

# Movimiento Browniano

## Trabajo Practico Nro. 3

Badi Leonel, Buchhalter Nicolás Demián y Meola Franco  
Román

19 de abril de 2016

Grupo 3

# Fundamentos

## Introducción

- Movimiento Browniano
- Dinámica molecular dirigida por eventos con choques elásticos
- Las partículas siguen un movimiento rectilíneo uniforme entre colisiones

# Fundamentos

## Variables relevantes

- $N$  : número de partículas
- Dominio cuadrado de  $0,5m$  de lado
- **$N$  partículas pequeñas**
  - $R_1 = 0,005m$
  - $m_1 = 1kg$
- **Una partícula grande**
  - $R_2 = 0,05m$
  - $m_2 = 100kg$

# Implementación

## Generación de los agentes

- Posiciones  $(x, y)$  aleatorias para todas las partículas evitando la superposición de las mismas
- $-0,1 \frac{m}{s} < v_0 < 0,1 \frac{m}{s}$  en  $x$  e  $y$  para las partículas chicas
- $v_0 = 0$  para la partícula grande
- Todas las paredes son rígidas

# Simulación

## Variables relevantes

- $dt$  intrínseco variable
- $dt2$  constante
- **time** : Tiempo en segundos a visualizar
- $dt2 = \text{frameRate}$  : Número de frames por segundo

# Implementación

```
public double timeToNextColision(){
    Double t = Double.MAX_VALUE;
    for(int i = 0; i < N; j++){
        Particle particle = getParticle(i);
        Double calculatedTime = timeToBorder(particle, t);
        if(calculatedTime < t){
            t = calculatedTime;
        }
        for(int j = i+1 ; j < N; j++){
            Particle particle2 = getParticle(j);
            if(!particle.equals(particle2)) {
                if (calculateD(particle, particle2) >= 0) {
                    if (calculatedTime(particle, particle2) < t) {
                        colParticle1 = particle;
                        colParticle2 = particle2;
                        t = calculatedTime;
                    }
                }
            }
        }
    }
    return t;
}
```

Código 1: Función para obtener el tiempo de la próxima colisión.

# Implementación

```
public void simulate(double time, double frameRate){  
    writeFrame();  
    while(time > 0){  
        accumulatedTime += timeToNextCollision();  
        if(accumulatedTime < frameRate) {  
            moveSystem(timeToNextCollision());  
            collide(particle1, particle2); //particle2 == null cuando es borde  
        } else {  
            moveSystem(frameRate - (accumulatedTime - timeToNextCollision()));  
            writeFrame();  
            time -= frameRate;  
            for(j=2*frameRate; j<accumulatedTime && time> 0; j+=frameRate) {  
                time -= frameRate;  
                moveSystem(frameRate);  
                writeFrame();  
            }  
            moveSystem(accumulatedTime - (j - frameRate));  
            accumulatedTime = 0;  
        }  
    }  
}
```

Código 2: Algoritmo de simulación del sistema.

# Implementación

## Problemas encontrados

- Precisión del `double`: Al colisionar con bordes obtuvimos `timeToNextCollision` negativos
- Utilización de una heurística: Cell Index Method resultó muy costoso, aumentando el tiempo necesario para la simulación con los valores de  $N$  probados.

# Implementación

## Visualización

- La simulación y la visualización son independientes
- El algoritmo de simulación escribe un archivo .tsv con los siguientes datos:
  - $(x, y)$
  - $r$
  - Color RGB para indicar las velocidades, donde R es la componente en el eje Y y G es la componente en eje X
- Por último, se carga en Ovito el archivo de salida .tsv para realizar la visualización

