Proyecto de programación orientada al rendimiento

J. Daniel Garcia (coordinador)
Arquitectura de Computadores
Departamento de Informática
Universidad Carlos III de Madrid

2023

1. Objetivo

Este proyecto tiene como objetivo fundamental hacer el que los estudiantes se familiaricen con la **optimización de programas secuenciales**.

En concreto, la práctica se centrará en el desarrollo de software secuencial en el lenguaje de programación C++ (incluyendo las mejoras de C++20).

2. Introducción

La simulación de flujos de fluidos se puede lograr con diversos métodos. Uno de ellos es la simulación de hidrodinámica de partículas suavizada (*Smoothed Particle Hydrodynamics* o *SPH*). La simulación SPH resuelve las ecuaciones de Navier-Stokes en pasos de tiempo. Un fluido se modela como una colección de partículas. Estas partículas tienen interacciones que tienen impacto en su aceleración, posición y velocidad.

Solamente las partículas que están a una corta distancia tienen una interacción real. Para acelerar la simulación, las interacciones de larga distancia se descartan. Para lograr esto, el espacio 3D se divide en una malla de bloques y solamente se tienen en cuenta las partículas en bloques vecinos.

La simulación incluye varios pasos:

- Lectura del estado inicial de la simulación.
- Simulación de las partículas para cada paso de tiempo o iteración:
  - Reposicionamiento de cada partícula en la malla.
  - Cálculo de las fuerzas y aceleraciones de cada partícula.
  - Procesamiento de las colisiones de partículas con los límites del recinto.
  - Movimiento de partículas.
  - Reprocesamiento de los límites del recinto.
- Almacenamiento del estado final de la simulación.

3. Visión General

3.1. Introducción

El objetivo de este proyecto es el desarrollo de una aplicación que realice una simulación de un fluido durante un número de pasos de tiempo,

La aplicación toma los siguientes parámetros de la línea de comandos:

- **Iteraciones**: Número de iteraciones o pasos de tiempo a calcular.
- **Entrada**: Archivo de datos con la descripción del estado inicial del fluido.
- **Salida**: Archivo de salida con la descripción del estado final del fluido.

IMPORTANTE: Aunque los archivos de entrada y salida contienen información representada en coma flotante de simple precisión, todos los cálculos se realizarán utilizando coma flotante en doble precisión.

3.2. Formato de archivo

Un archivo de fluidos contiene toda la información del estado del fluido en un punto determinado del tiempo. Los archivos se almacenan en formato binario con los siguientes requisitos:

- **Entero**: Todos los valores enteros utilizan 4 bytes.
- **Coma flotante**: Todos los valores en coma flotante utilizan representación IEEE-754 de simple precisión.

Un archivo está formado por una **cabecera** y un **cuerpo**. La cabecera incluye información general mientras que el cuerpo contiene información de cada partícula del fluido.
La cabecera contiene los campos especificados en la tabla 1.

Campo	Tipo	Descripción
ppm	**Coma flotante**	Partículas por metro
np	**Entero**	Número de partículas

Tabla 1: Campos en la cabecera del archivo.

Ten en cuenta que el **número de partículas** especifica cuantas partículas están almacenadas en el resto del archivo. La información y el formato para cada partícula en el **cuerpo** del archivo se especifica en la tabla 2.

Campo	Tipo	Descripción
px	**Coma flotante**	Coordenada x de la posición.
py	**Coma flotante**	Coordenada y de la posición.
pz	**Coma flotante**	Coordenada z de la posición.
hvx	**Coma flotante**	Coordenada x del vector hv.
hvy	**Coma flotante**	Coordenada y del vector hv.
hvz	**Coma flotante**	Coordenada z del vector hv.
vx	**Coma flotante**	Coordenada x de la velocidad.
vy	**Coma flotante**	Coordenada y de la velocidad.
vz	**Coma flotante**	Coordenada z de la velocidad.

Tabla 2: Campos para cada partícula en el cuerpo del archivo.

3.3. Constantes de la simulación

La tabla 3 muestra las constantes utilizadas en la simulación:

Constante	Descripción	Valor
r	**Multiplicador de radio**	1.695
ρ	**Densidad de fluido**	10^3
p_s	**Presión de rigidez**	3.0
s_c	**Colisiones de rigidez**	$3 \cdot 10^4$
d_v	**Amortiguamiento**	128.0
μ	**Viscosidad**	0.4
d_p	**Tamaño de partícula**	$2 \cdot 10^{-4}$
Δt	**Paso de tiempo**	10^{-3}

Tabla 3: Constantes escalares de la simulación.

