**REPORT ON THE VERLET INTEGRATOR**

Introduction

The aim of this report is to explain the Verlet Integrator we developed.

Our goal for this project is to have an interactable integrator that, given an initial data, in our concrete case the initial position x and y, speed vx and vy, acceleration ax and ay, a radius, a density and elapsed time, it computes their values at the end of that given time. Our intention is that it will also have a way to test the correct function of the integrator, which will print every frame and it will be able to pause it and a graphic representation. In addition, the Newton's Laws will be implemented in order to compare the results of the integrator with the final data and see the accuracy and the standard deviation.

Our frame rate will be of 60 fps. Air density will be implemented, as well as gravity. We will have a ground and the ball will be able to collide with it. Frame per frame we are going to be calculating each variable to update it.

To calculate the force in the x axis we will use the following formula:

fx = 0.5 \* AIR\_DENSITY \* new\_vx \* new\_vx \* area \* CD

Where:  
fx - force in the x axis  
AIR\_DENSITY - denstity of the air, constant  
new\_vx - the velocity for this frame  
area - area of the object (sphere)  
CD - drag coefficient

From the Verlet Integrator we know the following:

In case we didn't need the speed, the formulas we would use would be:

new\_vx = vx + new\_ax \* dt

Where:  
new\_vx - the velocity for this frame  
vx - the velocity of the previous frame   
new\_ax - the acceleration for this frame  
dt - the delta time (elapsed time) between the previous frame and this one

new\_x = new\_x \* 2 - x + new\_ax \* dt \* dt

Where:  
new\_x - the position for this frame  
x - the position of the previous frame  
dt - the delta time (elapsed time) between the previous frame and this one  
new\_ax - the acceleration for this frame

However, we are going to be using the velocity. Then, to compute the acceleration and the speed we will use MRUA, which is used to calculate the Velocity Verlet. In this case, the acceleraion would be take as constant and we would have:

new\_vx = vx + new\_ax \* dt

Where:  
new\_vx - the velocity for this frame  
vx - the velocity of the previous frame   
new\_ax - the acceleration for this frame  
dt - the delta time (elapsed time) between the previous frame and this one

new\_x = new\_x \* 2 - x + new\_ax \* dt \* dt

Where:  
new\_x - the position for this frame  
x - the position of the previous frame  
dt - the delta time (elapsed time) between the previous frame and this one  
new\_ax - the acceleration for this frame

Then again, we take into account that the acceleration may not be the same through all of this process and, therefore, the formulas we are finally coing to use are:

new\_ax = fx / mass

Where:  
new\_ax - the acceleration for this frame  
fx - force in the x axis  
mass - mass of the object, given a radius and a density

new\_vx = vx + new\_ax \* dt

Where:  
new\_vx - the velocity for this frame  
vx - the velocity of the previous frame   
new\_ax - the acceleration for this frame  
dt - the delta time (elapsed time) between the previous frame and this one

new\_x = x + vx \* dt + (new\_ax / 2.0) \* dt \* dt

Where:  
new\_x - the position for this frame  
x - the position of the previous frame  
vx - the velocity of the previous frame  
dt - the delta time (elapsed time) between the previous frame and this one  
new\_ax - the acceleration for this frame

We will use Newton's Laws to compare the results of the integrator with the "reality", as stated before. To compute the acceleration, the velocity and the position we will use the MRUA formulas:

new\_ax = fx / mass

Where:  
new\_ax - the acceleration for this frame  
fx - force in the x axis  
mass - mass of the object, given a radius and a density

new\_vx = vx + new\_ax \* dt;

Where:  
new\_vx - the velocity for this frame  
vx - the velocity of the previous frame   
new\_ax - the acceleration for this frame  
dt - the delta time (elapsed time) between the previous frame and this one

new\_x = x + vx \* dt + (new\_ax / 2.0) \* dt \* dt

Where:  
new\_x - the position for this frame  
x - the position of the previous frame  
vx - the velocity of the previous frame  
dt - the delta time (elapsed time) between the previous frame and this one  
new\_ax - the acceleration for this frame

To compute the forces, the acceleration, the speed and the position for the y axis, we will use the formulas stated above as well, but taking into account the gravity, as said before.

To see further information on where we took our information on what formulas to use from, please take a look at our "Data" folder in our GitHub repository (<https://github.com/Needlesslord/Physics2theory>) the following web pages:

<https://www.algorithm-archive.org/contents/verlet_integration/verlet_integration.html>

<https://en.wikipedia.org/wiki/Verlet_integration>

<https://www.gamedev.net/articles/programming/math-and-physics/a-verlet-based-approach-for-2d-game-physics-r2714>

To summarize, the final result of the integrator should have a welcome and small tutorial/explanation of how it works, the input of the data, then select whether the user wants to test the integrator or they want only the final results. If they choose the first option, every frame will be printed on the console, which can be paused, showing the data of the last frame, a pause signal and the results in that same position calculated with Newton's Laws and that can be unpaused again to continue the test. On the other hand, if the user chooses to only show the final data, the initial data, the final data calculated with the integrator, the final data calculated with Newton's Laws and a graphic representation will be shown. In both cases will be possible to go to the main selection again once finished.

Implementation

QUÉ HABÉIS HECHO, POR QUÉ LO HABÉIS HECHO Y CÓMO LO HABÉIS HECHO

The verlet integrator

The Integrator is a program that simulates how an object would perform inside a game with physics. For this simulation we have decided to use a cube. The reason is because it is because it is a shape used for the most part of the collisions because it is a shape that does not use a lot of resources. Also it is easier to apply friction to it than to a sphere.

When the program starts, it asks the user to enter some data to know from what state it has to start calculating the cube data over time. Input is done with the c++ standard input provided with iostream.

It takes into account the forces that the object receives to recalculate the acceleration of the object, its velocity, and its position. It is also prepared to calculate the aerodynamics forces depending on the medium the object is.

The integrator works calculating the new object data taking into account the previous frame data. The frame rate chosen for our integrator is 60 fps, which is the common frame rate in a large amount of videogames. To change the frame rate, the only thing needed to do is to change the global value of the fps variable. This will automatically change the time step.

It is also able to calculate when an object is collisioning with another calculating their dimensions and the distances between them. But there are no recalculation of the objects data after collisioning. That can be used in a game to make games collide in-game, or to use objects as detectors.

The distance between two cubes is calculated taking as reference the center of each object, and the collision is detected when the distance in X or Y is lower than half of the length of one plus half of the length of the other;

When the object information is displayed on screen, this is shown per frame. To make it easier for the user to read the time information, apart from showing the frames passed from the beginning, also is shown the second and the frames passed on that second (each second has as much subdivisions as fps).

Test of the integrator

Final data show

Forces

Frame rate

Input

Pause

Collision

Other things you have implemented, why you did it, how you did it, how it works

Newton's Laws

"Interface"

Graphic representation

All the code comes together and how it works

Results

Tests we want to run (conditions, initial data)

Results of the tests (with images!)

Standard deviation

Comparison with other integrators and Newton's Laws

Compare results with Newton's Laws

Compare results with other integrators (ask classmates)

Conclusions

Conclusions

NÚRIA

TOMÁS

ALBERT

RAUL ALEX ENRIC

MARC