# Resultados

Tiempo promedio para el cálculo de la simulación para distintos valores de  $N$

$N$	$t$
10	342 ms
100	4 seg
250	1 min 12 seg
500	18 min
1000	6hs 12 min

**Tabla:** Tiempo promedio para el cálculo de la simulación para distintos valores de  $N$ .

# Resultados

## Gráfico Frecuencia de colisiones

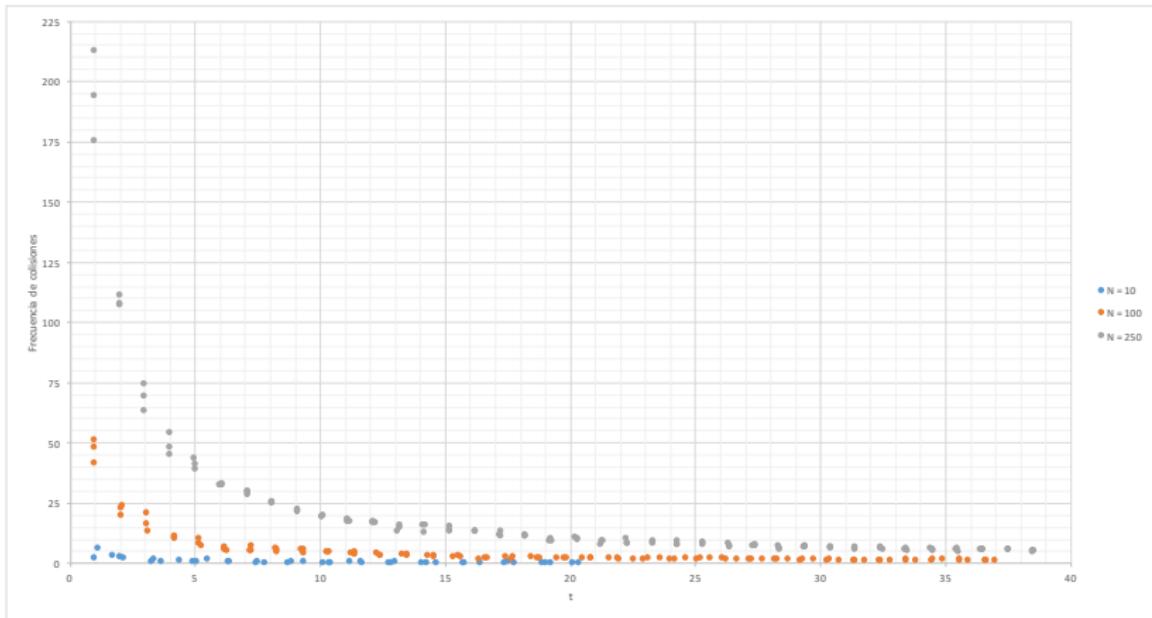


Grafico: Gráfico de la frecuencia de colisiones por unidad de tiempo

# Resultados

Frecuencia de colisiones para distintos valores de  $N$

$N$	$f \frac{\text{colisiones}}{s}$
10	0,76
100	5,25
250	22,33
500	86,3
1000	466,75

Tabla: Frecuencia de colisiones para distintos valores de  $N$ .

# Resultados

## Gráfico de la distribución de velocidades

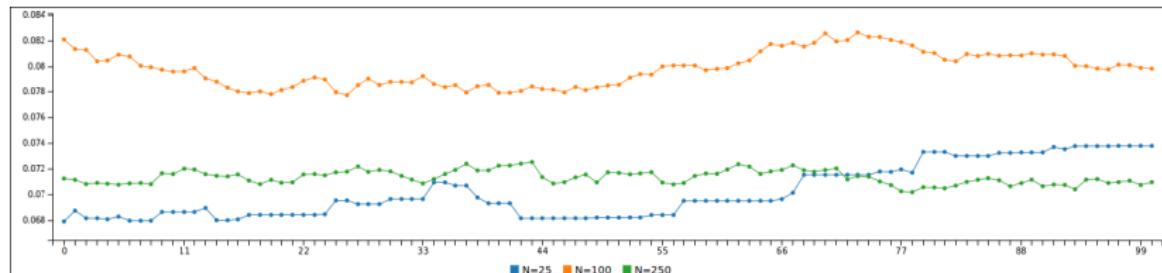
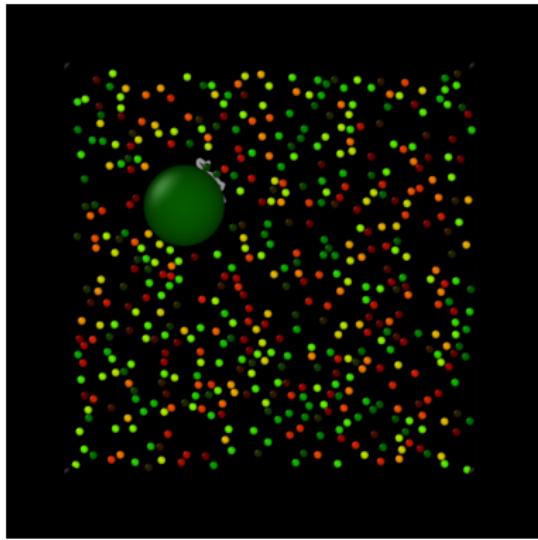