Además, la tabla 4 presenta la constantes vectoriales a utilizar en la simulación. Ten en cuenta que la **aceleración externa** ($\vec{g}$) es la aceleración de la gravedad, que en este ejemplo actúa a lo largo del eje y. El **límite superior del recinto** ($\vec{b}_{max}$) y **límite inferior del recinto** ($\vec{b}_{min}$) define dos vértices opuestos en un cuboide rectangular. Este es el dominio cerrado en el que ocurre la simulación. $\vec{b}_{max}$ viene dado por el vector $(x_{max}, y_{max}, z_{max})$ y $\vec{b}_{min}$ viene dado por el vector $(x_{min}, y_{min}, z_{min})$.

Constante	Descripción	x	y	z
$\vec{g}$	**Aceleración externa**	0.0	-9.8	0.0
$\vec{b}_{max}$	**Límite superior de recinto**	x_{max} = 0.065	y_{max} = 0.1	z_{max} = 0.065
$\vec{b}_{min}$	**Límite inferior de recinto**	x_{min} = -0.065	y_{min} = -0.08	z_{min} = -0.065

Tabla 4: Constantes vectoriales de simulación.

La figura 1 muestra una posible malla dividida en bloques y los límites del recinto b_{min} y b_{max}.

Ten en cuenta que pueden ser necesarias algunas constantes matemáticas (p.ej. π). En tal caso el valor preferido será el suministrado por la biblioteca estándar de C++20. Por favor, consulta el espacio de nombres **std::numbers** para dichas constantes.

3.4. Parámetros de la simulación

Varios parámetros dependen de la cantidad de partículas por metro (**ppm**).

La masa de una partícula m depende de la densidad ρ y de la cantidad de partículas por metro (**ppm**).

$$m = \frac{\rho}{ppm^3}$$

Otro parámetro relevante es la **longitud de suavizado** (h), que depende del **multiplicador de radio** (r) y de la cantidad de partículas por metro (**ppm**).

$$h = \frac{r}{ppm}$$

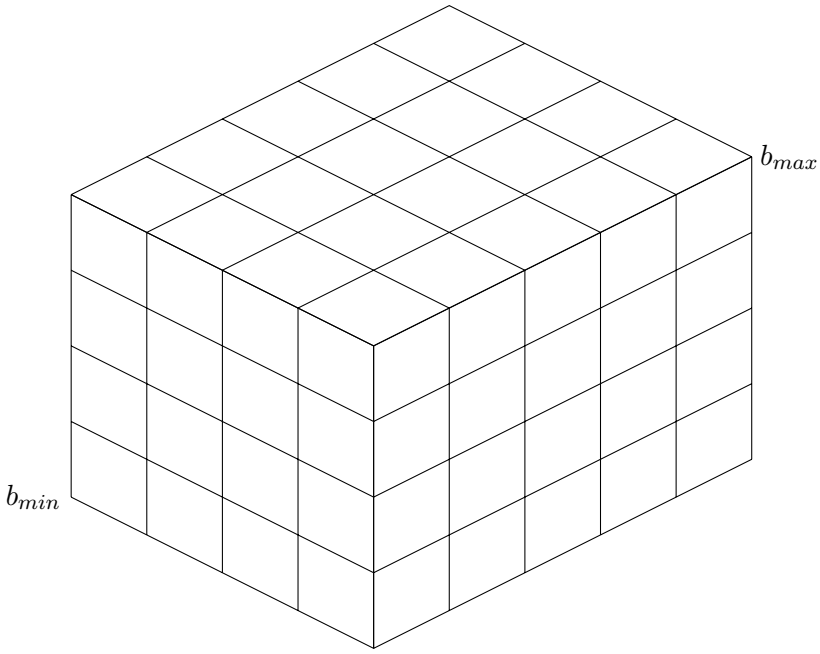

Figura 1: Cube dimension.

3.5. La malla de simulación

Para evita realizar demasiados cálculos, el recinto de simulación completo se divide en bloques de menor tamaño. Ten en cuenta que el recinto de simulación es un cuboide rectangular con longitudes diferentes en cada una de las tres dimensiones.

