**INTEGRADOR**

**DE**

**VERLET**

**MISIÓN #1**

Autores:

Marc Pagès

Arnau Falgueras

Marc Ramos

Roger Promera

Pol Camacho

Carlos Redolar

Pablo Galve

Josep Sànchez

Silvino Medina

ÍNDICE

[1. Introducción 3](#_Toc24654414)

[1.1 Integrador 3](#_Toc24654415)

[1.2 Algoritmo de Verlet 3](#_Toc24654416)

[1.3 Historia 3](#_Toc24654417)

[1.4 Objetivos 4](#_Toc24654418)

[2. Desarrollo 5](#_Toc24654419)

[2.1 Verlet integration 5](#_Toc24654420)

[2.2 Classical Motion 5](#_Toc24654421)

[2.3 Velocity Verlet 5](#_Toc24654422)

[2.4 Stormer Verlet 6](#_Toc24654423)

[2.5 Acceleration Sum & Drag Acceleration 6](#_Toc24654423)

[2.6 Calculate Time](#_Toc24654423) 7

[2.6 Check Collision](#_Toc24654423) 7

[3. Código](#_Toc24654424) 8

[4. Conclusiones](#_Toc24654425) 9

# 1. Introducción

Antes que nada, para ponernos en contexto es necesario hacer una introducción para saber qué es el algoritmo de Verlet y qué es un integrador, para así poder entender qué significa cada concepto y el funcionamiento en su totalidad.

1.1 Integrador

Un integrador es un dispositivo que en su salida realiza la operación matemática de integración, es decir, una generalización de la suma de infinitos sumandos, infinitesimalmente pequeños.

1.2 Algoritmo de Verlet

El algoritmo de Verlet es un procedimiento para la integración numérica de ecuaciones diferenciales ordinarias de segundo orden con valores iniciales conocidos. Es especialmente utilizado en las situaciones en que la expresión de la segunda derivada solo es función de las variables, ya sea dependiente o independiente, sin participar la primera derivada.

A parte del algoritmo base de Verlet, existe otro basado en la velocidad, el cual es considerado mejor que el algoritmo original. Es bastante parecido al algoritmo Leapfrog, el cual esta basado en actualizar posiciones y velocidades en puntos del tiempo intercalados, escalonados de tal manera que se saltan los unos a los otros. Pero, a diferencia de este otro algoritmo, la velocidad y la posición son calculados con el mismo valor que la variable de tiempo.

## 1.3 Historia

La primera persona en presentar la primera versión de la denominada Integración de Verlet fue el matemático francés Loup Verlet, que lo hizo en el año 1967 y se caracterizaba a la vez por su simplicidad, exactitud y estabilidad. La siguiente versión, propuesta en 1985, fue apodada como “Algoritmo de Verlet con velocidad”. Esta consistía en lo mismo que la anterior, con ligeras correcciones en la integración y con mejoras en la precisión y la estabilidad de las soluciones.

## 1.4 Objetivos

El objetivo principal del proyecto es implementar un integrador basado en el algoritmo de Verlet, el cual tenga un óptimo funcionamiento y que no solo esté basado en cálculos matemáticos y valores, sino que también disponga de una parte gráfica que proyecte todos los cálculos de una forma más visual y fácil de entender.

A parte, no queremos introducir solo las ecuaciones básicas del algoritmo de Verlet, sino ir más allá e implementar las máximas ecuaciones posibles para que el integrador sea lo más completo posible.

Además, queremos que el usuario, a través de la consola, pueda determinar cuales son los valores de las diferentes unidades de las fórmulas, para que el resultado que se refleje de forma gráfica sea el que la propia persona haya introducido en el menú inicial.

# 2. Desarrollo

## 2.1 Verlet integration



Esta es la ecuación base de la integración de Verlet, la cual hemos utilizado en primer lugar y tiene como unidades la posición, la aceleración y el tiempo.

La ecuación lo que hace es el calculo de la posición del objeto partiendo de la posición 0 y si ya se ha introducido una posición anteriormente parte de esa posición. Así pues, para calcularla, en primer lugar, sumamos la posición actual y la suma a la posición menos la posición previa. Acto seguido se multiplica este valor por la aceleración, y finalmente se multiplica por el tiempo, el cual es 1, al cuadrado.

Tanto los valores de la aceleración, como de la posición son dados por el usuario, y para ello disponemos de un menú en el cual se pregunta por dichos valores. Aparte de estos dos valores, también se pregunta por otros valores (velocidad y masa), los cuales no son utilizados en esta ecuación en concreto, pero sin en ecuaciones siguientes.

## 2.2 Classical Motion

En el caso de la función de Classical Motion, lo que hacemos es calcular tanto la nueva posición de la partícula, como su nueva velocidad, como su nueva aceleración.

En primer lugar, se calcula la aceleración mediante la fórmula utilizada anteriormente en Verlet Integration, donde se suma la posición de la partícula a la posición restado de la posición previa sumado de la multiplicación de la aceleración y el tiempo al cuadrado.

Seguidamente, para calcular la nueva velocidad, se suma la velocidad inicial de la partícula a la multiplicación de la aceleración multiplicado del tiempo.

Entonces, partiendo de la nueva velocidad obtenida, se consigue la nueva aceleración restando esta velocidad de la velocidad inicial y todo ello dividido del tiempo.

## 2.3 Velocity Verlet



En este caso nos encontramos delante de el algoritmo de Verlet en base de la velocidad el cual tiene unas cuantas variaciones en relación con el original.

