Facultad de Ciencias

Máster en Modelización e Investigación Matemática, Estadística y Computación

Trabajo de Fin de Máster :

Diagrama fundamental del movimiento de peatones: Efecto de la competitividad

Realizado por Iñaki Echeverría Huarte

Dirigido por:
Dr. Raúl Cruz Hidalgo

Ponente:
Dr. Juan José Mazo Torres

Resumen

En este trabajo se ha estudiado numéricamente el proceso de evacuación de un conjunto de personas a través de una salida estrecha. Con este objetivo, se implementaron modelos de elementos discretos (DEM), de sistemas compuestos por partículas no esféricas autopropulsadas (peatones). Así, se desarrollaron nuevas metodologías de interacción entre peatones teniendo en cuenta la forma de la partícula y su correlación con la dirección deseada de movimiento. Para ello, se hizo uso de la estructura paralela de las unidades de procesamiento gráfico (GPU). Se ejecutó un estudio sistemático, ajustando varios parámetros del modelo, con el objetivo de reproducir condiciones experimentales específicas. Así, se varió la agitación de los individuos y la tendencia a mantener una dirección de movimiento deseada, manteniendo constante la magnitud de la velocidad deseada. Como punto de partida, hemos utilizado valores de flujo obtenidos experimentalmente, para validar si el modelo presenta una respuesta coherente. Además, se exploraron propiedades micromecánicas del sistema, como son: orientación de los peatones en posiciones cercanas a la puerta de salida y la proyección de estas sobre la dirección deseada. Finalmente, y motivados por los resultados experimentales muy recientes, hemos añadido un grado de complejidad al sistema, introduciendo un obstáculo. Las conclusiones que hemos obtenido han resultado del todo interesantes, en general obtenemos que introducir un obstáculo no resulta beneficioso para mejorar el flujo de personas. Así, nuestros resultados parecen contradecir opiniones preliminares de que la presencia de un obstáculo favorecería el proceso de evacuación. Sin embargo, reproducimos de manera fidedigna resultados experimentales, obtenidos muy recientemente, en el departamento de Física y Matemáticas de la Universidad de Navarra.

Abstract

In this work, the evacuation process of a group of pedestrians through a narrow exit is studied numerically. With that aim, we implement discrete element models (DEM) of systems composed of self-propelled non-spherical particles. Thus, new methodologies of interaction between pedestrians are developed. As a novelty, these methodologies take into account the shape of the particle and its correlation with the desired direction of movement. Moreover, in the numerical implementation, the parallel structure of the graphics processing units (GPU) is used. A systematic study is carried out, adjusting several parameters of the model and reproducing specific experimental conditions. Thus, we vary the agitation of the individuals, and the tendency to maintain a desired direction of movement, keeping constant the magnitude of the desired velocity. As a starting point, we used experimental flow values to validate whether or not the model shows a coherent response. Also, micromechanical properties of the system are explored, such as the orientation of pedestrians as well as their projection on the desired direction. Finally, and motivated by very recent experimental results, we add a degree of complexity to the system, introducing an obstacle. The conclusions obtained are quite interesting. In general, we find that introducing an obstacle is not beneficial to reduce the evacuation time (improve the flow rate). Thus, our results seem to contradict preliminary opinions that the presence of an obstacle would favor the evacuation process. However, we reproduce in a reliable way experimental results, obtained very recently by researchers in the Department of Physics and Mathematics of the University of Navarra.

Índice general

Capítulo 1

Introducción y objetivos

El comportamiento competitivo que siguen las multitudes en condiciones libres y en confinamiento, puede hoy observarse en muchas circunstancias. Desde salidas de emergencia en situaciones de riesgo [1], hasta compras agresivas (*black friday*, rebajas, etc) [2, 3]. No cabe duda que la comprensión y simulación de este tipo de flujos en recintos confinados mejoraría el diseño de las instalaciones peatonales, consiguiendo así mejorar el rendimiento en términos de seguridad.

En los últimos años, se han desarrollado una cantidad importante de modelos de peatones para estudiar el efecto de la competitividad sobre la respuesta global de estos sistemas. Los enfoques que se han tomado han sido principalmente dos: **los autómatas celulares** [4] y **los modelos basados en la fuerza** [5].

El primero de los enfoques trabaja sobre un espacio de tiempo discreto. Además, parten de la premisa de que no se modelan fuerzas propiamente dichas, sino que los choques se definen a partir del intercambio de velocidades de acuerdo a una serie de reglas de colisión prefijadas. Este tipo de modelos son ampliamente aceptados dada su rápida ejecución computacional, permitiendo simular sistemas con una gran cantidad de individuos.

Por otro lado, los modelos basados en la fuerza de interacción, trabajan en espacios continuos de tiempo, proporcionándonos una descripción más natural de la dinámica del sistema y permitiéndonos calcular fuerzas y presiones de contacto entre individuos. En este sentido, uno de los métodos numéricos más extendido y utilizado en el contexto de sistemas de partículas es el algoritmo de **dinámica molecular** (MD). Este método fue desarrollado inicialmente para observar la interacción entre átomos y moléculas para más tarde ser extendido en nuevos contextos discretos como el de los sistemas de peatones. En este trabajo se ha utilizado el **método de elementos discretos (DEM)**(pasará a detallarse dicho algoritmo en el capítulo 3 del trabajo) [6, 7, 8] , que no es más que una extensión del algoritmo MD incluyendo la componente rotacional en el movimiento. Además, se ha realizado la proyección de peatones en un espacio bidimensional y se ha utilizado como escenario de simulación un proceso de evacuación de una habitación individual en condiciones altamente competitivas.

A día de hoy, la simulación de sistemas mediante algoritmos de dinámica molecular u otros métodos de elementos discretos es complejo y costoso desde el punto de vista computacional. Debido a que el número de partículas y procesos de cálculo son realmente altos, es necesario proponer métodos alternativos de paralelización y optimización de software que nos permitan la obtención de resultados en intervalos de tiempo razonables. Es por ello que en este trabajo, ha sido necesario la utilización y diseño de algoritmos híbridos CPU-GPU [9] que han permitido dar respuesta a dicha problemática. No obstante, es necesario aclarar que nuestro programa está en fase de validación. Así, hemos usado tamaños de sistemas pequeños, para poder comparar con resultados experimentales reales.

Con este esquema, pretenderemos satisfacer ciertos objetivos: desde la comprensión, diseño e implementación de algoritmos DEM en arquitecturas híbridas CPU-GPU, hasta la descripción global del comportamiento de la evacuación, estadísticas de los tiempos de salida y papel que juegan la variación de ciertos parámetros en el modelo.

Capítulo 2

Introducción al cálculo sobre GPU. CUDA

¿Por qué utilizar GPU como herramienta de cálculo?

La unidad de procesamiento gráfica o **GPU** no es más que un chip programable integrado en la tarjeta gráfica, que originalmente ha sido utilizado en la visualización y procesado de imágenes en 3D. La alta capacidad de este dispositivo en la realización de operaciones sencillas es debido al alto número de procesadores del que dispone (cientos o miles dependiendo de la arquitectura concreta de la tarjeta), además de unidades de almacenamiento externas. Dicha estructura está conectada a la placa base del ordenador y se comunica con la CPU (Unidad Central de Proceso) a través de una conexión de alta velocidad.

A día de hoy, el uso de GPUs es mucho más variado, desde su utilización en áreas financieras como científicas. Al ser un dispositivo programable, su adaptación en diferentes áreas es completa, permitiéndo un desarrollo de software muy variado con un compromiso de eficiencia muy alto.

Dicha eficiencia, como ya se ha comentado, radica en que las GPU han sido desarrollada para la realización de tareas sencillas y repetitivas con diferentes conjunto de datos, lo que nos permite, dada su naturaleza, obtener paralelizaciones de código muy óptimas [9]. Esta nueva metodología se contradice con la que se presentaba antiguamente con el uso de las CPU, en donde la ejecución del código era secuencial, repercutiendo directamente en la eficiencia global de las aplicaciones.

Uno de los aspectos más fundamentales en el uso de GPU dentro del ámbito científico, es su sencilla programación, mantenimiento y eficiencia, permitiendo desarrollar modelos que hasta la fecha precisaban de altos tiempos de ejecución. En este sentido, hay que agradecer a NVIDIA su compromiso y desarrollo en un conjunto de herramientas integradas y optimizadas como bibliotecas estandar de C/C++ que han permitido al programador desarrollar aplicaciones en

paralelo sobre GPU de forma sencilla. La gran novedad llega hace 6-7 años con la creación de **CUDA** (*Compute Unified Device Architecture)* [10], una plataforma conjunta de hardware y software que, conectando CPU con GPU, han evitado al desarrollador trabajar directamente con arquitecturas de hardware en lenguaje ensamblador.

Dada la arquitectura propia de GPU, se hace necesario seguir una metodología de programación de tipo multihilo (*multithreading*) [9]. Trataremos la GPU como un coprocesador, denominado *device* que trabaja al servicio de la CPU o *host*. De forma general, una GPU está compuesta por un conjunto de N multiprocesadores (SM), que a su vez están compuestos por M procesadores escalares (SP o *core*). El número de cores de la GPU dependen de la generación de la tarjeta gráfica, presentando actualmente modelos con 28 multiprocesadores de 128 cores cada uno (modelo NVIDIA *GeForce GTX 1080 Ti*). La arquitectura de una de nuestras tarjetas gráficas es la basada en el modelo NVIDIA *GeForce GTX 580* que nos ofrece 8 multiprocesadores de 192 cores cada uno (se presenta un esquema de la arquitectura de la tarjeta en Fig. 2.1) .

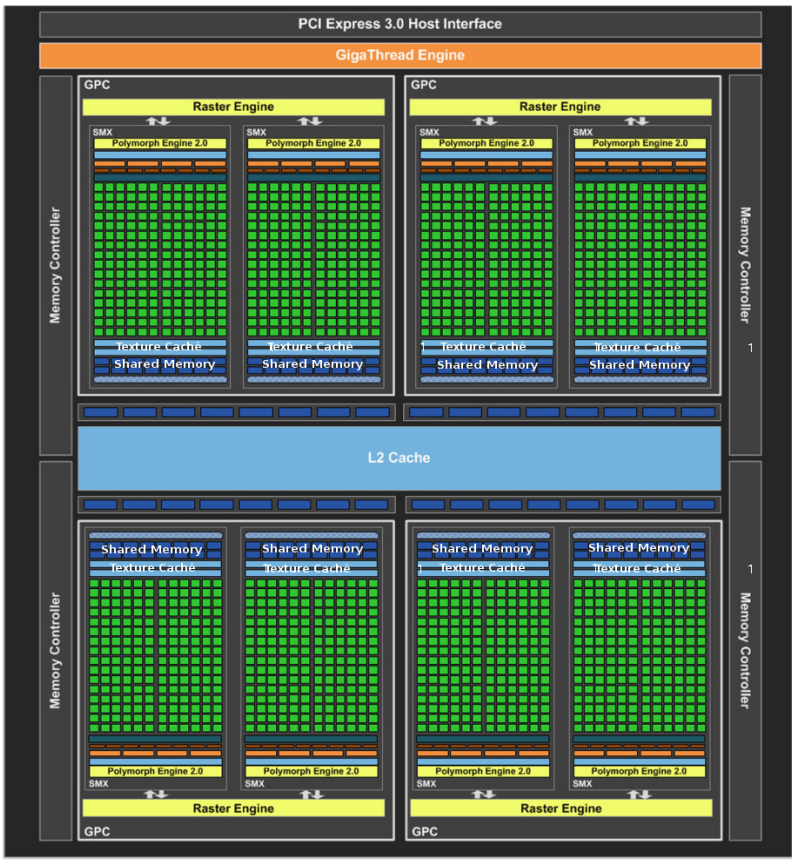


Figura 2.1: Esquema de conexión CPU-GPU. Se muestra la arquitectura de una GPU señalando las diferentes regiones de la memoria.

En cada ciclo, cada multiprocesador ejecuta una operación sobre todos sus cores, con total independencia del resto. Cada unidad secuencial ejecutada sobre un core se denomina hilo. Los datos que utiliza cada uno de estos cores son distintos, lo que permite una fácil paralelización en las aplicaciones CUDA. Esta asignación de tareas a los distintos nucleos de la tarjeta gráfica es posible gracias a la forma en que cada multiprocesador gestiona el uso de sus cores. Cada multiprocesador crea, administra y organiza la ejecución de un bloque de forma que sus hilos se organizan en grupos de 32 hilos consecutivos denominados *warps*.

En cuanto a la memoria y almacenamiento de datos y resultados, existen dos tipos:

1. **Memoria interna de la GPU**: También denominada memoria *on chip*, se subdivide a su vez en dos tipos:
   - **Memoria de Registros:** Ubicada en cada uno de los cores y únicamente accesible individualmente, se trata de una memoria de lectura y escritura con una capacidad de unos pocos kB.
   - **Memoria Compartida:** Ubicada en cada multiprocesador, es accesible únicamente por los cores que lo componen. Tiene un acceso algo más lento, pero permite originar cierto grado de comunicación entre los diferentes hilos en ejecución.
2. **Memoria integrada en la tarjeta gráfica:** También denominada *off chip*, está compuesto por tres nuevos tipos:
   - **Memoria Global:** Permite la comunicación entre GPU y CPU al tratarse de una memoria de lectura y escritura a la que se tiene acceso desde el *device* y *host*. El uso de este memoria ralentiza extremadamente el programa, aunque su uso es completamente imprescindible.
   - **Memoria de constantes:** Pese a tratarse de una memoria externa a la GPU, es una memoria de rápido acceso. Únicamente tiene accesos de escritura desde el *host*, dejando únicamente los permisos de lectura a los hilos en ejecución.
   - **Memoria de texturas:** Con los mismos permisos que la memoria de constantes con la salvedad de que está optimizada para el acceso a posiciones consecutivas

Introducción a la programación en CUDA

Llegados a este punto, es necesario conocer cómo se realiza el desarrollo de aplicaciones en CUDA. Las funciones que esta plataforma utiliza son los denominados ***kernels***. La particularidad de estas funciones reside en que su ejecución está subordinada a la invocación desde CPU, mientras que la ejecución se realiza en los distintos hilos que componen la GPU, trabajando a la vez con diferentes tandas de datos.

Las limitaciones que presenta la implementación de este tipo de estructuras es que deben ser funciones tipo ***void*** (siguiendo el estandar de C) [11], por lo que no se permite que se devuelva ningún parámetro de salida tras su invocación. Este resultado es lógico siguiendo la metodología de paralelizaje que se ha comentado, dado que resultaría incoherente almacenar en una sola posición de memoria el resultado de los distintos hilos en ejecución. A nivel sintáctico, CUDA implementa en sus librerías la etiqueta **__global__** que seguida de *void*, el nombre de la función y los parámetros de entrada, originan la declaración del kernel. Además, es necesario conocer si el diseño de la función esta creado para ser invocado desde el *host*, desde el *device* o desde cualquiera de ellos. En caso de querer ejecutar dicha función sobre un hilo, se debe añadir **__device__** al encabezado de la función. Por defecto, si la función carece de etiqueta, únicamente puede ser ejecutada desde CPU, pero por convenio se añade la etiqueta **__host__** para evitar confusiones. Se añadirán sendas etiquetas en caso de que se quiera poder ejecutar la función tanto desde el *device* como desde el *host*.

A todo lo comentado, el programador debe añadir el número de hilos dedicados a la ejecución de la función sobre el *device* en base a las necesidades del algoritmo. El número de hilos dedicados pueden agruparse en estructuras denominadas **bloques**, cuya finalidad radica en organizar la distribución de hilos en los multiprocesadores. La dimensión de estos bloques la predefine el programador y se almacena en una variable de tipo *blockDim*. Además, cada bloque viene acompañado de un identificador, *blockIdx*, que nos permitirá distinguirlos para su acceso durante la ejecución del programa. Además, CUDA permite la agrupación de bloques en estructuras llamadas **grid**. Un *grid* es un conjunto de bloques de hasta tres dimensiones.