Para determinar el número de bloques de cada dimensión se divide la longitud en esa dimensión por el parámetro h.

$$n_x = \lfloor \frac{x_{max} - x_{min}}{h} \rfloor$$
$$n_y = \lfloor \frac{y_{max} - y_{min}}{h} \rfloor$$
$$n_z = \lfloor \frac{z_{max} - z_{min}}{h} \rfloor$$

El tamaño de un bloque de la malla $\vec{s}\,(s_x, s_y, s_z)$ viene dado por

$$s_x = \frac{x_{max} - x_{min}}{n_x}$$
$$s_y = \frac{y_{max} - y_{min}}{n_y}$$
$$s_z = \frac{z_{max} - z_{min}}{n_z}$$

Dada una partícula con vector de posición $\vec{p}$, los índices de bloque se calcula teniendo en cuenta su distancia en cada dimensión a la coordenada mínima del recinto y el tamaño de un bloque de la malla ($\vec{s}$).

$$i = \lfloor \frac{p_x - x_{min}}{s_x} \rfloor$$
$$j = \lfloor \frac{p_y - y_{min}}{s_y} \rfloor$$
$$k = \lfloor \frac{p_z - z_{min}}{s_z} \rfloor$$

Ten en cuenta que para cada dimensión el correspondiente índice no puede ser menor que 0 o mayor o igual que el correspondiente número de bloques en esa dimensión.

$$i \in [0, n_x - 1]$$
$$j \in [0, n_x - 1]$$
$$k \in [0, n_x - 1]$$

Si un índice está fuera de su rango correspondiente, se debe ajustar al correspondiente límite.

El papel clave de los bloques de simulación es que solamente usarán para los cálculos las partículas en el mismo bloque o en bloques contiguos. En general, para cualquier bloque hay 26 bloques contiguos. Este número puede ser menor cuando el bloque está en los límites de la malla. Por ejemplo, cuando $i = j = k = 0$, el número de 7.

4. Etapas de la simulación

La estructura de la aplicación de simulación es la siguiente:

1. **Análisis de argumentos**: Se analizan los argumentos suministrados a través de la línea de comandos.
2. **Iniciación de la simulación**: Se carga el estado inicial de la simulación desde un archivo y se da valores iniciales a las estructuras de datos.
3. **Procesamiento de pasos de tiempo**: Para el número especificado de pasos de tiempo se calcula el nuevo estado para cada partícula.
4. **Almacenamiento de resultados**: El estado final de la simulación se guarda en un archivo.

4.1. Análisis de argumentos del programa

La aplicación tomará exactamente tres parámetros:

- **nts**: Número de *time steps* (pasos de tiempo). Este parámetro especifica durante cuántos paso de tiempo debe ejecutarse la simulación.
- **inputfile**: Archivo de entrada con el estado inicial de la simulación.
- **outputfile**: Archivo de salida con el estado final de la simulación.

Por ejemplo, el mandato:

```
$ fluid 2000 init.fld final.fld
```

Carga el archivo **init.fld** ejecuta **2000** pasos de tiempo y genera un archivo de salida llamado **final.fld**.

Si el número de argumentos no es exactamente tres argumentos, se generará un mensaje de error y el programa terminará con el código de error **-1**.

```
$ fluid
Error: Invalid number of arguments: 0.
$ fluid 4
Error: Invalid number of arguments: 1.
$ fluid 2000 init.fld
Error: Invalid number of arguments: 2.
$ fluid 2000 init.fld final.fld 45
Error: Invalid number of arguments: 4.
```

Si el primer argumento no es un número entero, se generará un mensaje de error y el programa terminará con el código de error **-1**.

```
$ fluid hello init.fld final.fld
Error: time steps must be numeric.
```

Si el número de pasos de tiempo es un número negativo, se generará un mensaje de error y el programa terminará con el código de error **-2**.

```
$ fluid -3 init.fld final.fld
Error: Invalid number of time steps.
```

Si el archivo de entrada no se puede abrir para lectura, se generará un mensaje de error y el programa terminará con el código de error **-3**.

```
$ fluid 1 init.fld final.fld
Error: Cannot open init.fld for reading
```

Si el archivo de salida no se puede abrir para escritura, se generará un mensaje de error y el programa terminará con el código de error **-4**.

```
$ fluid 5 init.fld final.fld
Error: Cannot open final.fld for writing
```

4.2. Iniciación de la simulación

Durante la iniciación, se abre el archivo de entrada y se leen los parámetros de simulación principales (partículas por metro y número de partículas). Después, se calculan otros parámetros que dependen de éstos.

Estos parámetros incluyen:

- La **longitud de suavizado** (h).
- La **masa de la partícula** (m).
- El **vector de tamaño de malla** ($\vec{n} = (n_x, n_y, n_z)$).
- El **tamaño de bloque de malla** ($\vec{s} = (s_x, s_y, s_z)$).

La información para cada partícula se lee del archivo de entrada. Aunque los archivos de entrada utiliza simple precisión para los valores en coma flotante, tan pronto como los valores se leen se convertirán a doble precisión. Las partículas reciben un identificador numérico entero consecutivo a partir del **0**, manteniendo el mismo orden que tienen en el fichero de entrada.

Una vez se ha leído una partícula, se determina su correspondiente bloque en la malla y la partícula se añade al conjunto de partículas de ese bloque.

