Práctica 4 - Simulación de Mallas Deformables y Roturas

Duración: 1 Sesión

Objetivos:

- Saber simular una malla deformable mediante una estructura de tipo masa-muelle bidimensional (2D).
- Utilizar diferentes disposiciones para la estructura masa-muelle 2D y entender los diferentes comportamientos que se producen.
- Saber generar roturas en la malla.
- Permitir la interacción de la malla deformable con otros objetos que colisionen con ella.

Entrega: 1 semana después de finalizar la práctica. Se debe entregar el código y una memoria/documento de análisis, donde se detallen los resultados obtenidos y se analicen e interpreten estos resultados.

Descripción General de la Simulación

El objetivo de la práctica es simular una malla deformable rectangular que se comporte adecuadamente cuando se le deja caer sobre ella un objeto esférico. Como la malla no es rígida, su estructura debe ser capaz de deformarse de forma realista en función de las fuerzas que actúen sobre sus diferentes partes.

Existen diferentes maneras de realizar esta simulación. En esta práctica simularemos la malla como un conjunto de pequeñas masas unidas (que llamaremos *nodos*) por muelles. Por tanto, la malla no será más que una estructura rectangular de masas y de muelles que unen esas masas entre sí. Posteriormente, se añadirá a la simulación la posibilidad de que distintas partes de la malla puedan romperse. Finalmente, simularemos la caída de una esfera sobre la malla, de modo que la esfera y la malla colisionen adecuadamente. Para la colisión usaremos también un modelo basado en muelles.

Se proporciona la estructura básica de clases del programa:

· *Programa principal*: inicializa y controla la simulación (y también el dibujado en 3D de la misma) a través del uso de los distintos objetos utilizados en la misma. También contiene constantes para cada uno de los parámetros de la simulación y una interfaz simple para cambiar algunos de estos parámetros. **Este programa principal se proporciona completo**, y permite (**sin realizar ningún cambio en el mismo salvo aquellos que sirvan para cambiar**

los valores de los parámetros de la simulación) simular en tiempo real el escenario de esta práctica si se implementan correctamente las diferentes clases del programa.

· Clase *Particle*: simula el movimiento de una partícula sin dimensiones. La utilizaremos para describir y simular el movimiento de cada uno los nodos de la malla. La clase realiza la integración numérica y simula el movimiento debido a la gravedad y a cualquier otra fuerza externa que se le pase en la función *addExternalForce()*. Para permitir que haya nodos de la malla que no se muevan (aquellos que definen el perímetro de la malla), la clase contiene una variable *_clamped*, que indica que la partícula no se moverá incluso aunque actúen fuerzas sobre ella. Además, cada partícula contiene un identificador (*_id*) único que puede ser útil para identificar las partículas que pertenecen a un muelle y evitar que se creen muelles que ya existan. Como suponemos que los nodos de la malla son adimensionales, en principio las partículas no se dibujarán. **Esta clase se proporciona completa y en principio no debe modificarse a no ser que quieran realizarse mejoras.**

· Clase *Ball*: hereda de *Particle* y simula el movimiento de la esfera. Es prácticamente idéntica a su clase padre (*Particle*), ya que sólo añade información para dibujarla, puesto que, a diferencia de las partículas, la esfera no es adimensional. **Esta clase se proporciona completa y en principio no debe modificarse a no ser que quieran realizarse mejoras**.

· Clase *DampedSpring*: representa un muelle con **amortiguación lineal**. Los muelles se usarán tanto para crear la malla deformable como para tratar las colisiones entre la esfera y la malla. Por eso, esta clase es muy importante y debe tener algunas particularidades especiales para que pueda servir para varios propósitos. En primer lugar, debe poder configurarse para que, si se desea, actúe sólo repeliendo (*_repulsionOnly*) las partículas (esto es muy útil cuando se usan los muelles para tratar colisiones). Además, la clase debe permitir que el muelle se rompa y deje de ejercer fuerza sobre las partículas. Para ello, no es necesario destruir el objeto muelle. Simplemente, es necesario detectar en la propia clase la condición de rotura y dejar de aplicar fuerzas a las partículas cuando el muelle esté roto. De esta forma, si se quiere reparar o reutilizar el muelle, no hace falta volver a crearlo. Sólo hace falta cambiar el valor de la variable que indica que está roto (*_broken*). **Esta clase se proporciona incompleta, aunque sólo falta por realizar la función *update()***.