En cuanto a la declaración de las variables, el programador puede determinar el tipo de memoria sobre la que se almacenarán estas. Principalmente existen 3 tipos de asignación de memoria para las variables:

1. Memoria de acceso privado para cada hilo, en donde lo común es hacer uso de la memoria de registros cuya latencia es muy baja. Dado que la capacidad de este tipo de memoria es limitado, CUDA hace uso de otro sector de memoria *off chip*, denominada memoria local, cuya velocidad de acceso es algo mayor que lo común. La declaración de las variables que hagan uso de este tipo de asignación de memoria es sin etiqueta.

2. En caso de querer utilizar una variable común a varios hilos, se precisará de la memoria compartida, teniendo una latencia algo mayor que la memoria de acceso privado pero mucho menor que la memoria global. En este caso, en la definición de la variable será preciso hacer uso de la etiqueta *__shared__*.

3. Por último, nos encontramos las variables que se asignan a la memoria global, permitiendo la comunicación entre el *host* y el *device*. Este tipo de asignación tiene acceso de lectura y escritura, haciendo que la latencia sea muy alta comparada con las hasta ahora vistas. Antes de la ejecución del *kernel*, es necesario volcar el conjunto de datos necesarios a la

memoria global de GPU, para que puedan ser usados por los hilos en el *device*. Este es un aspecto muy importante, dado que al inicio de los programas, es necesario realizar la definición de punteros a memoria global para poder almacenar dichos datos. Este proceso se realiza empleando la función predefinida en CUDA, *cudaMalloc(puntero, longitud en bytes)*.

En una segunda etapa, se realizará la transferencia de datos haciendo uso de las funciones predefinidas de CUDA, *cudaMemcpyHostToDevice* o *cudaMemcpyDeviceToHost*. Cada una de ellas realiza la transferencia en un determinado sentido (de CPU a GPU, o viceversa).

Finalmente, cabe destacar que CUDA permite la definición de variables con la etiqueta *__constant__* que se almacenan en la memoria de constantes. Su acceso es bastante rápido al dar permisos únicamente de lectura desde los hilos. La definición debe realizarse desde el *host*.

Un diagrama que resume cada una de las diferentes gestiones comentadas, es el mostrado en la Fig 2.2.

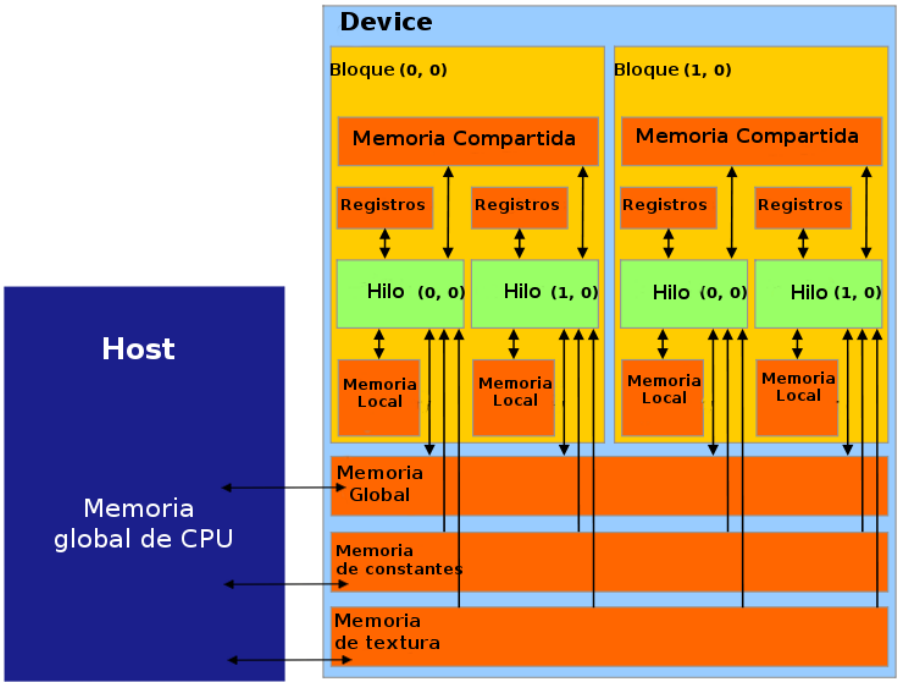


Figura 2.2: Jerarquía en la asignación de memoria en CUDA. Las flechas indican las distintas regiones de memoria con permisos de escritura y lectura a los que se tienen acceso partiendo de los distintos hilos y bloques con forman parte de la arquitectura de la GPU.

Programando en CUDA

Como último punto del capítulo, mostraremos los principales aspectos a tener en cuenta a la hora de crear código para ser implementado sobre GPU y mostraremos un ejemplo sencillo que lo ilustre.

Estructuración del programa

Toda aplicación CUDA está subordinada a un proceso principal cuya ejecución se realiza desde CPU. Así pues, previamente a la invocación de cualquier kernel, es preciso gestionar los recursos de memoria global de CPU, así como relizar una transferencia de memoria desde CPU a GPU. Recordar que estos dos procesos se llevan a cabo mediante las funciones *cudaMalloc* y *cudaMemcpy* respectivamente. Además, es preciso determinar el número de hilos y bloques asignados a cada kernel. Comentar que estos valores no tienen por qué ser fijos para todos los kernels definidos, sino que se trata de parámetros que son ajustables con el fin de optimizar la eficiencia del software que se esté desarrollando.

Así pues, una vez realizados los procesos comentados, el programa principal realiza una llamada al kernel, quedando inactivo hasta que todos los hilos finalizan la tarea asignada. En tiempo de ejecución, se realiza la creación y gestión de los bloques, se distribuyen por los multiprocesadores y se realiza el desglose en *warps* (conjuntos de 32 hilos).

Finalmente, tras la finalización de la invocación del kernel, debe realizarse una liberación de memoria global en GPU desde el host mediante la función *cudaFree*.

Con el fin de resumir todo este proceso, procedemos a mostrar un ejemplo concreto de un primer programa en CUDA.

Ejemplo. Suma de dos vectores.

En este apartado, se presenta un algoritmo que permite realizar la suma de dos vectores de dimensión prefijada, *dim*. Así pues, estructuraremos el proceso en dos partes. En la primera de ellas desarrollaremos el código de CPU que nos permita realizar la ejecución del kernel. Finalmente, introduciremos la función que caracteriza dicho kernel.

Código de CPU

Dado que es el *host*, en la CPU, quien gobierna toda la ejecución, es sobre él donde se realizan las primeras acciones de puesta a punto de los recursos de memoria. Así pues, inicialmente se realiza una declaración sobre GPU de los vectores sobre los que se realizará las distintas operaciones. A su vez, también será necesario declararlos sobre CPU. Para ello, se genera un

puntero a una variable del mismo tipo que el vector de CPU, y se reserva tanta memoria como ese vector ocupase sobre CPU. Esto se hace a través de la ya mencionada función *cudaMalloc*.

En un segundo paso se transfiere el contenido de las variables de la CPU a las nuevas de GPU. Esta transferencia de datos se hace con la función *cudaMemocpy*, empleando la etiqueta *cudaMemcpyHostToDevice* que indica el sentido de la transición (de CPU a GPU).

En base a la dimensión del vector y fijando el tamaño de los bloques, en este caso *n_hilos_bloque* = 32 (tamaño del Warp), se determina el número de bloques que deben ejecutarse. Así, se lanza el kernel con la notación habitual:

```
//  Declaracion de las variables de CPU
  float h_VA[dim];
  float h_VB[dim];
  float h_VC[dim];

//  Declaracion de las variables de GPU
  float *d_VA, *d_VB, *d_VC

//  Reserva de memoria sobre GPU
  cudaMalloc ((void**)&d_VA, dim*sizeof(float));
  cudaMalloc ((void**)&d_VB, dim*sizeof(float));
  cudaMalloc ((void**)&d_VC, dim*sizeof(float));

//  Manipulacion de datos en CPU
  .........

//  Transferencia de datos de CPU a GPU
  cudaMemcpy(d_VA,h_VA,dim*sizeof(float),cudaMemcpyHostToDevice);
  cudaMemcpy(d_VB,h_VB,dim*sizeof(float),cudaMemcpyHostToDevice);

//  Asignacion de las dimensions de grid y bloque
  n_hilos_bloque = 32;
  n_bloques = ceil (dimension/n_hilos_bloque);

//  Ejecucion del Kernel
  Suma_vectores <<< n_bloques, n_hilos_bloque >>> (d_VA, d_VB, d_VC, dim);

//  Transferencia de los datos de GPU a CPU
  cudaMemcpy(h_VC,d_VC,dim*sizeof(float),cudaMemcpyDeviceToHost);

//  .......

//  Liberacion de recursos
  cudaFree(d_VA)
  cudaFree(d_VB)
  cudaFree(d_VC)
```

Tras la ejecución del kernel, la CPU sólo debe hacer una transferencia del contenido de la variable de vectores a la variable de CPU con la función *cudaMemcpy*, esta vez identificada como *cudaMemcpyDeviceToHost*, (de GPU a CPU). Finalmente, se libera el espacio reservado a las variables sobre GPU con *cudaFree*

Código de GPU

El kernel para resolver este problema es muy sencillo. Cada hilo obtiene su índice, almacenado en la variable *index*, el cual depende del bloque al que pertenece y de la posición que ocupa dentro del mismo. Este índice estará asociado con las componentes de los vectores A y B sobre la que se computará la suma. Una vez obtenido dicho valor, este se almacenará en la posición correspondiente a su índice en el vector C.

Recordar que es recomendable que la dimensión de los bloques sea un múltiplo del tamaño del *warp* y que, en la medida de lo posible, el número de bloques sea un divisor de la dimensión del vector.

Así las cosas, el código implementado sería:

```
1 _global_ void Suma_vectores (float *A, float *B, float *C, int dim)
2 {
3   int index;
4   index = threadIdx.x + (blockIdx.x * blockDim.x);
5   if(index < N ){
6     C[index] = A[index] + B[index];
7   }
8 }
```

Capítulo 3

Método de Elementos Discretos (DEM)

Como ya se ha comentado, el método numérico que ha sido utilizado en este trabajo para la modelización del sistema ha sido el **método de elementos discretos** (DEM). Este algoritmo no es más que una extensión de los algoritmos de tipo dinámica molecular, presentando muchas similitudes pero algunas diferencias importantes de resaltar.

La particularidad de los algoritmos de dinámica molecular y que puede **no estar presente** en los algoritmos de tipo DEM, es que la fuerza ejercida entre las partículas puede ser descrita en términos de un potencial de interación. Dado que en nuestro modelo se incluyen otro tipo de fuerzas, debemos considerarlo un algoritmo de tipo DEM [8].

A efectos numéricos, los algoritmos de tipo DEM calculan las fuerzas y torques ejercidos sobre cada una de las partículas que conforman el sistema para posteriormente resolver las ecuaciones de Newton (tanto traslacionales como rotacionales) y de esta forma describir la trayectoria de cada partícula de manera más realista.

A lo largo de este tercer capítulo, describiremos el modelo utilizado que analiza el comportamiento de peatones evacuando un espacio confinado determinado. Explicaremos las peculiaridades que surgen a la hora de implementar este tipo de algoritmos en arquitecturas GPU [12] y comentaremos las generalidades que se han llevado a cabo para poder realizar una modelización de los peatones de formas más complejas a la esférica. Por último, se detallarán aspectos importantes sobre la implementación de condiciones de contorno especiales que han permitido recrear situaciones experimentales más realistas.

Método de elementos discretos (DEM) implementado sobre GPU

Descripción del modelo utilizado

Con el fin de dar una interpretación y modelización más sencilla al problema, el algoritmo utilizado trabaja sobre un **espacio bidimensional**. Así las cosas, comenzamos con la descrip-

ción en la evolución de la componente traslacional. Al principio de la simulación, los peatones se disponen sobre el espacio establecido, respetando las condiciones geométricas de no ocupación de un espacio ya ocupado por otro peatón y actualizan su posiciones iniciales en función de una velocidad dada.

Conforme el sistema evoluciona, surge la peculiaridad de que dos individuos no deben compartir regiones espaciales comunes, por lo que se introduce una fuerza de contacto de la forma:

$$\overrightarrow{F}_{G_i} = \sum_{j}^{N_c} \overrightarrow{F}_{ij} + \overrightarrow{F}_{\omega_i} \tag{3.1}$$

En donde $\overrightarrow{F_{\omega_i}}$ es la fuerza de contacto de los peatones con la pared y $\overrightarrow{F}_{ij}$ es la fuerza de contacto del peatón i con el conjunto de individuos j que sean vecinos (N_c) (posteriormente se detallará las expresiones de cálculo para las fuerzas de interacción entre partículas - Sec. 3.1.2 - así como el cálculo de las distancias de solapamiento entre dos partículas - Sec. 3.2 -).

Además, se ha considerado que los peatones son **partículas autopropulsadas**, con un mecanismo de autopropulsión que surgue a través de la introducción de una fuerza de tipo *drift* de la forma:

$$\overrightarrow{F}_{D_i} = m_i \frac{(\overrightarrow{v}_d - \overrightarrow{v}_i)}{\tau} \tag{3.2}$$

En donde m_i es la masa del peatón, $\overrightarrow{v}_i$ es su velocidad y $\overrightarrow{v}_d = v_d \cdot \hat{e}_i$ da cuenta de la velocidad deseada (se introduce como parámetro del modelo y su valor se estima a raiz de los experimentos realizados en [13]). En cuanto al vector $\hat{e}_i$ no es más que el vector unitario que tiene como dirección la recta que une el punto más cercano entre la posición del peatón y la puerta de salida. Finalmente, τ representa el tiempo característico que necesitaría el peatón para conseguir una velocidad v_d en caso de que no existieran interacción con otros peatones (se incluye como parámetro del modelo y su estimación se determina a partir de resultados experimentales presentes en [14]).

Así las cosas, las componentes traslacionales del movimiento de cada partícula, $i = 1, \cdots N$, se obtienen integrando la segunda ley de Newton:

$$\overrightarrow{F}_{D_i} + \overrightarrow{F}_{G_i} = m_i \overrightarrow{\ddot{r}}_i \tag{3.3}$$

El algoritmo integrador que se ha utilizado ha sido uno del tipo *verlet-velocity* en dos pasos.

En cuanto a la componente angular, se introducen dos términos. El primero de ellos es el más común y surge a partir de las propias fuerzas de interacción entre los peatones. Lo expresamos

como:

$$\overrightarrow{\tau}_{G_i} = \sum_{j}^{N_c} \overrightarrow{r}_{ij} \times \overrightarrow{F}_{ij} \tag{3.4}$$

donde $\overrightarrow{r}_{ij}$ es el vector que apunta desde el centro de la partícula i al punto de contacto con la partícula j.