Si el número de partículas encontrado en la cabecera es **0** o un número negativo, el programa termina con el código de error **-5** y se genera un mensaje de error.

```
$ fluid 2000 init.fld final.fld
Error: Invalid number of particles: 0.
```

Si el número de partículas encontrado en el archivo no coincide con el parámetro **número de partículas** (**np**), se genera un mensaje de error y el programa termina con el código de error **-5**. Por ejemplo, si la cabecera tiene el valor **4750** para **np** pero el archivo contiene **4700** partículas, el siguiente mensaje se envía a la salida estándar de errores.

```
$ fluid 2000 init.fld final.fld
Error: Number of particles mismatch. Header: 4750, Found: 4700.
```

Como otro ejemplo, si la cabecera tiene el valor **4750** para **np** pero el archivo tiene **5000** partículas, el siguiente mensaje se envía a la salida estándar de errores.

```
$ fluid 2000 init.fld final.fld
Error: Number of particles mismatch. Header: 4750, Found: 5000.
```

Si el archivo se lee con éxito, los valores de los parámetros se envían a la salida estándar.

```
$ fluid 2000 init.fld final.fld
Number of particles: 4800
Particles per meter: 204
Smoothing length: 0.00830882
Particle mass: 0.00011779
Grid size: 15 x 21 x 15
Number of blocks: 4725
Block size: 0.00866667 x 0.00857143 x 0.00866667
```

4.3. Procesamiento de la simulación

La etapa de procesamiento de la simulación realiza cálculos sobre todas las partículas en cada paso de tiempo. Por tanto, esta etapa se repite tantas veces como el número de pasos de tiempo que se especifique.

Para cada paso de tiempo se ejecutan las siguientes sub-etapas:

1. Reposicionamiento de cada partícula en la malla.
2. Cálculo de fuerzas y aceleraciones para cada partícula.
3. Procesamiento de colisiones.
4. Movimiento de partículas.
5. Procesamiento de límites.

4.3.1. Reposicionamiento de partículas

Durante esta etapa, se comprueban las coordenadas de cada partícula para asegurar que se encuentran en el bloque correcto de la malla de simulación. Si una partícula no está en el bloque correcto, se traslada al bloque que le corresponda.

Ten en cuenta que debido a que la posición de la partícula puede haber cambiado, puede ser necesario actualizar el bloque en el que se encuentra.

4.3.2. Cálculo de aceleraciones

Se realizan los siguientes pasos:

1. Iniciación de aceleraciones.
2. Incremento de densidades.
3. Transformación de densidades.
4. Transferencia de aceleraciones.

Iniciación de densidades y aceleraciones Durante este paso se da un valor inicial a la densidad (ρ_i) y el vector aceleración ($\vec{a_i}$) para cada partícula.

La densidad se inicia a **0**.

$$\rho_i = 0$$

El vector aceleración se inicia al valor de la **aceleración externa**.

$$\vec{a}_i = \vec{g}$$

Incremento de densidades En este paso, para cada bloque de la malla de simulación, se consideran todos los bloques contiguos así como el bloque actual. Para cada par de partículas i y j, se incrementan las densidades.

$$\Delta\rho_{ij} = \begin{cases} (h^2 - \|\vec{p_i} - \vec{p_j}\|^2)^3 & \text{si } \|\vec{p_i} - \vec{p_j}\|^2 < h^2 \\ 0 & \text{si } \|\vec{p_i} - \vec{p_j}\|^2 \geq h^2 \end{cases}$$

Ten en cuenta que el incremento de densidad $\Delta\rho_{ij}$ se aplica a ambas partículas: i y j.

$$\rho_i = \rho_i + \Delta\rho_{ij}$$
$$\rho_j = \rho_j + \Delta\rho_{ij}$$

No obstante, ten especial cuidado para evitar que una partícula se incremente dos veces con el mismo incremento (ya que $\Delta\rho_{ij} = \Delta\rho_{ji}$).

Transformación de densidad Para cada densidad calculada, se realiza una transformación lineal.

$$\rho_i = (\rho_i + h^6) \cdot \frac{315}{64 \cdot \pi \cdot h^9} \cdot m$$

Transferencia de la aceleración En este paso, para cada bloque de la malla de simulación, se consideran todos los bloques contiguos así como el bloque actual. Para cada par de partículas i y j, se evalúan las distancias.

