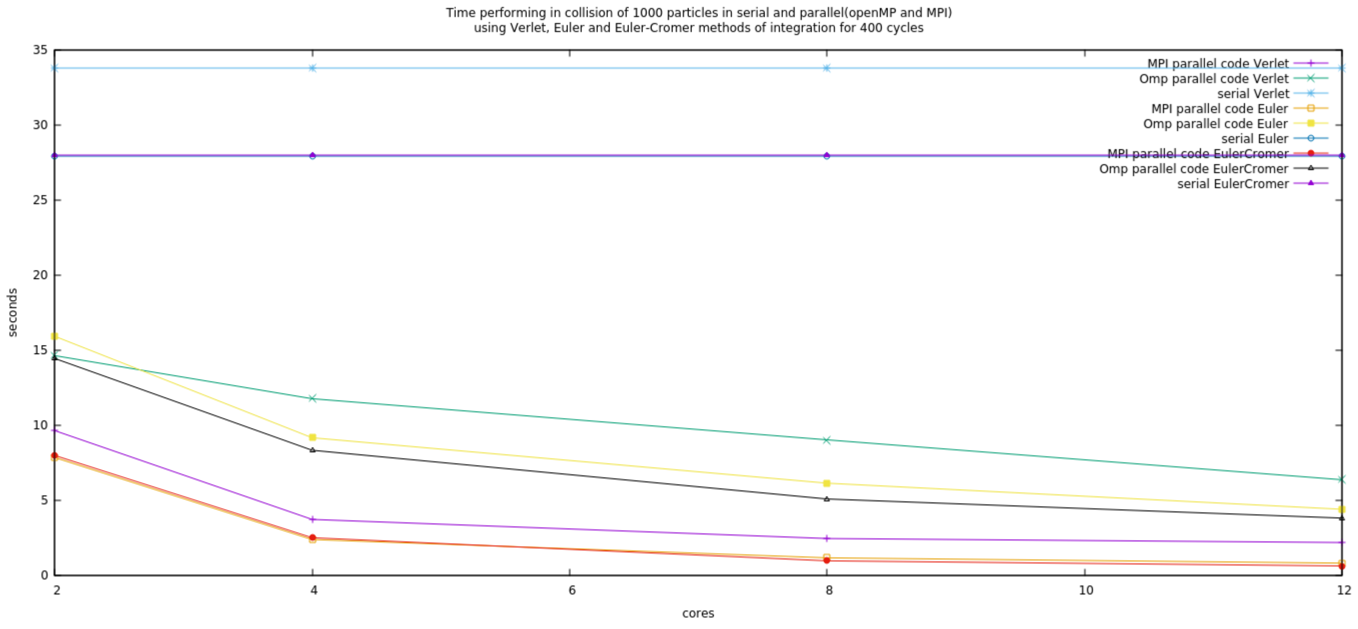

Figura 6: Time performing in collision of 1000 particles in serial and parallel(openMP and MPI) using Verlet, Euler and Euler-Cromer methods of integration for 400 cycles

Conclusion:

As we can see in the figures 5,6 by parallelizing the code we observe a substantial improvement in the performing of the code. Also, you can notice that MPI directives perform in a more efficient way, reducing the time necessary of calculations in this simulation. Another interesting point is that while more cores we use the time performing is monotonically reduced. Also, while more cores are used the performance of MPI directives tends to grow to almost double the performance of OpenMP. According to data obtained Euler-Cromer integration method is faster than Verlet and Euler integration method.