Además, se propone un segundo término que pretende originar orientaciones privilegiadas sobre los peatones. Así pues, se define un vector unitario $\hat{n}_e$ que teniendo en cuenta el modelado de los peatones, será un vector perpendicular a la orientación de la partícula. Dicho vector se comparará con el vector $\hat{e}_i$ previamente definido, dando lugar a la existencia de un torque *impulsor* de la forma:

$$\overrightarrow{\tau}_D = -S_D \overrightarrow{\Delta\theta} - \beta \overrightarrow{\dot{\theta}} + \overrightarrow{R}(t) \tag{3.5}$$

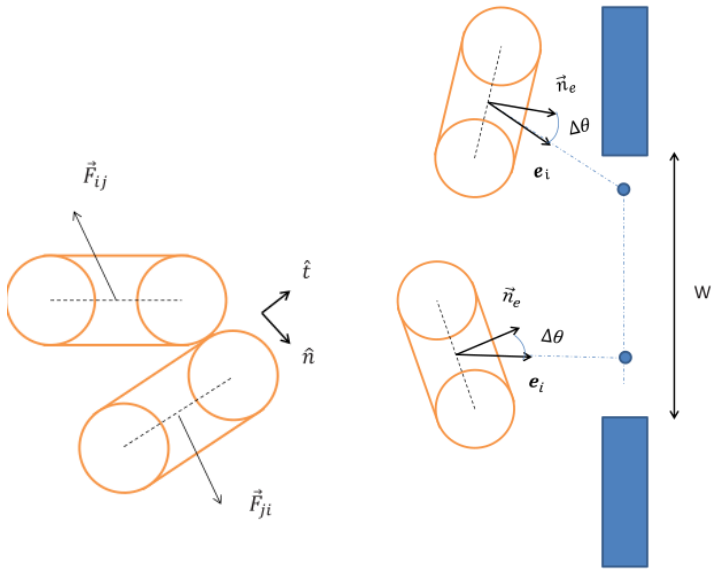
En donde $\overrightarrow{\Delta\theta}$ es la diferencia angular entre $\hat{n}_e$ y $\hat{e}_i$ y S_D es la rigidez angular. También se incluye un término de viscosidad, el cual refleja la dependencia de la velocidad angular $\overrightarrow{\dot{\theta}}$ a través del coeficiente de amortiguación β. En todo momento se ha usado un valor de $\beta = 4{,}5\sqrt{S_D}$ garantizándonos así condiciones de sobreamortiguamiento [14]. Finalmente, se ha introducido un ruido sinusoidal $\overrightarrow{R}(t)$ que busca representar la tendencia del peatón para girar su cuerpo cuando queda atrapado en una posición dada. $\overrightarrow{R}(t)$ se caracteriza por una amplitud η (se introduce como parámetro del modelo) y un periodo T, que se establece en $T = 1{,}0$ segundos, en todos los casos. La introducción de esta periodicidad marca la diferencia con los tipos de ruido usados con anterioridad [15] y su determinación es puramente experimental basándonos en los simulacros de evacuación donde los peatones atascados parecen realizar movimientos rotacionales durante lapsos de tiempo del orden de 1 segundo.

Finalmente, las componentes angulares del movimiento se obtienen a partir de las ecuaciones de Euler:

$$\overrightarrow{\tau}_{D_i} + \overrightarrow{\tau}_{G_i} = \overrightarrow{I}_i \cdot \overrightarrow{\omega}_i \tag{3.6}$$

En donde $\overrightarrow{I_i}$ es el vector momento de inercia de la partícula i y $\overrightarrow{\omega_i}$ su velocidad angular.

Una representación gŕafica de los diferentes vectores y ángulos definidos a lo largo del modelo puede verse en la figura 3.1.



(a) Definición del vector normal y tangencial en la fuerza de interacción $\overrightarrow{F}_{ij}$ entre dos peatones.

(b) Diferencia de ángulos ($\Delta\theta$) entre la orientación original del peatón ($\hat{n}_e$) y la orientación deseada ($\hat{e}_i$.

Figura 3.1: Ilustración obtenida de [16]

Así pues, una vez obtenida la fuerza total ejercida sobre una determinada partícula i ($\overrightarrow{F}_i$), junto con su torque de giro ($\overrightarrow{\tau_i}$), integramos Ec. 3.1 para obtener la trayectoria, velocidad y aceleración de la partícula para cada instante de tiempo.

Una particularidad que se ha considerado en la implementación de este modelo sobre las coordenadas angulares, es que se han modelizado siguiendo el formalismo de cuaterniones [17]. Este método no utiliza la formulación tradicional de ángulos de Euler, sino que a cada partícula se le asigna inicialmente un cuaternión unitario de la forma $q = (q_0, q_1, q_2, q_3)$, cumpliéndose que $q^2 = 1$. Este cuaternión, da cuenta de la orientación de la partícula en cada momento y se utiliza como referencia para ir actualizando las diferentes posiciones angulares al deber cumplirse la ecuación de movimiento:

$$\dot{q} = \frac{1}{2} Q(q)\omega \tag{3.7}$$

en donde

$$\dot{q} = \begin{pmatrix} \dot{q_0} \\ \dot{q_1} \\ \dot{q_2} \\ \dot{q_3} \end{pmatrix} \qquad Q(q) = \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix}, \qquad \omega = \begin{pmatrix} 0 \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{3.8}$$

La complejidad del sistema se reduce notablemente al trabajar en **espacios bidimensionales**, en donde no existe ni velocidades angulares en x e y, $\omega_x = \omega_y = 0$, ni torques actuando en las mismas direcciones, por lo que $\sum_{j=1}^{N_c} \tau_{ij}^x = \sum_{j=1}^{N_c} \tau_{ij}^y = 0$. Además se trabaja con partículas homogéneas, por lo que $I_x = I_y = I_z$. Bajo este esquema, las ecuaciones 3.4, 3.5 y 3.6 que estaban impuestas sobre tres dimensiones, pueden reescribirse de forma escalar:

$$\tau_{G_i}^z = \sum_{j}^{N_c} \overrightarrow{r}_{ij} \times \overrightarrow{F}_{ij} \tag{3.9}$$

$$\tau_D = -S_D \Delta\theta - \beta\dot{\theta} + R(t) \tag{3.10}$$

$$\tau_{D_i} + \tau_{G_i} = I_i \cdot \omega_i^z \tag{3.11}$$

La integración de las componentes rotacionales del movimiento (Ec. 3.7 o Ec. 3.11) se ha realizado mediante un método denominado algoritmo de *Finchanm's Leap-Frog* [18]. El desarrollo numérico empleado para su implementación se comenta en el Apéndice A.1.

Con todos estos ingredientes, simularemos un proceso de evacuación con un total de 192 personas que intentarán escapar a lo largo de una puerta de 70 cm de ancho. La habitación será cuadrada de 8m x 8m. La masa de los peatones será de 67 kg siguiendo una distribución normal truncada entre [45-114 Kg]. Los peatones serán definidos como esferocilindros (pasará a detallarse la evolución en su modelización en la sección 3.2), con un eje largo uniformemente distribuido entre [0.35-0.5 m] y un eje corto entre [0.24-0.33 m]. Estos valores se corresponden con un experimento real llevado a cabo por Garcimartín et al [14].

Fuerza de interacción entre las partículas

Como último punto y concluyendo la descripción del modelo DEM, pasamos a detallar las expresiones que dan cuenta de la interacción entre dos partículas. Antes de nada, comentar que esta interacción es binaria; esto es, a efectos de programa los choques son dos a dos, no permitiéndose que choquen más de dos partículas a la vez. En caso de que existan partículas que interactúen con más de un vecino, los choques se realizarán por separado.

Así pues, a la hora de evaluar la interacción de una partícula sobre sus vecinos, esta se computará como suma de dos términos. Un término correspondiente a la dirección normal de ambos vecinos y un segundo término en la dirección tangencial. La definición de ambos vectores $(\hat{n}, \hat{t})$ ya se ha detallado en la Fig. 3.1(a). Matemáticamente expresamos esta interpretación de $\overrightarrow{F}_{ij}$ como:

$$\overrightarrow{F}_{ij} = F_{ij}^n \cdot \hat{n} + F_{ij}^t \cdot \hat{t} \tag{3.12}$$

En donde F_{ij}^n y F_{ij}^t son las componentes normal y tangencial respectivamente de la fuerza

de interacción.

La expresión dada para la fuerza normal está compuesta a su vez de dos términos: el primero de ellos es un aporte elástico que impide el solapamiento de peatones, mientras que el segundo es un término disipativo y proporcional a la velocidad relativa entre las partículas que chocan:

$$F_{ij}^n = -k^n \delta - \gamma^n v_{rel}^n \tag{3.13}$$

En donde k^n y γ^n son la constante elástica y coeficiente de restitución del sistema en la dirección normal, δ es la distancia de solapamiento (dependiendo de la caracterización de la partícula su cálculo será diferente) y v_{rel}^n es la velocidad relativa normal entre ambas partículas en contacto.

La fuerza tangencial también contiene un término elástico y un segundo término de fricción que permite recoger el rozamiento entre partículas:

$$F_{ij}^t = -k^t \xi - \gamma^t v_{rel}^t \tag{3.14}$$

En donde γ^t y v_{rel}^t vuelven a tener el mismo significado que en la componente normal pero esta vez proyectados sobre la dirección tangencial. En cuanto a $\xi(t)$ representa la elongación de un muelle imaginario con constante elástica k^t, e incrementa como $\frac{d\xi(t)}{dt} = v_{rel}^t$ originando así la existencia de un solapamiento entre las partículas. Dicho valor se obtiene mediante la integración con un algoritmo de tipo Euler. Además, el término tangencial F^t está truncado para satisfacer las restricciones de Coulomb[1], debiéndose cumplir que $|\overrightarrow{F}_{ij}^t| < \mu|\overrightarrow{F}_{ij}^n|$.

Implementación del modelo sobre GPU

Una de las problemáticas más grandes que encontramos asociada a la implementación de algoritmos de este tipo, son los altos costes de computación en términos de tiempo de ejecución. Esto es debido a la integración de ecuaciones de movimiento para el sistema de partículas. Por esta razón, resulta novedoso presentar algoritmos DEM implementados y optimizados para su ejecución en paralelo sobre GPU [12]. Gracias al trabajo realizado, nos hemos podido permitir la simulación de sistemas con concentraciones de partículas más o menos altas, pero obteniendo tiempos de ejecución muy cortos en comparación a los que hubiéramos obtenido en caso de implementar un programa secuencial sobre CPU. De esta forma, en esta sección pasamos a detallar los rasgos más característicos a la hora de implementar el algoritmo DEM sobre GPU.

Como primer punto, queremos presentar un diagrama de flujo (Fig. 3.2) del método utilizado, en donde los procesos implementados sobre CPU se denotan con color Rojo y trazos continuos, mientras que los utilizados sobre GPU se detallan en color amarillo y sobre trazos discontinuos. El lenguaje utilizado para el desarrollo de todos los procedimientos, como ya se comentó en la

[1]El modelo de Coulomb propone que la fricción trata de minimizar la velocidad tangencial relativa, pero la fuerza aplicada no puede ser mayor que una fracción de la fuerza normal.

sección 2.1, es C/C++ utilizando la extensión CUDA ofrecida por NVIDIA [10].

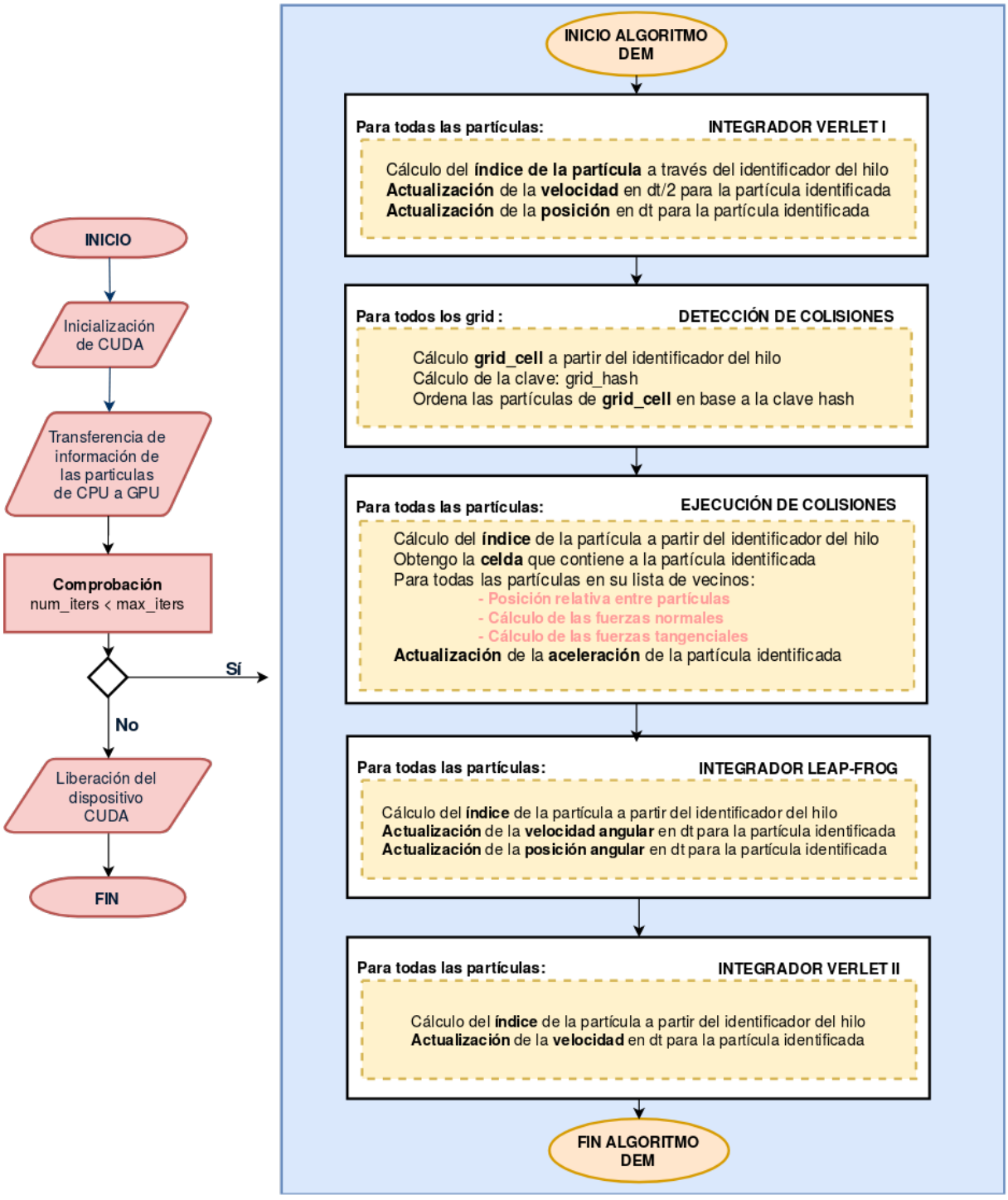


Figura 3.2: Diagrama de flujo del algoritmo DEM implementado. Aquellos procesos que se explican bajo cajas cuyos bordes son trazos discontinuos son las que se ejecutan en paralelo sobre GPU.

El programa comienza inicializando el dispositivo CUDA, permitiéndonos así acceder a las utilizades propias de la GPU. Dada la necesidad de disponer de declaraciones dobles en variables y procesos, tras la inicialización del dispositivo se procede a realizar la transferencia de la información recogida en la CPU hacia la GPU. De esta forma y tras la finalización de estos dos pasos, el sistema está completamente inicializado y listo para comenzar a evolucionar.

Siguiendo el diagrama de flujo de Fig. 3.2, el siguiente paso da comienzo al bucle temporal que da cuenta de la evolución del sistema. Así pues, las ecuaciones del movimiento traslacional ya se ha comentado que se integran mediante un *algoritmo de Verlet* de dos etapas. El proceso consiste en, al principio de la iteración, toma las aceleraciones obtenidas en la iteración anterior y actualiza las velocidades y posiciones. La primera de ellas hasta la mitad de la presente iteración y la segunda una iteración completa. Finalmente y antes de concluir, reactualiza la velocidad de la partícula para el final del intervalo.

En términos de GPU, el proceso se ha llevado a cabo mediante la utilización de las funciones de la librería *Thrust* [11]. Lo que se nos permite con esta librería es trabajar desde la CPU originando para cada partícula una estuctura individual en la que se guardan su posición, velocidad y aceleración. Todo el conjunto de estructuras posteriormente se pasan a la GPU, y esta, a través de un iterador de Thrust, realiza las integraciones Ec. 3.1 asignando una estructura (o partícula) a los diferentes hilos de la GPU.

Una de los puntos positivos del uso de la librería *Thrust* es que la asignación del número de hilos y bloques están realmente optimizados: en tiempo de ejecución, dependiendo del número de estructuras originadas y de la complejidad de la función integradora, se decide el número de bloques e hilos a utilizar. El punto negativo viene de que el tamaño total de las estructuras es bastante reestrictivo, limitando el tipo de funciones que se permiten paralelizar. Es por esta razón por la que nos hemos visto limitados en la implementación de otro tipo de algoritmos integradores más complicados como es el caso de *Runge-Kutta*.