· Clase *DeformableSurface*: representa la malla deformable. Esta clase contiene una matriz rectangular de nodos (*_nodes*) que representa a las partículas que forman la malla, una lista de muelles (*_springsSurface*) que sirve para almacenar los muelles que se crean entre los diferentes nodos de la malla, y un diccionario con los muelles que se usarán para simular la colisión entre la esfera y los nodos de la malla (*_springsCollision*). **Esta clase se proporciona incompleta, puesto que faltan por realizar las funciones *createNodes()*, *createSurfaceSprings()* y *avoidCollisions()*.**

Primera Parte – Simulación de la Malla

Comenzaremos simulando el comportamiento de la malla deformable en presencia de gravedad, sin que haya otras interacciones externas. Crearemos la malla en el plano XY a una cierta altura sobre el eje Z. Los parámetros principales de la malla (todos ellos están ya definidos en el programa principal y tienen también sus correspondientes variables miembro en las clases correspondientes) son los siguientes:

NET_LENGTH_X: longitud total de la malla en dirección X (m).
NET_LENGTH_Y: longitud total de la malla en dirección Y (m).
NET_POS_Z: posición de la malla sobre el eje Z (m).
NET_NUMBER_OF_NODES_X: número de nodos de la malla en dirección X.
NET_NUMBER_OF_NODES_Y: número de nodos de la malla en dirección Y.
NET_NODE_MASS: masa de cada uno de los nodos de la malla (kg).
NET_KE: constante elástica de cada uno de los muelles de la malla (N/m).
NET_KD: constante de amortiguamiento lineal de cada uno de los muelles de la malla (kg/s).

Además, hay otros parámetros, propios de la simulación:

G: aceleración de la gravedad (m/s^2).
SIM_STEP: paso de simulación (s).

Existen diferentes formas de unir los nodos de la malla mediante muelles:

- Unión estructural (*STRUCTURAL*): donde cada nodo se une con los nodos adyacentes a izquierda, derecha, arriba y abajo.
- Unión en tijera (*SHEAR):* donde cada nodo se une con los nodos que se encuentran en sus diagonales.
- Unión anti-arrugas (*BEND*): donde cada nodo se une con los nodos que se encuentran a una distancia de 2 nodos, tanto horizontal como verticalmente.

Estas tres disposiciones no son excluyentes y se pueden combinar. En esta práctica se deben implementar las siguientes combinaciones:

· *structural*
· *shear*
· *structural + shear*

Segunda Parte – Simulación de la Rotura de la Malla

Se desea implementar, además, la rotura de la malla. Los objetos deformables no pueden generalmente deformarse sin límite, ya que, si se estiran mucho, se rompen. Vamos a modelar este efecto rompiendo aquellos muelles que estén sometidos a una carga muy grande. En concreto, vamos a suponer que los muelles que forman la malla se romperán cuando ocurra alguna de estas dos circunstancias:

- La fuerza que ejerce el muelle supera un cierto umbral.
- Las partículas que une ese muelle se separan demasiado, superando un cierto umbral de distancia.

Modelaremos este efecto de rotura, pues, rompiendo el muelle cuando la fuerza del mismo supere un cierto umbral o cuando la distancia supere un cierto umbral. Para la rotura por distancia, lo más sencillo es establecer este parámetro de distancia umbral como un factor que multiplica a la distancia de reposo del muelle. Por ejemplo, si el factor vale 2, querrá decir que el muelle se romperá cuando la distancia entre las partículas supere el doble de la longitud de reposo.

Los parámetros principales para esta parte de la simulación (todos ellos están ya definidos en el programa principal y tienen también sus correspondientes variables miembro en las clases correspondientes) son:

NET_MAX_FORCE: máxima fuerza (N) que puede soportar cada muelle de la malla. A partir de este umbral, se romperá el muelle. Si vale 0, supondremos que el muelle no se puede romper por esta causa.
NET_BREAK_LENGTH_FACTOR: factor que multiplica a la distancia de reposo de cada muelle de la malla para decidir si se rompe el muelle o no. Si vale 0, supondremos que el muelle no se puede romper por esta causa. El valor de este parámetro debería ser normalmente mayor que 1, ya que, si no es así, los muelles se romperán incluso estando elongados una cantidad inferior a la distancia de reposo.

Para romper el muelle habrá que detectar si se cumple alguna de estas dos causas, y marcar el muelle como roto (*_broken*). Por tanto, la rotura del muelle es realmente muy fácil de implementar.

El dibujado de la clase *DeformableSurface* ya tiene en cuenta la rotura de muelles y no dibujará los muelles que estén marcados como rotos.