Si las dos partículas están suficientemente cerca ($\|\vec{p_i} - \vec{p_j}\|^2 < h^2$) se actualiza la aceleración.

$$dist_{ij} = \sqrt{\text{máx}(\|\vec{p_i} - \vec{p_j}\|^2, 10^{-12})}$$

$$\Delta\vec{a}_{ij} = \frac{(\vec{p_i} - \vec{p_j}) \cdot \frac{15}{\pi \cdot h^6} \cdot m \cdot \frac{(h - dist_{ij})^2}{dist_{ij}} \cdot (\rho_i + \rho_j - 2 \cdot \rho) + (\vec{v_j} - \vec{v_i}) \cdot \frac{45}{\pi \cdot h^6} \cdot \mu \cdot m}{\rho_i \cdot \rho_j}$$

$$\vec{a}_i = \vec{a}_i + \Delta\vec{a}_{ij}$$

$$\vec{a}_j = \vec{a}_j - \Delta\vec{a}_{ij}$$

No obstante, ten especial cuidado para evitar que una partícula se incremente dos veces con el mismo incremento (ya que $\vec{a}_{ij} = \vec{a}_{ji}$).

4.3.3. Colisiones de partículas

En esta etapa, se tienen en cuenta las colisiones de las partículas con las paredes del recinto para actualizar el vector aceleración. Dado que los índices para un bloque de la malla son c_x, c_y y c_z, se tienen en cuenta las siguientes opciones:

- $c_x = 0$
- $c_x = n_x - 1$
- $c_y = 0$
- $c_y = n_y - 1$
- $c_z = 0$
- $c_z = n_z - 1$

Colisiones con los límites del eje x Cuando se fija c_x, para cualquier combinación de c_y y c_z se procesan todas las partículas. Primero, se calcula la coordenada x de la nueva posición.

$$x = \vec{p}_x + \overrightarrow{hv}_x \cdot \Delta t$$

Después, la diferencia con el límite se compara con el **tamaño de la partícula** (d_p).

$$\Delta x = \begin{cases} d_p - (x - x_{min}) & \text{if } c_x = 0 \\ d_p - (x_{max} - x) & \text{if } c_x = n_x - 1 \end{cases}$$

Si Δx es mayor que 10^{-10}, se actualiza la aceleración, teniendo en cuenta la **colisiones de rigidez** (c_s) y el **amortiguamiento** (d_v).

$$a_x = \begin{cases} a_x + c_s \cdot \Delta x - d_v \cdot v_x & \text{if } c_x = 0 \\ a_x - c_s \cdot \Delta x + d_v \cdot v_x & \text{if } c_x = n_x - 1 \end{cases}$$

Colisiones con los límites del eje y Cuando se fija c_y, para cualquier combinación de c_x y c_z se procesan todas las partículas. Primero, se calcula la coordenada y de la nueva posición.

$$y = \vec{p}_y + \overrightarrow{hv}_y \cdot \Delta t$$

Después, la diferencia con el límite se compara con el **tamaño de la partícula** (d_p).

$$\Delta y = \begin{cases} d_p - (y - y_{min}) & \text{if } c_y = 0 \\ d_p - (y_{max} - y) & \text{if } c_y = n_y - 1 \end{cases}$$

Si Δy es mayor que 10^{-10}, se actualiza la aceleración, teniendo en cuenta la **colisiones de rigidez** (c_s) y el **amortiguamiento** (d_v).

$$a_y = \begin{cases} a_y + c_s \cdot \Delta y - d_v \cdot v_y & \text{if } c_y = 0 \\ a_y - c_s \cdot \Delta y + d_v \cdot v_y & \text{if } c_y = n_y - 1 \end{cases}$$

Colisiones con los límites del eje z Cuando se fija c_z, para cualquier combinación de c_x y c_y se procesan todas las partículas. Primero, se calcula la coordenada z de la nueva posición.

$$z = \vec{p}_z + \overrightarrow{hv}_z \cdot \Delta t$$

Después, la diferencia con el límite se compara con el **tamaño de la partícula** (d_p).

$$\Delta z = \begin{cases} d_p - (z - z_{min}) & \text{if } c_z = 0 \\ d_p - (z_{max} - z) & \text{if } c_z = n_z - 1 \end{cases}$$

Si Δz es mayor que 10^{-10}, se actualiza la aceleración, teniendo en cuenta la **colisiones de rigidez** (c_s) y el **amortiguamiento** (d_v).

$$a_z = \begin{cases} a_z + c_s \cdot \Delta z - d_v \cdot v_z & \text{if } c_z = 0 \\ a_z - c_s \cdot \Delta z + d_v \cdot v_z & \text{if } c_z = n_z - 1 \end{cases}$$

4.3.4. Movimiento de partículas

En esta etapa, se actualizan todas las partículas.

$$\begin{aligned} \vec{p_i} &= \vec{p_i} + \overrightarrow{hv_i} \cdot \Delta t + \vec{a_i} \cdot (\Delta t)^2 \\ \vec{v_i} &= \overrightarrow{hv_i} + \frac{\vec{a_i} \cdot \Delta t}{2} \\ \overrightarrow{hv_i} &= \overrightarrow{hv_i} + \vec{a_i} \cdot \Delta t \end{aligned}$$

4.3.5. Interacciones con los límites del recinto

En esta etapa, se tienen en cuenta las colisiones de las partículas con los límites del recinto. Dado que los índices en un bloque de la malla son c_x, c_y y c_z, se tienen en cuenta las siguientes opciones.