Otro de los aspectos importantes a tener en cuenta es la detección de colisiones. No todas las partículas son susceptibles a tener una colisión; esto es, compartir un espacio común con otra partícula. Para que el algoritmo sea realista, este debe localizar aquellas partículas que puedan producir colisiones antes de que se produzcan, permitiendo así realizar una integración correcta del sistema. Uno de los métodos más extendidos en este tipo de situaciones es la creación de una matriz de vecinos en donde se recorre la mitad del sistema de partículas y se ve cuáles cumplen ciertas reestricciones de distancias para considerar una pareja como vecinos. Seguir este tipo de técnicas en sistemas con muchas partículas puede ralentizar mucho la ejecución, por lo que en nuestro programa se propone la creación de una lista de vecinos que se diseña a partir de un método de *link cell* ofrecido por el *ToolKit* de Nvidia. Dado que las posiciones de cada partícula se actualizan en cada iteración, la lista debe ser actualizada continuamente, lo que implica que

su correcta implementación involucra tiempos de ejecución menores.

Una vez detectadas las partículas susceptibles de sufrir alguna colisión, el diagrama Fig. 3.2 realiza la ejecución de colisiones. Para ello, recorre la lista de vecinos de cada partícula, calculando la fuerza y el torque que cada uno de los posibles contactos ejerce sobre la partícula en cuestión. Al terminar de recorrer la lista, se procede a la actualización de las aceleraciones lineales y angulares para posteriormente utilizarlas en los algoritmos integradores correspondientes. De esta forma, acaba el algoritmo en una iteración y vuelve a comprobarse si el número de iteraciones ha sobrepasado el límite establecido. En caso negativo, se repite de nuevo todo el proceso.

Finalmente, cabe comentar la caracterización propia del sistema de partículas; esto es, la determinación del número de celdas en las que será dividido el espacio de cálculo, así como la relación de volúmenes que existe entre el tamaño de la celda y el de la partícula. Así pues y con el fin de optimizar las tareas asignadas a cada *warp* (conjunto de 32 hilos consecutivos), el número de celdas generalmente se escoge como una potencia de 2. Además, con el fin de minimizar el coste computacional en la realización de la detección de vecinos y la ejecución de las colisiones, el volumen de las partículas es siempre inferior al de las celdas, implicando así que el número de partículas en el sistema sea un múltiplo del tamaño del *warp*.

Caracterización de la forma de las partículas

Uno de los aspectos más importantes en la simulación de peatones es la geometría de estos, dado que es a partir de dicha geometría sobre la que se definen las fuerzas de interacción del sistema.

En esta sección veremos la evolución que se ha llevado a cabo para la caracterización de los peatones en un espacio bidimensional. Veremos cómo se ha realizado el cálculo de la distancia de solapamiento δ para casos sencillos (esferas) y comentaremos como esto nos ha servido para extrapolar los resultados obtenidos a geometrías más complicadas.

Partículas esféricas

Las partículas esféricas son las más sencillas de modelar. Dadas dos esferas de las que conocemos su radio (R_1 y R_2), así como la distancias de sus centros de masas ($\overrightarrow{r_1}$ y $\overrightarrow{r_2}$) (Fig. 3.3), el cálculo de la distancia de solapamiento se obtiene trivialmente mediante:

$$\delta = (R_1 + R_2) - |\overrightarrow{r_1} - \overrightarrow{r_2}| \tag{3.15}$$

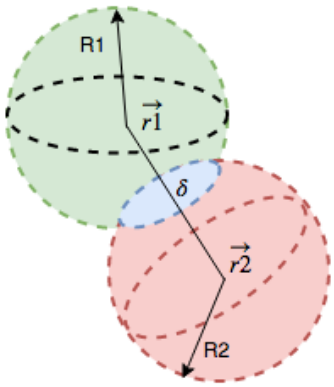


Figura 3.3: Representación del contacto entre dos esferas conocidos los radios y posiciones del centro de masas de ambas. El solapamiento queda representado en azul.

De esta forma, dos partículas únicamente podrán considerarse que están en contacto siempre y cuando las distancias entre sus centros sea menor que la suma de sus radios.

Segmentos lineales

Una segunda aproximación más realista al problema es considerar segmentos lineales con una longitud característica L_i y grosor diferencial.

Evaluar el solapamiento en este tipo de partículas carece de sentido, dado que al tratarse de cuerpos sin grosor, no existe un solapamiento *per se*. Sin embargo, hemos considerado de interés esta configuración, dado que el estudio de los posibles puntos de contacto entre dos partícula lineales es un ingrediente necesario para evaluar el solapamiento de partículas con geometrías más complejas.

A todo ello, además se debe añadir que, al trabajar en espacio bidimensionales, el punto de contacto entre dos segmentos lineales será, en al menos una de las partículas, alguno de los dos vértices que las conforman. En caso tridimensionales, sí que existe la posibilidad de choques en posiciones no extremas, pero en el caso que nos ocupa, este problema se simplifica.

Así pues, supuestas dos partículas como en la Fig. 3.4, debemos estudiar cuáles son los puntos de contacto de la partícula 1 sobre la 2. Para ello, nos bastará conocer la distancia mínima entre un punto y una recta.

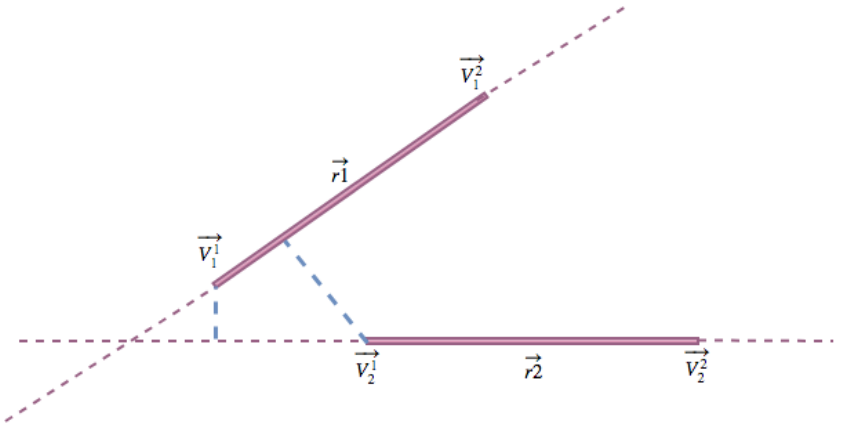
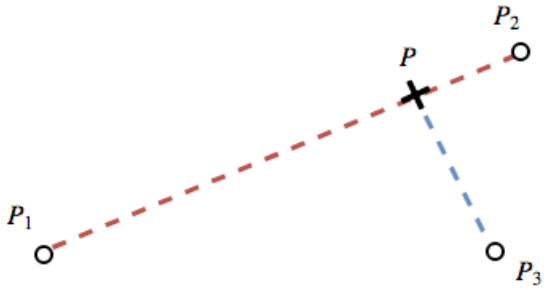


Figura 3.4: Representación del contacto entre dos partículas lineales. La notación de los vértice V_i^j se entiende como el vertice i asociado a la partícula j.

La técnica que se utiliza para encontrar la menor distancia entre un punto y una recta consiste en definir la expresión vectorial de todos los puntos pertenecientes a la recta cuyo vector director es el segmento que une los vértices de la partícula 2. Así pues tendremos que:

$$\overrightarrow{P} = \overrightarrow{V_2^1} + u(\overrightarrow{V_2^2} - \overrightarrow{V_2^1}) \tag{3.16}$$



Dado un punto externo P_3 (en nuestro caso serán o bien $\overrightarrow{V_1^1}$ o $\overrightarrow{V_1^2}$), debemos encontrar el punto P que cumpla:

$$(\overrightarrow{V_1^i} - \overrightarrow{P}) \cdot (\overrightarrow{V_2^2} - \overrightarrow{V_2^1}) = 0 \tag{3.17}$$

Sustituyendo la expresión de P en esta última ecuación, llegamos a:

$$[\overrightarrow{V_1^i} - \overrightarrow{V_2^1} - u((\overrightarrow{V_2^2} - \overrightarrow{V_2^1})] \cdot (\overrightarrow{V_2^2} - \overrightarrow{V_2^1}) = 0 \tag{3.18}$$

De donde se puede calcular un valor de u:

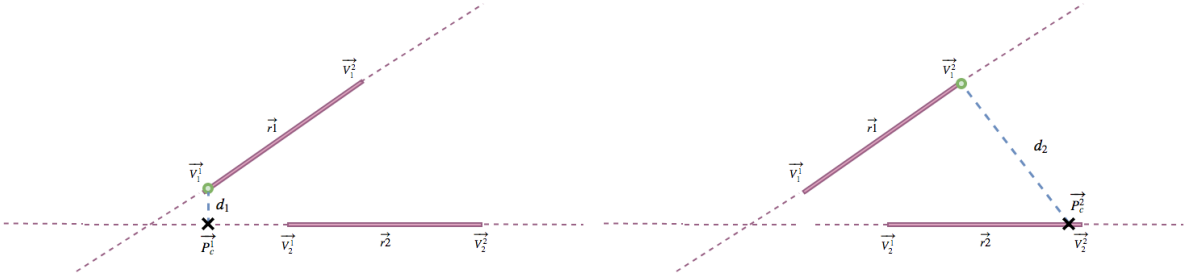
$$u = \frac{(\overrightarrow{V_1^i} - \overrightarrow{V_2^1}) \cdot (\overrightarrow{V_2^2} - \overrightarrow{V_2^1})}{||\overrightarrow{V_2^2} - \overrightarrow{V_2^1}||^2} \tag{3.19}$$

Una vez obtenida la expresión de u para cada uno de los dos vértices, obtendremos dos

puntos de contacto sobre la recta (en la Fig. 3.5 se denotan como P_c^1 y P_c^2). Aquel vector que une dichos puntos con sus vértices correspondientes (d_1 y d_2) y que tenga un módulo menor, nos dará el punto más cercano de 1 a la recta 2.

Finalmente, conocido el punto más cercano, su valor de u y el punto de contacto obtenido, puede ocurrir:

1. El punto de contacto se encuentra dentro de los límites de la partícula. En este caso no es necesario realizar ninguna modificación.
2. El punto de contacto se encuentra a la izquierda del vértice 1 de la partícula 2. Esta situación se corresponde con valores de u negativos. Para este tipo de casos, el punto de contacto será el propio vértice 1.
3. El punto de contacto se encuentra a la derecha del vértice 2 de la partícula 2. Esta situación se corresponde con valores de u mayores que 1. La solución que ofrecemos es considerar el vértice 2 como punto de contacto.



(a) Distancia d_1 del vértice 1 de la partícula 1 a la recta definida por la partícula 2.

(b) Distancia d_2 del vértice 2 de la partícula 1 a la recta definida por la partícula 2.

Figura 3.5

El caso que se explica gráficamente en la Fig. 3.5 se corresponde con la segunda de las opciones comentadas. El vértice más cercano de 1 a la recta originada por la partícula 2 es el vértice 1. Su punto de contacto correspondiente con la recta queda fuera de los límites de la propia partícula 2 por lo que el punto de contacto de 1 sobre 2 será el vértice 1 de la segunda partícula.

En términos de evolución del algoritmo, el procedimiento que se sigue consiste en ir seleccionando todo el conjunto de partículas que conforman el sistema. Para cada partícula, conocer cuáles son sus vecinos con su lista de vecinos asociada. Finalmente, aplicar el procedimiento comentado a todos lo vecinos. Al terminar con todos los vecinos, pasar a la partícula siguiente.

Peatones

Finalmente, la forma elegida para la modelización de los peatones ha sido la esferocilíndrica. Se ha considerado que la proyección de un peatón sobre un espacio bidimensional es muy similar

a la de un esferocilindro. En términos geométricos, un esferocilindro se obtiene recorriendo un segmento de extremos conocidos con una esfera de radio constante.

Esta caracterización resulta muy novedosa, dado que en la mayoría de casos vistos en la bibliografía [12], se trabajaban con peatones esféricos.

Así, un esferocilindro cualquiera i, queda definido por el segmento de extremos V_i^1 y V_i^2, y la esfera de radio fijo R_i. Una vez establecidos estos parámetros, dos partículas estarán en contacto siempre que haya solapamiento entre las dos esferas virtuales que generan sus volúmenes en el punto de contacto. Siguiente el procedimiento de búsqueda de puntos de contacto entre dos segmentos lineales (Sec. 3.2.2), podemos determinar la posición de los centros de dichas esferas y calcular sus fuerzas de contacto. Una representación gráfica del procedimiento a seguir se detalla en la Fig. 3.6.

Una vez determinado el centro de las esferas virtuales que generan el contacto, calcular el solapamiento δ entre las dos partículas, se reduce simplemente al caso esférico. Así pues, el valor de δ que será sustituido en Ec. 3.13 será:

$$\delta = R_i + R_j - |\overrightarrow{C}_i - \overrightarrow{C}_j| \tag{3.20}$$

Donde R_i,R_j y C_i, C_j se corresponden con el radio y el centro de las esferas virtuales en contacto de cada uno de los esferocilíndros.

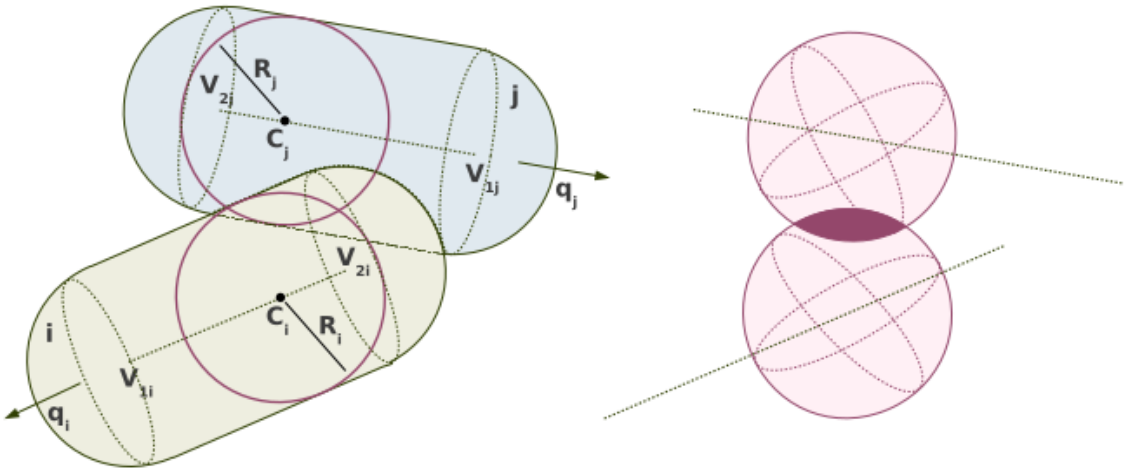


Figura 3.6: A la izquierda, representación de dos esferocilindros en contacto. En el esquema de la derecha se muestra la situación que hay que computar realmente para determinar la fuerza de interacción entre ambas partículas.

El punto de aplicación de la fuerza de interacción para el caso de partículas rígidas será el centro de masas del esferocilindro, que coincide con el punto medio del eje mayor que lo conforma. En cuanto al punto de aplicación del torque, deberemos considerar que no es el punto de contacto calculado sobre el eje, sino que se debe añadir el radio correspondiente a la esfera virtual. Matemáticamente, el vector que lo define será:

$$\overrightarrow{C}_i^{\Gamma} = \overrightarrow{C}_i + \frac{\overrightarrow{C}_j - \overrightarrow{C}_i}{2} \tag{3.21}$$

Finalmente, el valor de $\overrightarrow{r}_{ij}$ en Ec. 3.9, no será mas que el vector que une el centro de masas del esferocilindro con el punto de contacto dado por el vector $\overrightarrow{C}_i^{\Gamma}$.

Caracterización de las condiciones de contorno

Como último punto en la caracterización del modelo y tras determinar el método de interacción entre partículas, pasamos a describir las condiciones de contorno elegidas.

Condiciones de contorno periódicas

Las condiciones de contorno periódicas (PBC) suelen utilizarse en sistemas grandes pretendiendo minimizar el efecto de los bordes sobre el comportamiento global de las partículas. A efectos de interpretación, suponer PBC consiste en considerar que nuestro sistema está rodeado en todas las direcciones por copias idénticas de si mismo. Así pues, cuando una partícula abandona el sistema por un determinado lugar, entra de nuevo en él por el punto homólogo al de salida pero en el extremo contrario.