Tercera Parte – Simulación de la Esfera y su Colisión con la Malla

Para simular la esfera, asumiremos que esta se comporta como una partícula. Esto es una simplificación, puesto que la esfera no es obviamente una partícula adimensional. Tiene volumen y puede girar, pero ignoraremos las rotaciones y simularemos la esfera como una partícula, aunque lógicamente la dibujaremos con volumen.

La simulación del movimiento de la esfera no reviste ninguna dificultad puesto que ya se ha hecho en otras prácticas (de hecho, la simulación ya la hace la clase padre *Particle* por lo que no hay que repetir el código en la clase *Ball*).

Sin embargo, hemos de ser capaces de evitar que la esfera atraviese la malla cuando colisione contra ella. Aunque existen otros modelos de colisión posibles, vamos a suponer que cuando la esfera se acerque demasiado a cada uno de los nodos de la malla (que son en realidad partículas de pequeña masa) se generará un muelle (entre la esfera y el nodo de la malla) para evitar que la esfera atraviese ese nodo de la malla. Para implementar este muelle se puede usar **la misma clase** *DampedSpring* que se usa para los muelles que forman la malla. Sin embargo, este muelle **sólo debe generar fuerza de repulsión** (nunca de atracción). Eso evitará que el muelle siga actuando cuando la colisión haya sido detenida y la esfera se aleje de la malla. La clase *DampedSpring* debe permitir generar muelles de sólo repulsión con la variable *_repulsionOnly*. Además (aunque esto no es imprescindible) se podría implementar también la rotura del muelle de colisión si se supera una cierta fuerza umbral o una cierta distancia.

Para realizar la gestión de colisiones se usará un diccionario llamado *_springsCollision* (de tipo *TreeMap*) que almacenará los muelles que se emplean para simular la colisión entre la esfera y los nodos de la malla. Este diccionario estará indexado mediante un nombre (de tipo *String*) único para cada muelle de colisión. Para asegurarnos de que el nombre de cada muelle sea único y poder posteriormente buscar el muelle una vez haya sido añadido (para evitar crear muelles duplicados para la misma colisión), lo más fácil es darle a cada muelle un nombre que involucre los dos identificadores de los nodos que une. Por ejemplo: "<*Id_particula1*>-<*Id_particula2*>", donde <*Id_particula1*> e <*Id_particula2*> serían los números de identificación de cada partícula.

Los parámetros principales de la esfera (todos ellos están ya definidos en el programa principal y tienen también sus correspondientes variables miembro en las clases correspondientes) son:

BALL_MASS: masa de la esfera (kg).
BALL_RADIUS: radio de la esfera (m).

Los parámetros principales de los muelles de colisión (todos ellos están ya definidos en el programa principal y tienen también sus correspondientes variables miembro en las clases correspondientes) son:

COLLISION_KE: constante elástica de cada uno de los muelles de colisión (N/m).
COLLISION_KD: constante de amortiguamiento de cada uno de los muelles de colisión (kg/s).
COLLISION_MAX_FORCE: máxima fuerza (N) que puede soportar cada muelle de colisión. A partir de este umbral, se romperá el muelle. Si vale 0, supondremos que el muelle no se puede romper por esta causa.
COLLISION_BREAK_LENGTH_FACTOR: factor que multiplica a la distancia de reposo para decidir si se rompe el muelle. Si vale 0, supondremos que el muelle no se puede romper por esta causa. El valor de este parámetro debería ser normalmente mayor que 1, ya que, si no es así, los muelles se romperán incluso estando elongados una cantidad inferior a la distancia de reposo.

Entrega

Se deberá entregar el código y una memoria en la que se expliquen las soluciones implementadas (incluyendo las ecuaciones que se han implementado para el muelle con amortiguación lineal) y se reflexione sobre las siguientes cuestiones:

1- ¿Qué diferencias se observan entre los tipos de disposiciones (*structural*, *shear*, etc.) de muelles de la malla? ¿Cambia el comportamiento de la simulación? ¿En qué casos?
2- ¿Son todas las disposiciones igual de costosas a nivel de tiempo de computación? ¿Por qué?
3- ¿Qué puede ocurrir si la esfera tiene un radio menor que la distancia entre los nodos de la malla? ¿Por qué?
4- ¿Qué ocurre si la velocidad de la esfera es extremadamente alta cuando la malla es irrompible? ¿y cuándo no es irrompible? ¿Por qué?
5- ¿Qué ocurre cuando la masa de la esfera es muy grande en relación a la de los nodos de la malla? ¿Por qué?
6- ¿En qué condiciones es la simulación inestable?

Se deben incluir en la memoria enlaces a videos (con comentarios textuales o de voz) de tu simulación que ilustren las diferentes situaciones que se comentan en la lista de cuestiones anterior.