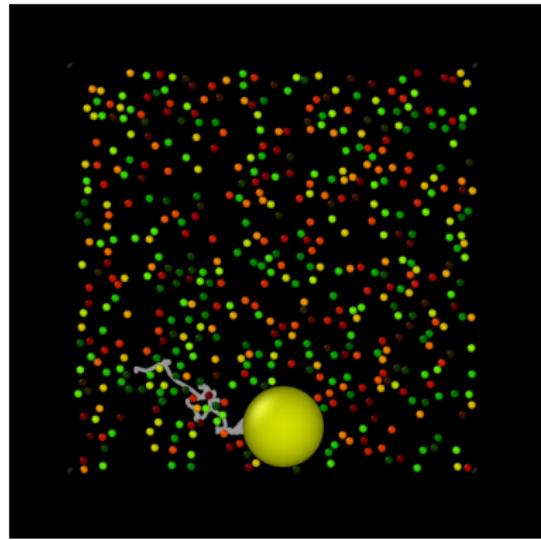
Grafico: Gráfico de la distribución de velocidades en el último tercio de la simulación

# Resultados

Gráfico de la trayectoria de la partícula grande



$$|v_0| = 0,1 \frac{m}{s}$$



$$|v_0| = 0,5 \frac{m}{s}$$

# Resultados

Gráfico de la trayectoria de la partícula grande

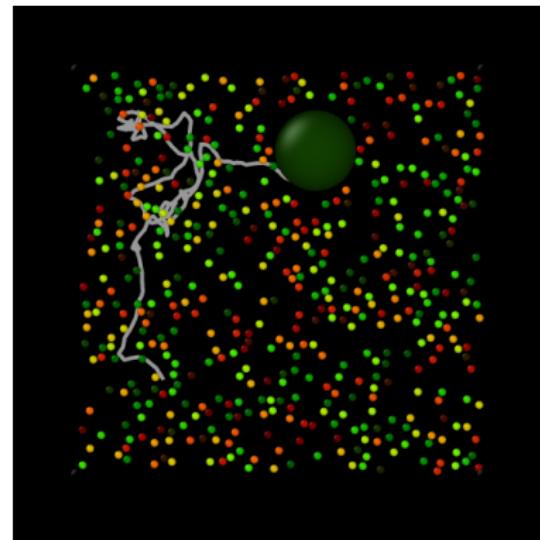


Grafico: Trayectoria de la partícula grande con  $|v_0| = 1 \frac{m}{s}$

# Resultados

Animación de la simulación para  $N = 100$

# Resultados

Animación de la simulación para  $N = 500$

# Resultados

Animación de la simulación para  $N = 1000$

# Conclusiones

- Aumentando la energía cinética se puede simular un aumento en la temperatura.  $K = nRT$
- Con  $N = 1000$  el comportamiento se asemeja a un material viscoso, existiendo un límite definido por las dimensiones del dominio.
- A mayor  $N$  la partícula grande se desplaza menos.
- Aumentando la velocidad de las partículas, aumentan la frecuencia de colisión y el tiempo de procesamiento.
- El sistema no se estabiliza ya que todas las fuerzas son conservativas.

# Gracias