Imponer PBC en una simulación numérica supone tener en cuenta ciertas restricciones adicionales sobre el sistema. Una de las más importante surge a raíz de las interacciones que tiene una partícula con el resto. Supongamos que un individuo i está cerca de la frontera en una determina dirección. Debemos considerar que, además de interactuar con todas las partículas j que se encuentren próximas a él dentro de la superficie principal, se debe añadir también el contacto con partículas próximas a la frontera en las direcciones opuestas. Además puede ocurrir que una partícula esté próxima a varias fronteras al mismo tiempo, pudiendo estar en contacto con una partícula j en varias de esas copias del sistema. Así pues, realizar una buena implementación PBC nos garantiza que en todo momento, la partícula i que abandona la región principal efectúe el contacto (en caso de que exista) con la copia de la partícula j más próxima a ella.

Interacción de peatones con las paredes del recinto de confinamiento

En sistemas de peatones, uno de los puntos más importantes es la caracterización del choque de las partículas con la pared (recordar término $\overrightarrow{F_{\omega_i}}$ en Ec. 3.1). En nuestro sistema nos hemos decidido por definir las paredes como un conjunto de partículas con las mismas características que los individuos que conforman el sistema. De esta forma, evitamos definir nuevas

reglas de interacción para este caso particular de choque al ya haber definido el contacto entre dos partículas previamente (Sec 3.1.2). El único punto adicional que queda por determinar es el centro de la partícula perteneciente a la pared que interactúa con la partícula que la golpea.

Para dar respuesta a esta pregunta, los procedimientos a seguir son varios. En principio se podría pensar que la pared está constituido por un conjunto finito de partículas que ocupan posiciones fijas (Fig. 3.7 - izda -). El problema de este tipo de configuración reside en que se origina una pared rugosa (presenta ciertos huecos), lo que en ciertas ocasiones puede resultar en implementaciones no aceptables.

La segunda forma consiste en implementar paredes con un número infinito de partículas. Cuando un peatón se acerca a la pared, se determina la distancia mínima de este a la frontera, simulando la interacción con una partícula de sus mismas características. (Fig. 3.7 - dcha -). El problema de encontrar cuál es el punto más cercano de la pared con el peatón ya ha quedado resuelto dado que coincide con el problema de buscar la distancia mínima entre un punto y una recta (recordar que nos encontramos en el caso bidimensional).

A efectos de programa, el proceso de implementación se describe tal y como se indica en Fig. 3.8. De esta forma, cada partícula determina su posición relativa con respecto a la pared comprobando si está o no en contacto con esta. En caso afirmativo, calcula la fuerza de interacción ejercida siguiendo el mismo procedimiento que el ya comentado previamente. Este procedimiento debe realizarse de forma inmediatamente posterior a la ejecución de colisiones presente en Fig.3.2.

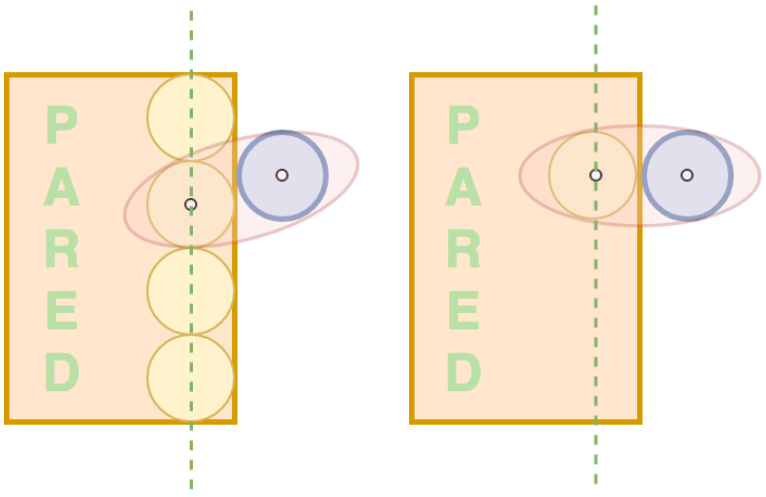


Figura 3.7: A la **izquierda**, contacto de una partícula con una pared formada por a unidades finitas de esferas en posiciones fijas. A la **derecha**, esferas situadas a la distancia mínima con la partícula que produce el choque.

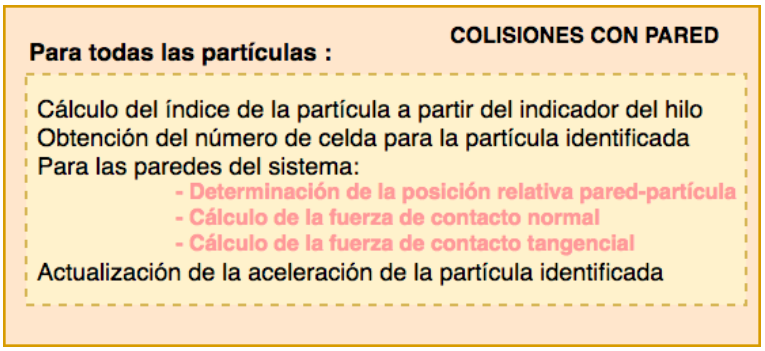


Figura 3.8: Algoritmo de detección de colisión del individuo con la pared del recinto.

Capítulo 4

Resultados del modelo

En este último capítulo, presentaremos y discutiremos los resultados obtenidos tras la implementación del modelo descrito en el capítulo anterior. Contrastaremos si, efectivamente, este recoge el comportamiento observado experimentalmente en los procesos de evacuación en [14, 13] permitiéndonos así corroborar la validez de la implementación numérica aquí realizada.

Adicionalmente, añadiremos un punto de complejidad adicional al sistema, añadiendo un obstáculo cerca de la puerta de salida y analizando cómo se modifican las magnitudes presentadas en el modelo inicial conforme modificamos la distancia entre la salida y el obstáculo. Esta modificación se introduce basándonos en los diferentes estudios llevados a cabo [19, 20, 21, 22], en donde en un principio se supone que el añadir un obstáculo al sistema ayuda a descongestionarlo.

Modelo sin obstáculo

Estudio del flujo de peatones a través de la puerta

Comenzaremos determinando la magnitud macroscópica más relevante del sistema; esto es, el flujo medio de personas que atraviesan la puerta por segundo. Con ello, pretenderemos determinar qué parámetros favorecen la evacuación y representan de manera más realista los valores de flujo obtenidos experimentalmente en [14, 13].

Así pues, se ha analizado el flujo del sistema para distintos valores de η (recordar que η representa la amplitud del ruido $R(t)$ en Ec. 3.10).Se ha ejecutado un estudio sistemático, aumentando dicho parámetro en intervalos de 0.125 desde una amplitud de 0 hasta una amplitud de 0.5. Esta simulación se ha realizado dos veces, la primera de ella con un valor de $S_D = 20Nm$ y la segunda con una rigidez de $S_D = 15Nm$.

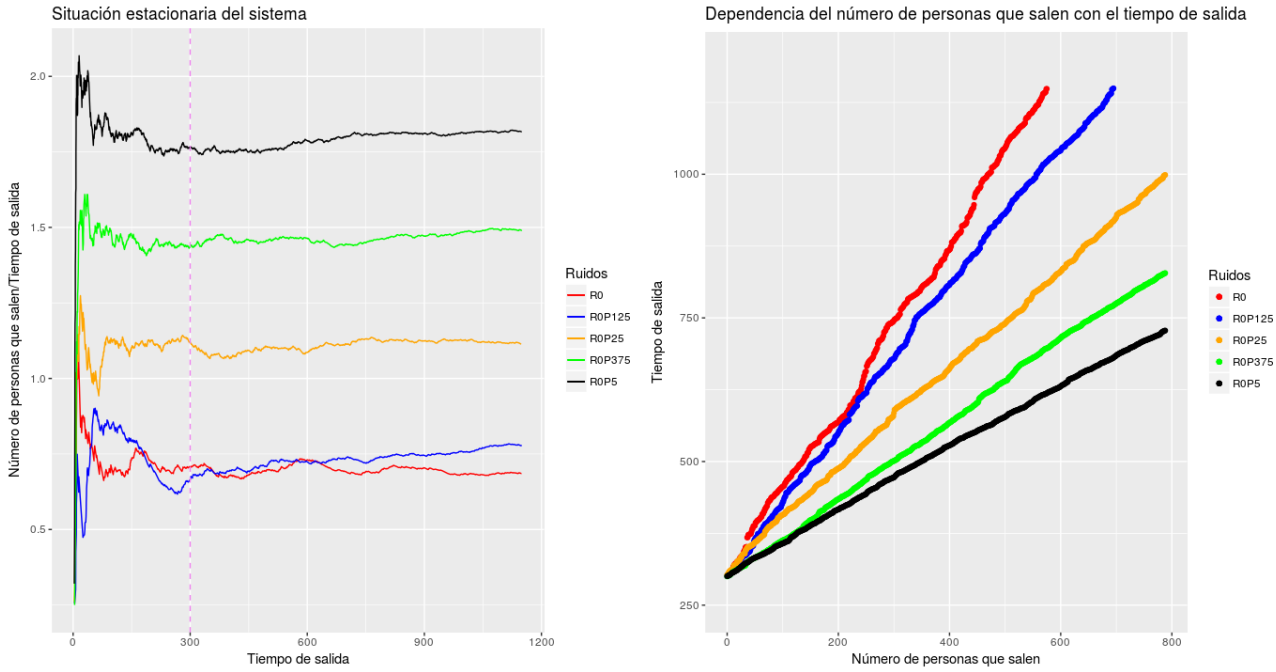


Figura 4.1: La línea punteada marca el alcance del régimen estacionario para el sistema (izda). Dependencia del número de personas que abandonan la habitación con el tiempo en el que lo hacen (dcha).

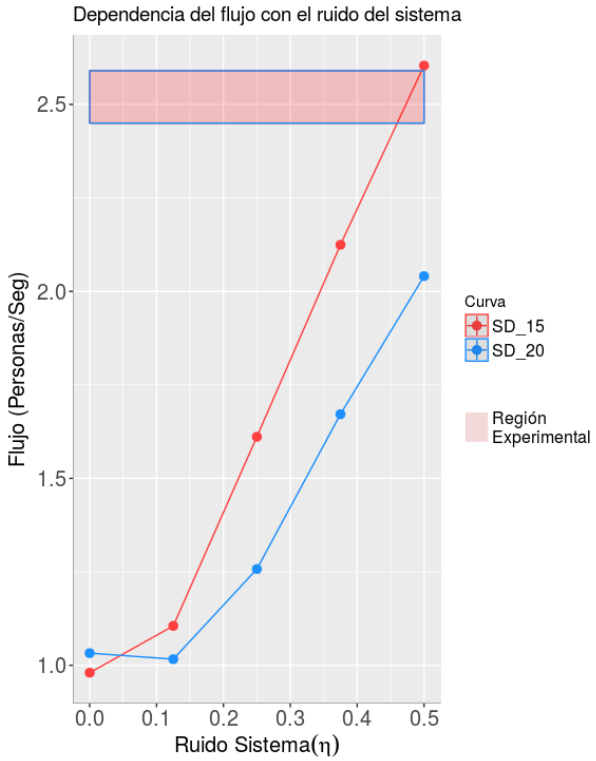


Figura 4.2: Dependencia del flujo con el ruido del sistema para distintos valores de S_D. En naranja, valores de flujo obtenidos experimentalmente en [13].

El cálculo del flujo se ha determinado una vez el sistema ha alcanzado el estado estacionario. Evitando los primeros segundos de simulación, se ha contabilizado el número de personas que salen así como el tiempo en el que lo hacen. La dependencia que presenta este comportamiento en condiciones estacionarias es aproximadamente lineal, independientemente de la elección de parámetros (Fig. 4.1).

Con estas consideraciones, el cálculo del flujo se ha realizado mediante una regresión lineal, obteniendo de esta forma el valor de la pendiente y su error asociado. Así, el valor de la pendiente resultará ser valor de flujo en [personas/Seg].. Los resultados obtenidos se detallan en Fig. 4.2.

Se podrían considerar formas alternativas a la hora de obtener el valor de flujo. Una buena solución sería realizar varias repeticiones de la simulación y promediar los flujos asociados a cada valor de ruido. Seguir este procedimiento supondría obtener valores con mayor validez estadística, pero el coste computacional que implicaría su implementación sería alto. De esta forma, y observando que los valores de flujo obtenidos son coherentes con los resultados experimentales (que oscilan entre 2.45 y 2.6 personas/seg), consideramos que el método elegido en la determinación del flujo del sistema ha sido correcta.

Evaluando ahora el comportamiento que se muestra en Fig. 4.2, observamos en primer lugar que conforme aumenta el valor de η (amplitud de ruido), el sistema tiende a valores de flujo mayores; esto es, se favorece la evacuación. Este comportamiento es coherente con la propia definición que habíamos dado de la amplitud de ruido. Conforme los peatones tienen una mayor capacidad de cambiar su dirección de forma aleatoria en situaciones de atasco, el sistema tiende a descongestionarse más fácilmente y pasar así de estados concentrados a estados más diluidos.

El mismo comportamiento puede observarse conforme disminuye el valor de S_D (rigidez angular). Siguiendo el mismo razonamiento que el ofrecido para el caso de η, resulta coherente pensar que conforme los peatones se muestren menos reacios a cambiar su orientación deseada (valores de S_D cada vez más bajos), el flujo se ve favorecido. Dada una situación de alta compactibilidad, el permitir que los peatones adquieran un mayor abanico de orientaciones, supone una mayor probabilidad de diluir el sistema. Un sistema más diluido implica mayores valores de flujo para las partículas que lo constituyen.

Distribución de orientaciones cerca de la salida

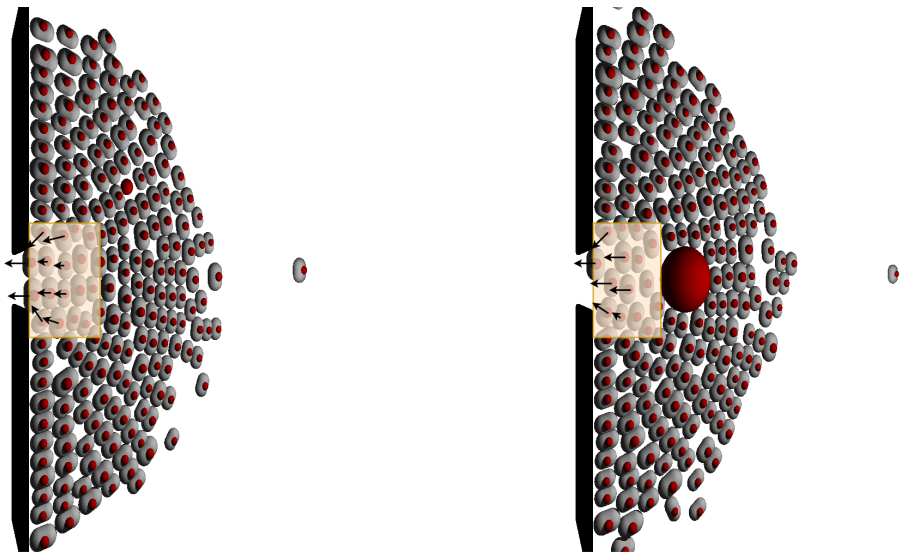
Uno de los puntos más interesantes que obtenemos con la implementación de este tipo de simulaciones numéricas, es la capacidad de análisis de ciertas magnitudes que experimentalmente son difíciles de calcular. De hecho, la mayoría de resultados experimentales presentados a día de hoy se centran principalmente en determinar magnitudes macroscópicas (valores de flujo), mientras que las propiedades micromecánicas del sistema, quedan apartadas dado su alta complejidad de medida. Así pues, y visto que el modelo propuesto responde coherentemente a

las magnitudes macroscópicas experimentalmente medidas, pasamos a intentar comprender qué ocurre a nivel microscópico en el sistema para que se den este tipo de comportamientos.