Opcional 1:

Implementar las siguientes combinaciones para la disposición de los muelles:
· *structural + bend*
· *structural + shear + bend*

Opcional 2:

Contesta a estas preguntas adicionales

1- ¿Cómo podemos simular una malla que se comporte como una colchoneta elástica?
2- ¿Cómo podemos simular una malla que se comporte como una manta?
3- ¿Cómo podemos simular una malla que se comporte como una alambrada metálica de las que se utilizan para vallar terrenos?
4- ¿Cómo podemos simular una malla que se comporte como una plancha rígida de metal?

Notas Importantes:

1) Es fundamental entender lo que hace y pretende hacer el código que se proporciona antes de realizar modificaciones sobre el mismo. En principio sólo hace falta implementar la función *update()* de la clase *DampedSpring* y las funciones *createNodes(), createSurfaceSprings() y avoidCollisions()* de la clase *DeformableSurface.* **Si se respeta la estructura que se proporciona, no es necesario realizar ni modificar ninguna función más**, a no ser que se deseen añadir funcionalidades adicionales o mejoras a la simulación.

2) Se debe también evitar añadir parámetros o variables que ya estén implementadas en el programa proporcionado, y/o utilizarlos fuera de su ámbito.

3) Para utilizar el programa se debe instalar la biblioteca **PeasyCam** previamente (si no estaba ya instalada). Esta biblioteca se puede añadir desde la pestaña *Libraries* (dentro de *Herramientas -> Añadir Herramientas*) de Processing.

Anexo – Capturas de Pantalla

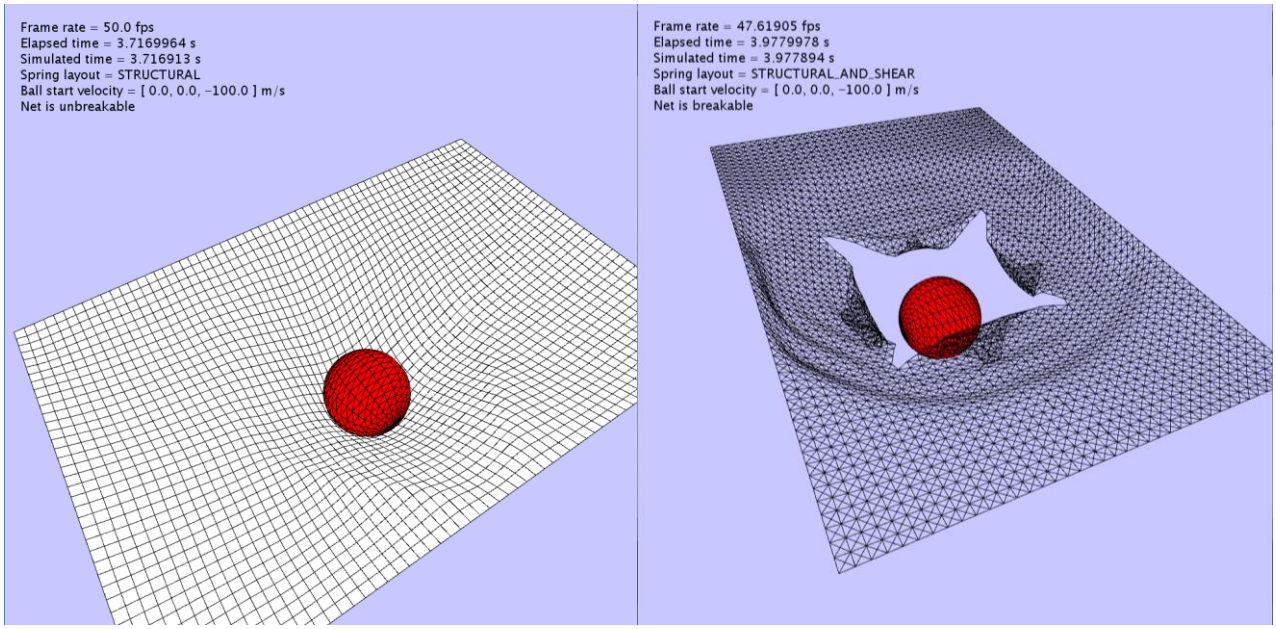

Figura 1 - Capturas de pantalla de la aplicación funcionando. A la izquierda se muestra la malla cuando no se puede romper (se dibuja como una superficie continua, sin dibujar los muelles). A la derecha se muestra la malla cuando sí se puede romper (no se dibuja como una superficie, se dibujan los muelles como segmentos siempre que no estén rotos).