- $c_x = 0$
- $c_x = n_x - 1$
- $c_y = 0$
- $c_y = n_y - 1$
- $c_z = 0$
- $c_z = n_z - 1$

Colisiones con límites en el eje x Cuando se fija c_x, para cualquier combinación de c_y y c_z se procesan todas las partículas.

$$d_x = \begin{cases} \vec{p}_x - x_{min} & \text{if } c_x = 0 \\ x_{max} - \vec{p}_x & \text{if } c_x = n_x - 1 \end{cases}$$

Solamente cuando d_x es negativo hay una colisión. En ese caso, se actualizan p_x, v_x y hv_x

$$p_x = \begin{cases} x_{min} - d_x & \text{if } c_x = 0 \\ x_{max} + d_x & \text{if } c_x = n_x - 1 \end{cases}$$
$$v_x = -v_x$$
$$hv_x = -hv_x$$

Colisiones con límites en el eje y Cuando se fija c_y, para cualquier combinación de c_x y c_z se procesan todas las partículas.

$$d_y = \begin{cases} \vec{p}_y - y_{min} & \text{if } c_y = 0 \\ y_{max} - \vec{p}_y & \text{if } c_y = n_y - 1 \end{cases}$$

Solamente cuando d_y es negativo hay una colisión. En ese caso, se actualizan p_y, v_y y hv_y

$$p_y = \begin{cases} y_{min} - d_y & \text{if } c_y = 0 \\ y_{max} + d_y & \text{if } c_y = n_y - 1 \end{cases}$$
$$v_y = -v_y$$
$$hv_y = -hv_y$$

Colisiones con límites en el eje x Cuando se fija c_z, para cualquier combinación de c_x y c_y se procesan todas las partículas.

$$d_z = \begin{cases} \vec{p}_z - z_{min} & \text{if } c_z = 0 \\ z_{max} - \vec{p}_z & \text{if } c_z = n_z - 1 \end{cases}$$

Solamente cuando d_z es negativo hay una colisión. En ese caso, se actualizan p_z, v_z y hv_z

$$p_z = \begin{cases} z_{min} - d_z & \text{if } c_z = 0 \\ z_{max} + d_z & \text{if } c_z = n_z - 1 \end{cases}$$
$$v_z = -v_z$$
$$hv_z = -hv_z$$

4.4. Almacenamiento de resultados

Durante el almacenamiento, el archivo de salida se abre para escritura en modo binario. Primeramente, se escribe los parámetros generales (partículas por metro y número de partículas). Posteriormente, se escriben todas las partículas. Ten en cuenta que para cada partícula solamente se escriben los vectores $\vec{p}$, $\overrightarrow{hv}$ y $\vec{v}$. El orden en el que las partículas deben escribirse en el archivo es en orden ascendente de identificadores. Esto es, el mismo orden en el que fueron leídos.

IMPORTANTE: Aunque todos los cálculos internos se ejecutarán utilizando doble precisión, los resultados finales se almacenarán usando simple precisión.

5. Tareas

5.1. Desarrollo de software

5.1.1. Programa a desarrollar

Esta tarea consiste en el desarrollo de una versión secuencia de la aplicación descrita utilizando C++20. Por favor, ten en cuenta que no se permite utilización de varios hilos de ejecución (*multithreading*) ni de paralelismo de ninguna clase.

Para el desarrollo del programa los equipos pueden considerar distintas alternativas como el uso de estructuras de arrays, arrays de estructuras o bien alternativas híbridas entre ambos enfoques.

5.1.2. Componentes

Se desarrollarán los siguientes componentes

- **sim**: Biblioteca con todos los componentes usados desde el programa principal.
- **utest**: Pruebas unitarias de todos los componentes de la biblioteca **sim**.
- **ftest**: Pruebas funcionales de la aplicación.
- **fluid**: Programa ejecutable con la aplicación.