Analizaremos la distribución que sigue las orientaciones de los peatones en posiciones cercanas a la puerta de salida. Este tipo de estudios nos puede orientar a la hora de saber si la elección de ciertos parámetros originan configuraciones geométricas que favorecen la salida de la habitación.

Una vez determinada las posiciones de los peatones a lo largo de la simulación y alcanzado el estado estacionario, hemos seleccionado una región de estudio cercana a la puerta. Sobre aquellos peatones que se encontrasen en dicha región, hemos calculado su orientación y la proyección de esta sobre la dirección deseada o vector $\hat{e}$ (recordar su definición en la sección 3.1.1). Las definiciones del vector dirección y vector target para cada uno de los peatones se han tomado siguiendo el criterio comentado en Fig. 3.1. Un ejemplo de la región de estudio y orientaciones de los peatones dentro de ella puede verse en la Fig. 4.3.

Finalmente se ha realizado el histograma de las diferentes orientaciones tomadas por los peatones para distintos valores de ruido: $\eta = (0, 0.125, 0.25, 0.375, 0.5)$ y para distintos valores de rigidez angular: $S_D = (15, 20)$. Los resultados obtenidos pueden observarse en Fig. 4.4, Fig. 4.5 (para valores de $S_D = 20$) y Fig. 4.6, Fig. 4.7 (para valores de $S_D = 15$). Las gráficas correspondientes a valores de η =0.125 y η =0,375 para sendos valores de rigidez angular, se encuentran en el apéndice B.1.



(a) Orientaciones de los peatones en el modelo sin obstáculo dentro de la zona de estudio.

(b) Orientaciones de los peatones en el modelo con obstáculo dentro de la zona de estudio.

Figura 4.3: En color naranja, definición de la región de estudio cercana a la puerta donde se determinará la distribución de las orientaciones.

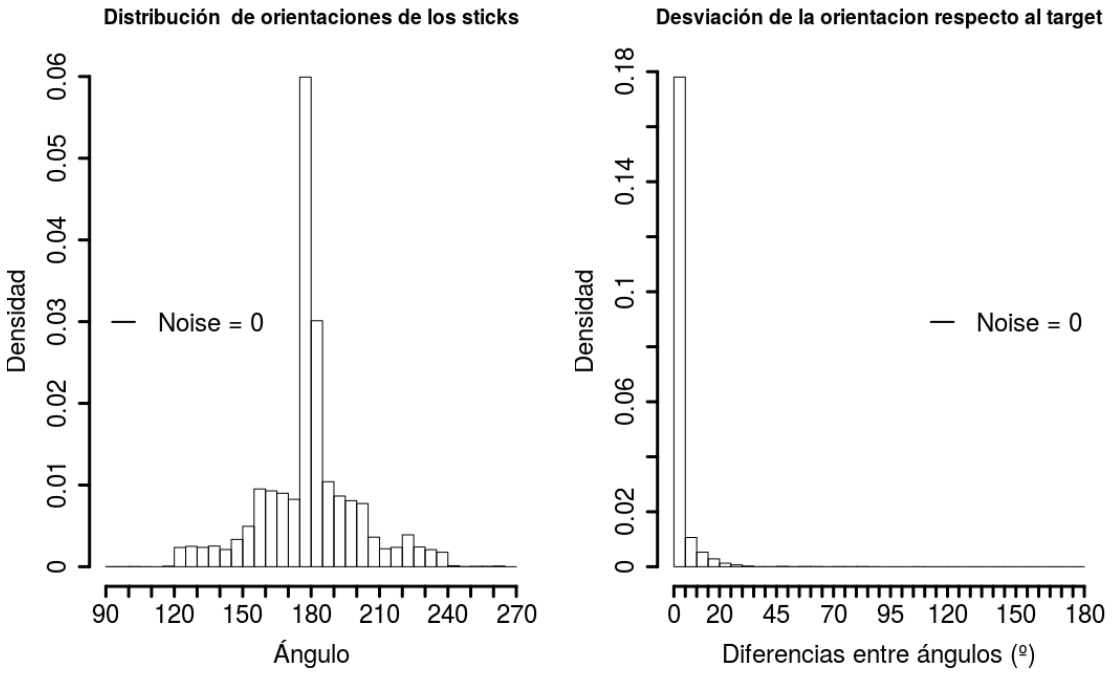
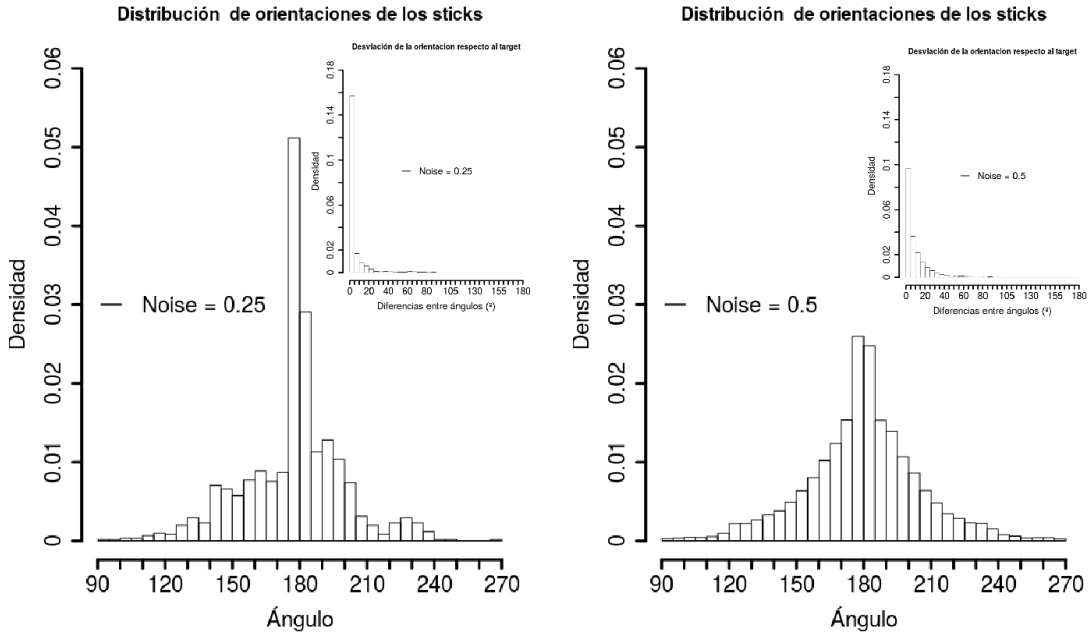


Figura 4.4: Distribución de las orientaciones de los peatones cercanos a la puerta para un valor de $\eta = \mathbf{0}$ y $\mathbf{S_D = 20}$ (Izda). Ángulo de desviación del peatón con respecto a su orientación deseada (Dcha).



(a) $\eta = 0{,}25$ y $S_D = 20$.

(b) $\eta = 0{,}5$ $S_D = 20$.

Figura 4.5: Distribución de las orientaciones de los peatones cercanos a la puerta para distintos valores de ruido. En pequeño, ángulo de desviación del peatón con respecto a $\hat{e}$ (*target* u orientación deseada).

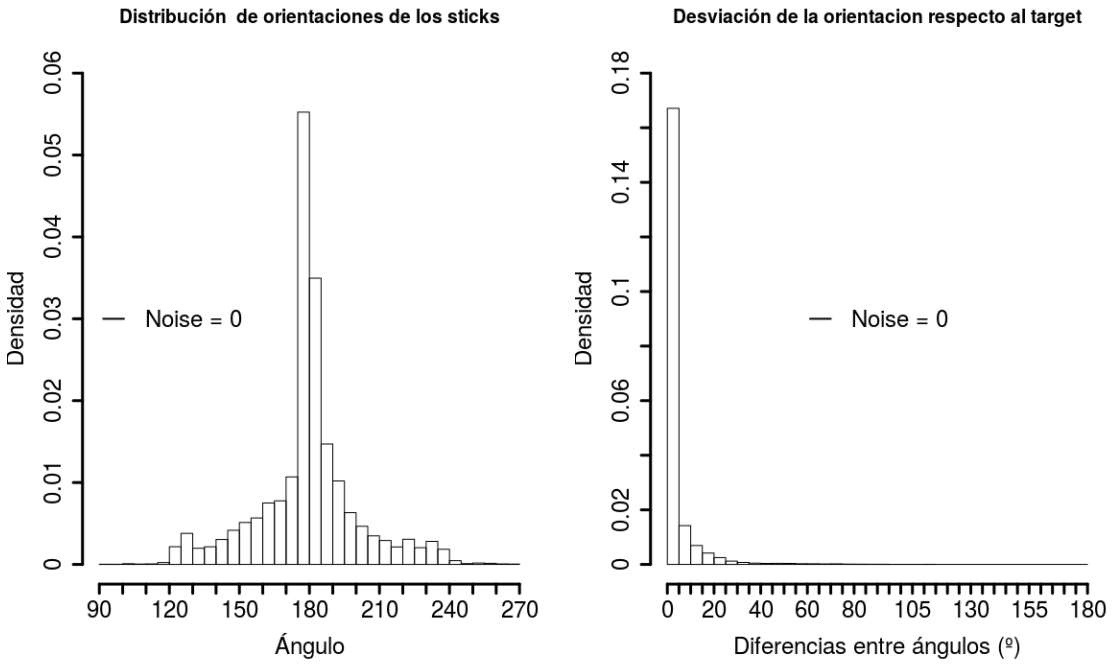
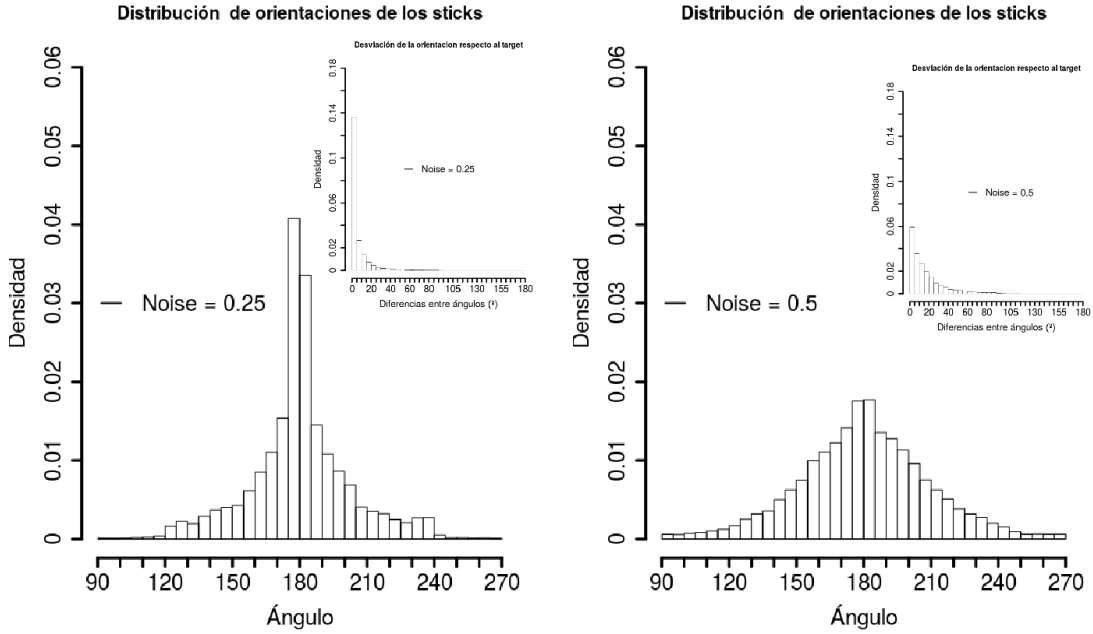


Figura 4.6: Distribución de las orientaciones de los peatones cercanos a la puerta para un valor de $\eta = \mathbf{0}$ y $\mathbf{S_D = 15}$ (Izda). Ángulo de desviación del peatón con respecto a su orientación deseada.



(a) $\eta = 0{,}25$ $S_D = 15$.

(b) $\eta = 0{,}5$ $S_D = 15$.

Figura 4.7: Distribución de las orientaciones de los peatones cercanos a la puerta para distintos valores de ruido. En pequeño, ángulo de desviación del peatón con respecto a $\hat{e}$ (*target* u orientación deseada).

El comportamiento que puede observarse para los cuatro valores de ruido que hemos presentado es que el sistema tiende a orientar a los peatones en su forma natural (media del histograma en 180°). Además, esta orientación real del peatón coincide con su orientación deseada, originando que la proyección del vector orientación sobre $\hat{e}$ sea en su mayor parte 1 (diferencias entre ángulos de 0°).

Como era de esperar, ocurre que conforme aumentamos el ruido en el sistema, las orientaciones conservan la misma media, pero las colas de los histogramas se ensanchan (aumenta su dispersión), originando orientaciones que para ruidos de amplitud baja no se percibían. Esto repercute directamente en la proyección sobre el target, originando diferencias entre ángulos mayores. Este tipo de configuraciones en las que el abanico de orientaciones es mayor, se caracterizan porque no favorecen individualmente al peatón (al no permitirle tener una orientación más próxima a su orientación deseada), pero sí favorecen globalmente al sistema, evitando congestionarlo. A efectos globales y como hemos podido comprobar en Fig. 4.2, resulta más efectivo inducir una distribución de orientaciones con mayor varianza en ella, originando así configuraciones del sistema menos compactas y favoreciendo el proceso de evacuación.

En cuanto a las diferencias en las distribuciones para distintos valores de rigidez angular, el comportamiento que observamos es bastante similar. Si bien puede observarse que las distribuciones tienen mayor varianza conforme para menores valores de S_D, en principio no podríamos justificar de manera clara por qué se presentan las diferencias de flujo observadas en Fig. 4.2 atendiendo únicamente a las orientaciones de los individuos.

Modelo con obstáculo

Como último punto del trabajo, y motivados por los experimentos realizados por Pastor et al. [23, 24], estudiaremos las mismas magnitudes que las hasta ahora presentadas, añadiendo un obstáculo al sistema que los peatones deberán sortear.

Se cree que la colocación de obstáculos frente a las puertas es una estrategia efectiva para aumentar el flujo de peatones, mejorando así el proceso de evacuación. Desde que se sugirió por primera vez [5], esta característica contraintuitiva se considera un sello distintivo de los flujos de partículas autopropulsadas. De hecho, a pesar de la poca evidencia experimental [21, 22], la colocación de un obstáculo ha sido aclamada como una de las soluciones más óptimas para solucionar los problemas de evacuación. A pesar de ello, los últimos resultados experimentales obtenidos demuestran que la magnitud del flujo de peatones es insensible a la presencia del obstáculo. Este resultado está en desacuerdo con las recientes demostraciones sobre su idoneidad para los casos de medios granulares [25], ovejas [15] y ratones [26], advirtiéndonos sobre la peligrosidad de extrapolar el comportamiento animal a los humanos.

Siguiendo esta línea, en nuestro modelo hemos querido implementar esta idea introduciendo un obstáculo en el sistema. Hemos fijado el valor de $S_D = 15$ (aquel que nos daba mejores resultados en Fig. 4.2) y estudiaremos cómo se ve afectado el flujo para distintas distancias entre el obstáculo y la puerta, variando el ruido de las partículas. También hemos determinado las distribuciones de las orientaciones dada la región de estudio utilizada en el primero de los casos (Fig. 4.3).

La configuración del obstáculo ha resultado ser una esfera compacta de radio constante $R_o = 0.5$. Además, hemos querido añadir una pequeña modificación en Ec. 3.10 interpretando de distinta manera el vector $\hat{e}$ o target. En base a esto, hemos realizado dos tipos de simulaciones:

1. La primera de ellas sin modificación alguna sobre $\hat{e}$. De esta forma, los peatones, independientemente de su posición, tienden a orientarse en la dirección que une el centro de esferocilindro con el punto más cercano de la puerta. Esta forma de modelizar cómo los peatones evitan el obstáculo no es nada realista, dado que para ellos es como si no existiese, comportándose de la misma forma que en el sistema sin obstáculo.