Esta ecuación calcula la velocidad a la que se desplaza un cuerpo basándose en la velocidad inicial, la aceleración inicial, otra aceleración nueva y el tiempo, el cual es 1.

Lo que hace la ecuación es el calculo de la nueva velocidad partiendo de la velocidad inicial sumado a la suma de las aceleraciones dividido entre dos y todos multiplicado por el tiempo, 1. En este caso, en el menú inicial se le pregunta al usuario cual es la velocidad que quiere utilizar, si solo quiere utilizar la gravedad como aceleración o también tener en cuanta la densidad, el área y el coeficiente aerodinámico de arrastre.

## 2.4 Stormer Verlet

2.5 Acceleration sum & Drag acceleration

No se puede entender la función de Acceleration Sum sin antes calcular la de Drag Acceleration. Aquí asumimos que la aceleración de la partícula no es constante y varía según una serie de factores que veremos a continuación.



En este caso, con la Drag Acceleration calculamos la desaceleración de una partícula cuando existe una fuerza de rozamiento (pongamos por caso, con el aire) contraria al movimiento de la partícula. Esta aceleración, que puede ser en el eje x o y, depende de la densidad, del coeficiente de rozamiento, el área de la partícula, la velocidad (en x o y) y la masa de la partícula.

Por lo tanto, el resultado es la multiplicación de la densidad por el coeficiente de rozamiento por el área por ½, todo ello multiplicado por la velocidad al cuadrado (dependiendo de si queremos coger la aceleración en x o y tendremos que utilizar la velocidad en x o y) y todo lo anterior dividido entre la masa.

En el menú inicial le preguntamos al usuario si quiere tener una única aceleración, la gravedad. Si la respuesta es NO, entonces le pedimos que dé valores de gravedad, de coeficiente aerodinámico, de densidad y de masa. La velocidad ya está definida por el usuario anteriormente.

Cabe decir que esta desaceleración es en dirección opuesta al movimiento de la partícula, y por lo tanto se deberá restar a la aceleración final como veremos a continuación.

Con esto ya podemos calcular la Acceleration Sum, que no es más que el cálculo de la aceleración total en un momento determinado. Para ello, mediante un iPoint, creamos la suma de aceleraciones en x y en y, y las inicializamos en 0. Después, igualamos la suma a la aceleración de ese momento de la partícula, y para la suma en y, se le añade la gravedad. Finalmente la suma total de la aceleración será ella misma menos la Drag Acceleration.

2.6 Calculate Time

Esta función se utiliza siempre que la partícula colisiona. Con ella calculamos el tiempo total de colisión desde que esta se produce hasta que la partícula deja de colisionar. Para ello se usa la posición previa del cuerpo cuando colisiona, la posición nueva cuando deja de colisionar, la velocidad en ese intervalo de tiempo y la aceleración.

2.7 Check Collision

Esta función solamente comprueba si existe una colisión entre la partícula y un objeto dentro del mundo desde todas las direcciones posibles (por arriba, por abajo, por la derecha y por la izquierda.

Empezamos creando un booleano llamado ret, y le asignamos el valor false. Este se volverá true cuando exista la colisión, y seguirá falso cuando no la haya.

Para que exista la colisión, se deben cumplir 4 condiciones que están incluidas dentro del if: La primera es si la posición x de la partícula más su radio es igual o superior a la posición x del objeto. La segunda, si la posición x de la partícula menos su radio es igual o menor a la posición x del objeto más su anchura. La tercera, si la posición y de la partícula menos su radio es igual o menor a la posición y del objeto más su altura. Y la cuarta, si la posición y de la partícula más su radio es igual o mayor a la posición y del objeto.

Si cualquiera de estos 4 casos se cumple, entonces el booleano ret se vuelve true y la función retorna ese valor.

A continuación existe otra función de Check Collision, pero con la diferencia de que retorna todas las posibles colisiones que puedan existir entre la partícula y el objeto.

2.8 Calculate Collision Position

Una vez sabemos que se ha producido una colisión, entonces utilizamos esta función para saber qué posición ocupa la partícula justo antes de que esta colisione. Para eso, necesitaremos una variable float llamada time, la función Calculate Time (que se explicará más adelante) y dos booleanos (col\_x y col\_y) que representan las colisiones en x y en y.

Empezamos igualando la variable time a 0 y los booleanos a false, ya que solo serán true si se cumplen 4 condiciones de colisión muy parecidas a las de la función Check Collision.

Si la posición x previa de la partícula (no la actual) más su radio es más pequeña que la posición x del otro objeto, entonces igualamos la variable time a la función Calculate Time, donde le pasamos la posición previa de la partícula, su velocidad, su aceleración y la posición x del objeto. El booleano col\_x se iguala a true.

Si la posición x previa de la partícula menos su radio es superior a la posición x del objeto más su anchura, igualamos la variable time a la función Calculate Time y le pasamos las mismas variables que antes pero añadiendo la anchura del objeto. El booleano col\_x se iguala a true.

Si la posición y previa de la partícula (no la actual) más su radio es más pequeña que la posición x del otro objeto, entonces igualamos la variable time a la función Calculate Time, donde le pasamos la posición previa de la partícula, su velocidad, su aceleración y la posición x del objeto. El booleano col\_x se iguala a true.

Si la posición y previa de la partícula (no la actual) más su radio es más pequeña que la posición x del otro objeto, entonces igualamos la variable time a la función Calculate Time, donde le pasamos la posición previa de la partícula, su velocidad, su aceleración y la posición x del objeto. El booleano col\_x se iguala a true.

# 3. Código

# 4. Conclusiones