A continuación, se describen algunos componentes con más detalle:

sim : Componentes de la biblioteca de simulación.
Contendrá, al menos, los siguientes archivos fuente:

- **progargs.hpp, progargs.cpp**: Tratamiento de argumentos para los parámetros de **main()**.
- **grid.hpp, grid.cpp**: Representación de la malla.
- **block.hpp, block.cpp**: Representación de un bloque.

utest : Pruebas unitarias.
Contendrá, al menos, los siguientes archivos fuente:

- **progargs_test.cpp**: Pruebas unitarias para **progargs**.
- **grid_test.cpp**: Pruebas unitarias para **grid**.
- **block_test.cpp**: Pruebas unitarias para **block**.

fluid : Programa con la aplicación.
Contendrá un único archivo fuente:

- **fluid.cpp**: Contendrá solamente una función **main()** para la aplicación. No incluirá ninguna función adicional.

5.1.3. Compilación del proyecto

Todos los archivos fuente deben compilar sin problemas y no emitirán ninguna advertencia del compilador. Se incluirá un archivo de configuración de CMake con las siguientes opciones:

CMakeLists.txt principal

```
cmake_minimum_required(VERSION 3.26)
project(fluid LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS OFF)

# Set compiler options
add_compile_options(-Wall -Wextra -Werror -pedantic -pedantic-errors)
set(CMAKE_CXX_FLAGS_RELEASE -march=native)

# Enable GoogleTest Library
include(FetchContent)
FetchContent_Declare(
        googletest
        GIT_REPOSITORY https://github.com/google/googletest.git
        GIT_TAG v1.14.0
)
set(gtest_force_shared_crt ON CACHE BOOL "" FORCE)
FetchContent_MakeAvailable(googletest)

# Enable GSL Library
FetchContent_Declare(GSL
        GIT_REPOSITORY "https://github.com/microsoft/GSL"
        GIT_TAG v4.0.0
        GIT_SHALLOW ON
)
FetchContent_MakeAvailable(GSL)
```

```
# Run clang-tidy on the whole source tree
# Note this will slow down compilation.
# You may temporarily disable but do not forget to enable again.
set(CMAKE_CXX_CLANG_TIDY clang-tidy -header-filter=.*)

# All includes relative to source tree root.
include_directories(PUBLIC .)

# Process cmake from sim and fluid directories
add_subdirectory(sim)
add_subdirectory(fluid)

# Unit tests and functional tests
enable_testing()
add_subdirectory(utest)
add_subdirectory(ftest)
```

El archivo de configuración de CMake para el directorio **sim** incluirá las siguientes opciones:

CMakeLists.txt para la biblioteca sim

```
# Add to this list all files related to sim library
add_library(sim
        progargs.hpp
        progargs.cpp
        grid.cpp
        grid.hpp
        block.cpp
        block.hpp
)

# Use this line only if you have dependencies from stim to GSL
target_link_libraries(sim PRIVATE Microsoft.GSL::GSL)
```

El archivo de configuración de CMake para el directorio **fluid** incluirá las siguientes opciones:

CMakeLists.txt para programa fluid

```
add_executable(fluid fluid.cpp)
target_link_libraries(fluid sim)
```

Ten en cuenta que estas reglas son solamente ejemplos.

Recuerda que todas las evaluaciones se realizarán con las optimizaciones del compilador activadas con la opción de CMake **-DCMAKE_BUILD_TYPE=Release**.

IMPORTANTE: La única biblioteca permitida es la biblioteca estándar de C++. No se permite bibliotecas externas.

EXCEPCIÓN: Se permite el uso de la biblioteca de soporte a las *C++ Core Guidelines*. La última versión se puede obtener en: `https://github.com/microsoft/GSL`.

5.1.4. Reglas de calidad de código

El código fuente debe estar bien estructurado y organizado, así como apropiadamente documentado. Se recomienda (aunque no se requiere) seguir las **C++ Core Guidelines** (`http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines`).

Se publicará un conjunto de reglas en un documento separado.

5.1.5. Pruebas unitarias

Se definirá un conjunto de pruebas unitarias que también se entregarán. Se recomienda, aunque no se requiere, el uso de GoogleTest (`https://github.com/google/googletest`).

En cualquier caso, se entregarán evidencias de que hay pruebas unitarias suficientes.

Si utilizas GoogleTest como marco de trabajo para pruebas unitarias, a continuación puedes ver un ejemplo de archivo CMake que puede resultarte útil.