2. La segunda forma es algo más realista. Según Fig. 4.8, se definen dos zonas en el sistema. En color azul, la zona delimitada por las dos rectas tangentes al obstáculo y que pasan por ambos límites de la puerta de salida. En verde el resto del sistema. Así pues, si un peatón se encuentra en la zona azul, redefinimos el vector $\hat{e}$ de forma que apunte en la dirección definida por la recta tangente que une la posición del peatón y el punto de tangencia sobre el obstáculo. De esta forma conseguimos que el peatón más o menos rodee el obstáculo.

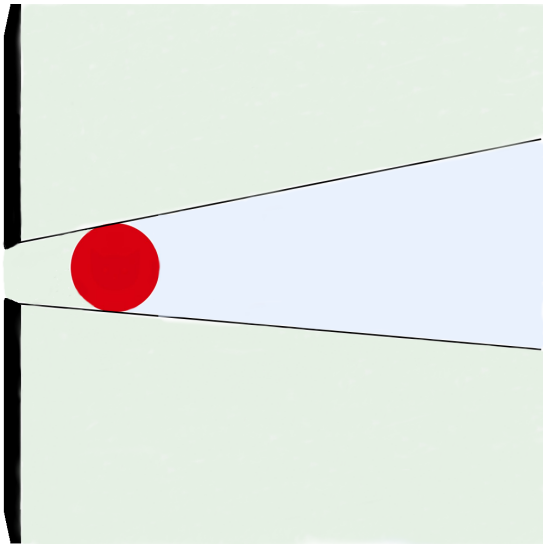


Figura 4.8: Definición de las regiones de orientación. En azul, zona en la que los peatones se orientan de forma tangente al obstáculo. En verde, se orientan de la forma habitual.

El cálculo del punto de tangencia sobre una circunferencia dado un punto exterior es un problema geométrico sencillo y su desarrollo lo detallamos en Fig. 4.9.

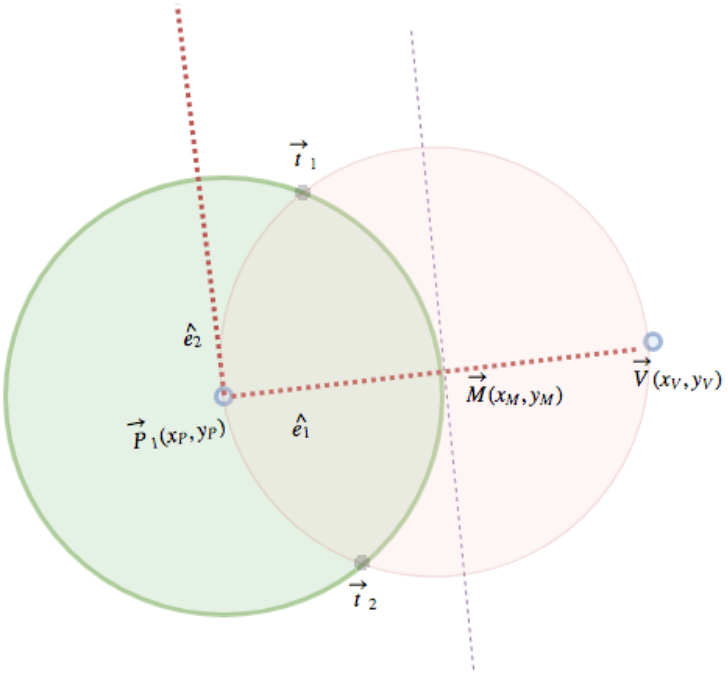


Figura 4.9: Obtención de los dos puntos de tangencia sobre una circunferencia dado un punto exterior.

Dado el centro del obstáculo $\vec{P}_1$ y la posición del peatón $\vec{V}$ (ambos vectores referidos a tierra), calculamos el punto medio entre $\vec{P}_1$ y $\vec{V}$. Lo definimos como $\vec{M}$. Calculado $\vec{M}$, definimos un nuevo sistema de ejes centrados en $\vec{P}_1$ que estará determinado por la base:

$$\hat{e_1} = \frac{\vec{M} - \vec{P}_1}{|\vec{M} - \vec{P}_1|} \qquad \hat{e_2} = rot(90)\hat{e_1} = (-e_1^y, e_1^x) \tag{4.1}$$

Así, como el radio de la circunferencia imaginaria con centro en $\vec{M}$:

$$R_M = |\overrightarrow{MP}| \tag{4.2}$$

Definimos ahora dos circunferencias con centros en $\vec{P_1}$ y $\vec{M}$ y radios R_o y R_M respectivamente. Desde el sistema de referencia de $\hat{e_1}$ y $\hat{e_2}$, las ecuaciones reducidas de dichas circunferencias serán:

$$x^2 + y^2 = R_o^2 \tag{4.3}$$

$$(x - R_M)^2 + y^2 = R_M^2 \tag{4.4}$$

Sustituyendo Ec. 4.3 en 4.4, obtenemos:

$$(x - R_M)^2 + R_o^2 - x^2 = R_M^2 \tag{4.5}$$

De donde se obtiene un valor de x igual a :

$$x = \frac{R_o^2}{2R_M} \tag{4.6}$$

Finalmente, sustituyendo en 4.3, obtenemos dos valores para la coordenada y

$$y = \pm\sqrt{1 - \left(\frac{R_o}{2R_M}^2\right)} \tag{4.7}$$

En donde la raíz siempre está definida si el punto V es externo a la circunferencia. Finalmente, para obtener las expresiones de los puntos de tangencia respecto a tierra:

$$\overrightarrow{t}_1 = \overrightarrow{P}_1 + x\hat{e_1} + y\hat{e_2} \tag{4.8}$$

$$\overrightarrow{t}_2 = \overrightarrow{P}_1 + x\hat{e_1} - y\hat{e_2} \tag{4.9}$$

La elección de uno u otro punto de tangencia dependerá de si la posición del peatón está por encima o por debajo del centro del obstáculo. Así pues, si un peatón se encuentra por encima del centro de este, tomaremos como punto de tangencia Ec. 4.8. En caso contrario, tomaremos 4.9

Estudio del flujo de peatones a través de la puerta

En esta ocasión, volveremos a estudiar la relación entre el flujo y el ruido del movimiento individual de los peatones, para distintas distancias entre el obstáculo y la puerta de salida. Con ello, pretenderemos corroborar los resultados experimentales comentados anteriormente, así como esclarecer los efectos que tendría la colocación de un obstáculo sobre el proceso de evacuación.

Así pues, hemos determinado el flujo para valores de ruido (η) similares al caso anterior (sección 4.1.1) y para cuatro distancias entre la puerta y el obstáculo: $50, 75, 100$ y 125 cm, las cuales corresponden al experimento real llevado a cabo en [13].

Los valores de flujo obtenidos para las dos aproximaciones desarrolladas, se detallan en Fig. 4.10. Como se puede observar, los valores de flujo obtenidos con ciertas combinaciones de los parámetros, llegan a reproducir correctamente los valores obtenidos en [24] y que oscilan entre [2 y 2.5 pers/seg]. Esto induce a pensar que la implementación del modelo con obstáculo ha sido correcta.

Resulta interesante comparar la curva de flujo obtenida en el sistema sin obstáculo con el resto de curvas. A raíz de los resultados experimentales realizados en [23, 24], conocíamos que la introducción de un obstáculo al sistema no beneficiaba, por el momento en humanos, el proceso de evacuación. Observando Fig. 4.10, nos damos cuenta que nuestro algoritmo ha conseguido

reproducir dicho comportamiento, al encontrarse todas las curvas asociadas a las distintas distancias del obstáculo, por debajo de los valores de flujo obtenidos para el sistema sin él. Así pues, podemos concluir que, independientemente de la distancia del obstáculo, los valores del flujo no mejoran los resultados previamente obtenidos. Además, conforme se reduce la distancia entre la puerta y este, el comportamiento negativo se acentúa aún más, obteniéndose los valores más bajos cuando el obstáculo está en la posición más cercana a la puerta.

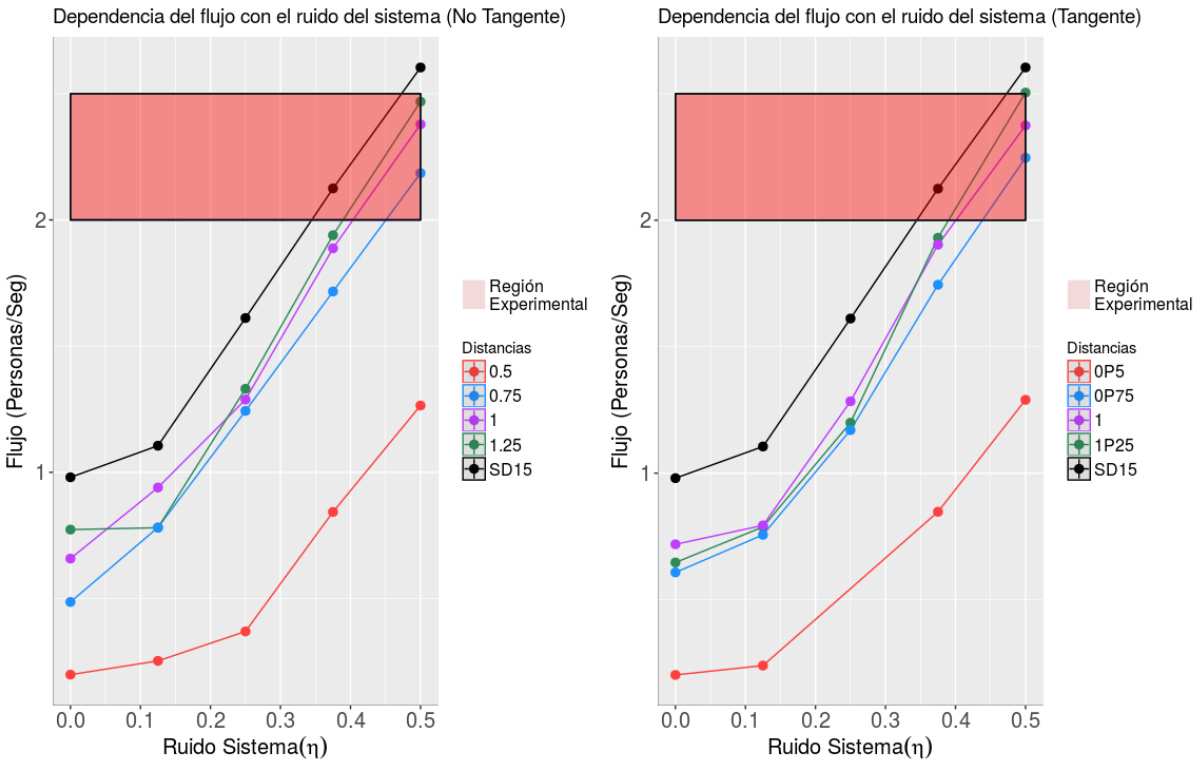
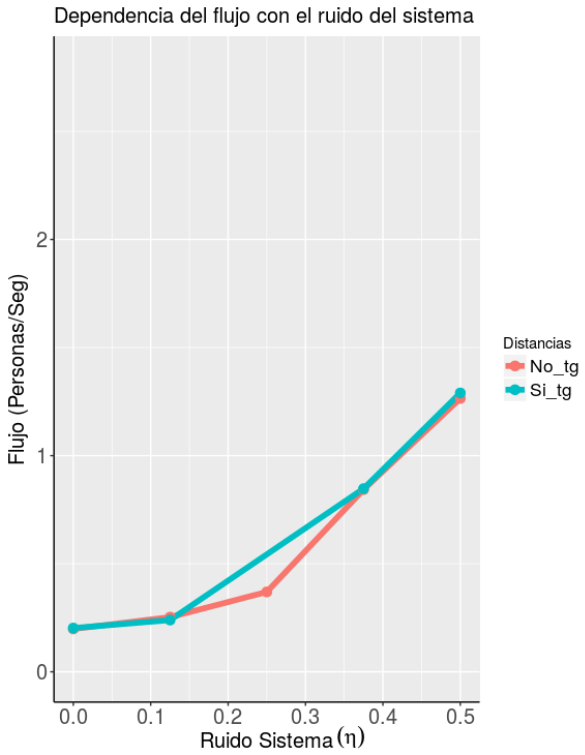


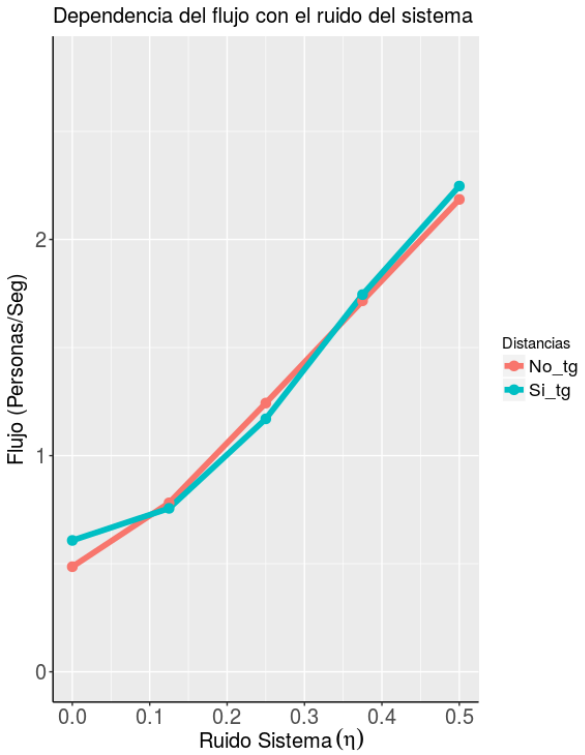
Figura 4.10: Dependencia del flujo con respecto al ruido para distintas distancias del obstáculo a la puerta de salida. A la izquierda, primer tipo de simulación comentada en donde el peatón solo se fija en la puerta. A la derecha, segundo tipo de simulación en donde la orientación deseada del peatón depende de su posición con respecto al obstáculo.

La relación del flujo con el ruido del sistema parece mantenerse inalterable independientemente de la introducción del obstáculo. Como podemos comprobar en la Fig. 4.10, el flujo mantiene un comportamiento monótono conforme el ruido aumenta.

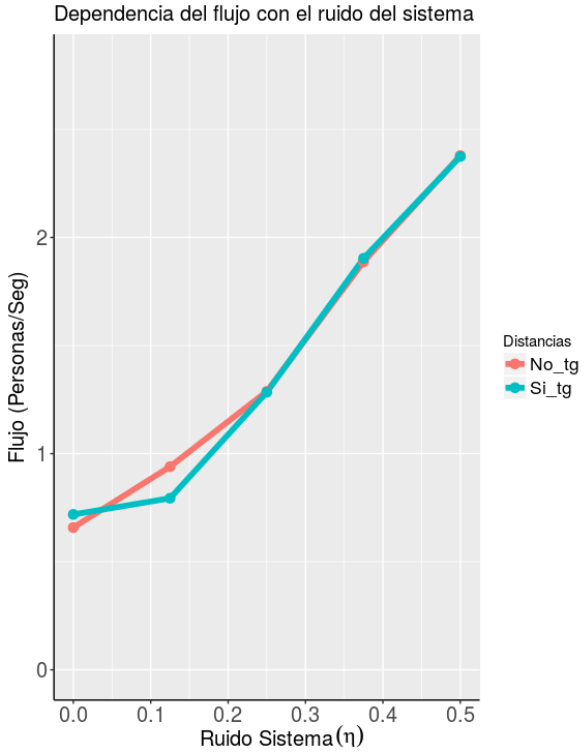
Adicionalmente, hemos realizado la comparativa entre pares de curvas que se corresponden con una misma distancia del obstáculo. De ese modo podremos apreciar si existe alguna diferencia entre las dos implementaciones descritas anteriormente. Los resultados pueden observarse en Fig. 4.11.