CMakeLists.txt para pruebas unitarias

```
# Executable for all unit tests with list of sources
# For example, you may have one *_test.cpp for each *.cpp in sim
add_executable(utest
        particle_info_test.cpp
        math_vector_test.cpp
        block_test.cpp)

# Library dependencies
target_link_libraries(utest
        PRIVATE
        sim
        GTest::gtest_main
        Microsoft.GSL::GSL)

# Discover all tests and add them to the test driver
include(GoogleTest)
gtest_discover_tests(utest)
```

5.1.6. Uso de herramientas de IA

El uso de herramientas basadas en IA durante el proyecto está permitido. No obstante, se debe tener en cuenta lo siguiente:

- Si usas una herramienta basada en IA, se deben declarar los usos en la sección de diseño de la memoria del proyecto. No se realizará ninguna penalización en la declaración siempre que se incluya una declaración de uso.
- No se dará soporte al uso de dichas herramientas. En particular, si tienes preguntas sobre tu código, deberás ser capaz de explicar dicho código.
- Ten en cuenta que algunas herramientas de IA pueden generar código inseguro o código con bajo rendimiento.
- Cualquiera de tus profesores podrá requerirte una explicación de tu propio código.

5.2. Evaluación del rendimiento y la energía

Esta tarea consiste en la realización de una evaluación del rendimiento. Para llevar a cabo la evaluación del rendimiento, se medirá el tiempo total de ejecución. Además, también se medirá la energía y se derivará la potencia.

Todas las evaluaciones del rendimiento se realizarán en un nodo del clúster **avignon**.

Representa en una gráfica los tiempos totales de ejecución, uso de energía y potencia para distintos números de iteraciones.

La memoria del proyecto incluirá conclusiones derivadas de los resultados. Por favor, no te limites a una mera descripción de los datos. Trata de encontrar explicaciones convincentes de estos resultados.

6. Calificación

Las calificaciones finales para este proyecto se obtienen de la siguiente forma:

- Rendimiento: 15 %.
- Energía usada: 15 %.
- Pruebas unitarias: 10 %.
- Pruebas funcionales: 5 %.
- Calidad del diseño: 5 %.
- Calidad del código: 10 %.
- Evaluación de rendimiento y energía en la memoria: 15 %.
- Contribuciones de cada miembro: 20 %.
- Conclusiones: 5 %.

ADVERTENCIAS IMPORTANTES:

- Si el código entregado no compila, la nota final de la práctica será de 0.
- Si se ignora injustificadamente alguna norma de calidad de código, la nota final de la práctica será de 0.
- En caso de copia todos los grupos implicados obtendrán una nota de 0. Además, se notificará a la dirección de la EPS para la correspondiente apertura de expediente disciplinario.

7. Procedimiento de entrega

La entrega del proyecto se realizará a través de Aula Global.
Para ello se habilitarán 2 entregadores separados:

- **Entregador de memoria**. Contendrá la memoria del proyecto, que será un archivo en formato pdf con el nombre **report.pdf**.
- **Entregador de código**: Contendrá todo el código fuente necesario para compilar la aplicación.
  - Debe ser un archivo comprimido (formato zip) con el nombre **fluid.zip**.

La memoria no deberá exceder de 15 páginas con una fuente mínima de 10 puntos incluyendo la portada y todas las secciones. no se tendrá en cuenta en la corrección los contenidos a partir de la página 16 si fuese el caso. Deberá contener, al menos, las siguientes secciones:

- **Página de título**: contendrá los siguientes datos:
  - Nombre de la práctica.
  - Nombre del grupo reducido en el que están matriculados los estudiantes.
  - Número de equipo asignado.
  - Nombre y nia de todos los autores.
- **Diseño original**. Debe incluir el diseño de cada uno de los componentes. En esta sección se deben explicar y justificar cada una de las principales decisiones de diseño tomadas.

- **Optimización**. Debe contener una discusión de las optimizaciones aplicadas y su impacto. En particular, deberán indicarse optimizaciones realizadas sobre el código fuente original, así como optimizaciones activadas con flags de compilación adicionales que se estime oportuno.
- **Pruebas realizadas**: Descripción del plan de pruebas realizadas para asegurar la correcta ejecución. Debe incluir pruebas unitarias, así como pruebas funcionales de sistema.
- **Evaluación de rendimiento y energía**: Deberá incluir las evaluaciones de rendimiento y energía llevadas a cabo.
- **Organización del trabajo**: Deberá describir la organización del trabajo entre los miembros del equipo haciendo explícita las tareas llevadas a cabo por cada persona.
  - Debe contener una división del proyecto en tareas.
  - Las tareas deben ser suficientemente pequeñas como para que se pueda asignar una única persona a una tarea.
  - Todos los integrantes del equipo deberán realizar contribuciones relevantes en el diseño y construcción de componentes software.
  - No se podrá asignar más de una persona a una tarea. En tal caso, la tarea debe subdividirse en subtareas.
  - Se debe indicar el tiempo dedicado por cada persona a cada tarea.
- **Conclusiones**. Se valorará especialmente las derivadas de los resultados de la evaluación del rendimiento, así como las que relacionen el trabajo realizado con el contenido de la asignatura.