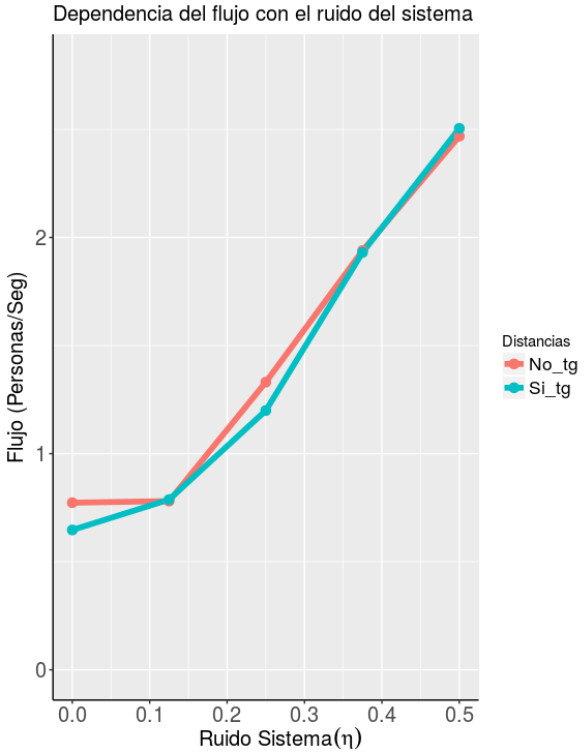
(a) Distancia del obstáculo = **0.5 metros.**



(b) Distancia del obstáculo = **0.75 metros.**



(c) Distancia del obstáculo = **1 metro.**



(d) Distancia del obstáculo = **1.25 metros.**

Figura 4.11: Comparativa entre pares de curvas con misma distancia de obstáculo.

A priori no se observan grandes diferencias entre el modelo *No tangente* y el modelo *Tangente* (en adelante pasaremos a denominarlos así). Hasta el momento, las razones que justifiquen este comportamiento no han sido esclarecidas. Podríamos comentar, que la razón más directa sería que nuestra implementación no capturara correctamente las transmisiones de impulso que se originarían en caso de que los peatones rodeasen el obstáculo, en lugar de enfrentarse a él. Básicamente, la introducción del obstáculo al sistema se propone como método para reducir la presión en las cercanías de la salida. Proponer modelos que no modelizan correctamente las transmisiones de impulso entre peatón y obstáculo, implica obtener resultados similares a simplemente actuar como si el obstáculo no existiera, concentrándose únicamente en la puerta de salida.

Una segunda justificación provendría de la elección de los parámetros del modelo. Debemos comentar que el modelo utilizado está en proceso de validación; esto es, actualmente se está estudiando cuáles son los parámetros relevantes que consiguen reproducir correctamente el comportamiento de los peatones durante las evacuaciones. Uno de los aspectos más importantes en la modelización de peatones es el mecanismo de autopropulsión que se utiliza. En nuestro caso, si recordamos Ec. 3.2, el parámetro V_d o velocidad deseada juega un papel relevante [24]. Dado que su determinación experimental es realmente dificultosa y varía dependiendo del proceso de evacuación realizado, es posible que la elección de dicho parámetro haya sido subestimada y no hayamos conseguido representar correctamente la autopropulsión del sistema como para advertir cambios en las transmisiones de impulso entre el obstáculo y el peatón.

Uno de los aspectos futuros a tener en cuenta y siguiendo esta línea argumentativa sería aumentar el valor de V_D, haciendo más competitivo el sistema y observando si así se aprecia algún cambio en las magnitudes hasta ahora medidas.

Distribución de orientaciones cerca de la salida

Por último, hemos vuelto a estudiar la distribución de las orientaciones de los peatones en posiciones cercanas a la puerta de salida. Dado que en este caso, trabajamos con cuatro posiciones de obstáculo distintas y para cada una de ellas tenemos 5 valores de ruido, en lugar de presentar los histogramas, hemos estudiado directamente los parámetros de las distribuciones.

La media resultó nuevamente ser la orientación de 180 grados. En cuanto al estudio de las desviaciones estándar, pasamos a detallarlo en Fig. 4.12. Como podemos comprobar, la tendencia general que sigue el sistema es que conforme aumentamos el ruido del mismo, la desviación en las orientaciones aumenta. Existen sin embargo algunos puntos que no siguen esta tendencia y que precisarían de un estudio particular o una mayor estadística para su correcta interpretación.

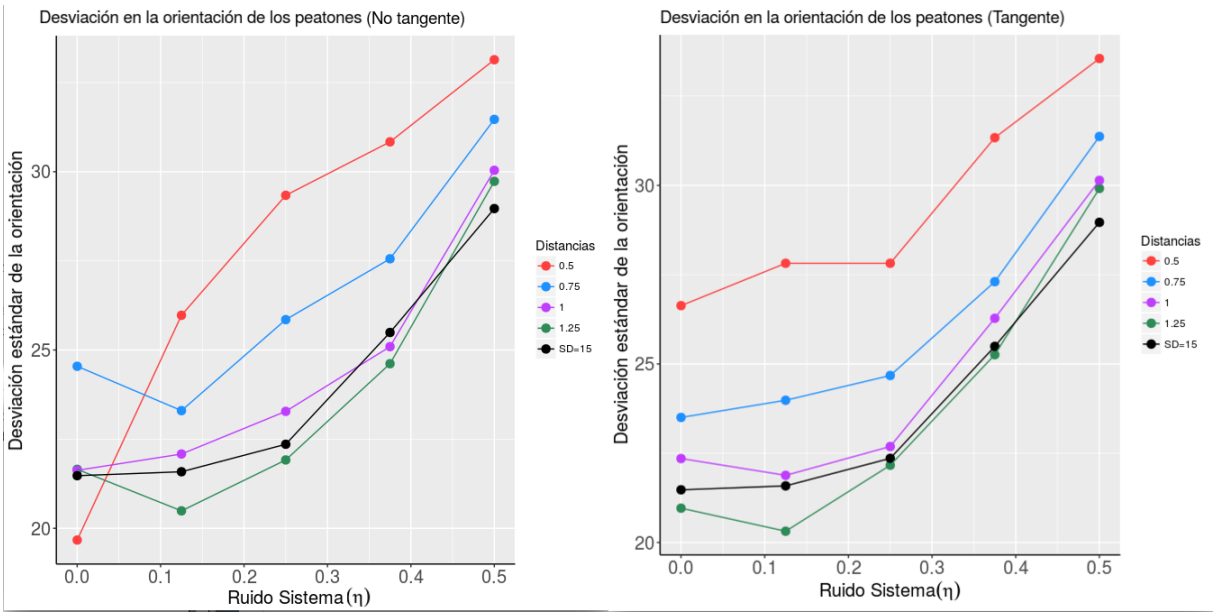


Figura 4.12: Desviación estandar en la orientaciones de los peatones para distintos valores de ruido y distancias del obstáculo. A la izquierda el modelo *No tangente*. A la derecha el modelo *Tangente*.

En cuanto a las diferencias entre los dos modelos de cruce de obstáculo, no logramos apreciar comportamientos distintos, salvo para el punto ($\eta = 0$, dist $= 0.5$). En este caso, parace ocurrir que en el modelo *No tangente*, la desviación en este punto es bastante menor que para el caso del modelo *Tangente*. Esto puede ser debido a que se haya originado cierta configuración geométrica cerca de la puerta de salida que haga que lo peatones, dado el escaso espacio que existe en este caso en particular, solo puedan orientarse en posiciones específicas, haciendo que la desviación del histograma decaiga radicalmente. En el caso del modelo *Tangente*, puede ocurrir que los peatones tiendan a concentrarse en zonas más exteriores a la central, permitiendo que la presión de zonas cercanas a la puerta sea menor y aumentando la desviación en la distribución angular.

Para el resto de combinaciones, no podemos apreciar diferencias significativas. La razón de ello puede justificarse a raíz del estudio llevado a cabo con anterioridad (sección 4.2.1) en donde se ha comprobado que las diferencias presentadas por cada modelo en términos de flujo han sido insignificantes, lo cual sugiere nuevamente que el modelo propuesto podría no capturar correctamente el comportamiento deseado.

Capítulo 5

Conclusiones y líneas futuras

A continuación se resumen las principales ideas tratadas en los capítulos precedentes. Además, se comentan otras ideas que pueden proponerse en la línea de investigación aquí tratada.

En este trabajo se ha introducido un novedoso algoritmo híbrido, CPU-GPU, basado en el método de elementos discretos. La implementación se ha utilizado para simular el movimiento de peatones en condiciones de confinamiento. La gran ventaja de este método frente a otros métodos tradicionales sobre CPU, es que se ahorran tiempos de ejecución muy elevados. Esto nos ha permitido modelar sistemas más realistas donde la geometría de la partícula como la del recinto pueden ser modificadas. Uno de los principales objetivos de este trabajo ha sido el optimizar este proceso, motivo por el cual se han mostrado multitud de detalles técnicos sobre la implementación del algoritmo sobre GPU.

El punto de partida fue modelar sistemas de peatones con forma esferocilíndrica y evaluar el comportamiento del sistema variando algunos parámetros del modelo. Hemos utilizado los valores de flujo obtenidos experimentalmente, para validar si el modelo utilizado presenta una respuesta coherente. Una vez visto que nuestro algoritmo reproduce los valores experimentales, hemos decidido por estudiar algunas propiedades micromecánicas del sistema, como son: orientación de los peatones en posiciones cercanas a la puerta de salida y la proyección de estas sobre la dirección deseada.

Finalmente y motivados por resultados experimentales recientes, hemos añadido un grado de complejidad al sistema, introduciendo un obstáculo. Analizadas las presuposiciones de las que partíamos, nuestro objetivo ha sido comprobar cuál es el papel que juega la presencia del obstáculo y su localización exacta sobre la magnitud de flujo de peatones, y si además esto originaba alguna modificación sobre las propiedades micromecánicas del sistema. Las conclusiones que hemos obtenido han resultado del todo interesantes, en general obtenemos que introducir un obstáculo no resulta beneficioso para mejorar el flujo de personas. Así, nuestros resultados parecen contradecir opiniones preliminares de que la presencia de un obstáculo favorecería el proceso de evacuación. Sin embargo, reproducimos de manera fidedigna resultados experimenta-

les, obtenidos muy recientemente, por investigadores del departamento de Física y Matemáticas de la Universidad de Navarra.

Las nuevas líneas de investigación que abre este trabajo son variadas, algunas de ellas ya comentadas a lo largo del proyecto. Lo principal ahora reside en crear modelos con un grado de competitividad mayor y observar si, efectivamente, el obstáculo sigue siendo un ingrediente negativo para el sistema. Además, se podría seguir estudiando qué posición/posiciones del obstáculo ofrecen valores de flujo similares al sistema sin obstáculo. De la misma forma, resultaría útil a modo de validación, modificar el alineamiento de los peatones y originar un modelo que reprodujera el comportamiento de animales tales como ovejas, ratones, etc. De esta forma, podríamos utilizar nuevos resultados experimentales como validación del modelo hasta conseguir proponer una modelización correcta, que nos permita obtener los aspectos más determinantes que caracterizan un proceso de evacuación.

Finalmente, con los resultados mostrados en este trabajo de fin de máster se espera haber colaborado en cierto modo para comprender un poco mejor algunas de las propiedades intrínsecas de los modelos de peatones, además de servir como punto de partida para el estudio de otras muchas cuestiones que aún no han sido resueltas.

Bibliografía

[1] M. Moussaïd, D. Helbing, and G. Theraulaz. How simple rules determine pedestrian behavior and crowd disasters. *Proceedings of the National Academy of Sciences*, 108(17):6884–6888, 2011.

[2] K. Rogers. Black friday: Crowdsourcing communities of risk. *WSQ: Women's Studies Quarterly*, 40(1):171–186, 2012.

[3] J. Boyd Thomas and C. Peters. An exploratory investigation of black friday consumption rituals. *International Journal of Retail & Distribution Management*, 39(7):522–537, 2011.

[4] A. Kirchner and A. Schadschneider. Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics. *Physica A: statistical mechanics and its applications*, 312(1-2):260–276, 2002.

[5] D. Helbing, I. Farkas, and Vicsek T. Simulating dynamical features of escape panic. *Nature*, 407:487–490, 2000.

[6] F A Tavarez and ME. Plesha. Discrete element method for modelling solid and particulate materials. *International journal for numerical methods in engineering*, 70(4):379–404, 2007.

[7] Stefan Van Baars. Discrete element modelling of granular materials. *Heron*, 41(2):139–157, 1996.

[8] T. Pöschel and T. Schwager. *Computational granular dynamics: models and algorithms*. Springer Science & Business Media, 2005.

[9] M. Rumpf and R. Strzodka. Graphics processor units: New prospects for parallel computing. In *Numerical solution of partial differential equations on parallel computers*, pages 89–132. Springer, 2006.

[10] J. Sanders and E. Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[11] CUDA Nvidia. Nvidia cuda c programming guide. *Nvidia Corporation*, 120(18):8, 2011.

[12] RC. Hidalgo, T. Kanzaki, F. Alonso-Marroquin, and S. Luding. On the use of graphics processing units (gpus) for molecular dynamics simulation of spherical particles. In *AIP Conference Proceedings*, volume 1542, pages 169–172. AIP, 2013.

[13] A. Garcimartín, JM. Pastor, C. Martín-Gómez, D. Parisi, and I. Zuriguel. Pedestrian collective motion in competitive room evacuation. *Scientific reports*, 7(1):10792, 2017.

[14] A. Garcimartín, DR. Parisi, JM. Pastor, C. Martín-Gómez, and I. Zuriguel. Flow of pedestrians through narrow doors with different competitiveness. *Journal of Statistical Mechanics: Theory and Experiment*, 2016(4):043402, 2016.

[15] I. Zuriguel, DR. Parisi, RC. Hidalgo, C. Lozano, A. Janda, PA. Gago, JP. Peralta, LM. Ferrer, LA. Pugnaloni, E. Clément, et al. Clogging transition of many-particle systems flowing through bottlenecks. *Scientific reports*, 4:7324, 2014.

[16] RC. Hidalgo, DR. Parisi, and I. Zuriguel. Simulating competitive egress of noncircular pedestrians. *Physical Review E*, 95(4):042319, 2017.

[17] SM. Johnson, JR. Williams, and BK. Cook. On the application of quaternion-based approaches in discrete element methods. *Engineering Computations*, 26(6):610–620, 2009.

[18] D. Fincham. Leapfrog rotational algorithms. *Molecular Simulation*, 8(3-5):165–178, 1992.

[19] A. Garcimartín, JM. Pastor, LM. Ferrer, JJ. Ramos, C. Martín-Gómez, and I. Zuriguel. Flow and clogging of a sheep herd passing through a bottleneck. *Physical Review E*, 91(2):022808, 2015.

[20] A. Kirchner, K. Nishinari, and A. Schadschneider. Friction effects and clogging in a cellular automaton model for pedestrian dynamics. *Phys. Rev. E*, 67:056122, 2003.

[21] D. Helbing, L. Buzna, A. Johansson, and T. Werner. Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions. *Transportation science*, 39(1):1–24, 2005.

[22] L. Jiang, J. Li, C. Shen, S. Yang, and Z. Han. Obstacle optimization for panic flow-reducing the tangential momentum increases the escape speed. *PloS one*, 9(12):e115463, 2014.

[23] JM. Pastor, A. Garcimartín, PA. Gago, JP. Peralta, C. Martín-Gómez, LM. Ferrer, D. Maza, DR. Parisi, LA. Pugnaloni, and I. Zuriguel. Experimental proof of faster-is-slower in systems of frictional particles flowing through constrictions. *Physical Review E*, 92(6):062817, 2015.

[24] A. Garcimartín, D. Maza, JM. Pastor, D. Parisi, C. Martín-Gómez, and I. Zuriguel. Redifining the role of oobstacle in pedestrian evacuation. 2018.

[25] I. Zuriguel, A. Janda, A. Garcimartín, C. Lozano, R. Arévalo, and D. Maza. Silo clogging reduction by the presence of an obstacle. *Physical review letters*, 107(27):278001, 2011.

[26] P. Lin, J. Ma, T. Liu, T. Ran, Y. Si, F. Wu, and G. Wang. An experimental study of the impact of an obstacle on the escape efficiency by using mice under high competition. *Physica A: Statistical Mechanics and its Applications*, 482:228–242, 2017.