

## Práctica 1 – Parte 2: Masa-Muelle y ODEs

(Asignación 20 de Septiembre; Entrega 13 de Octubre a las 23:59)

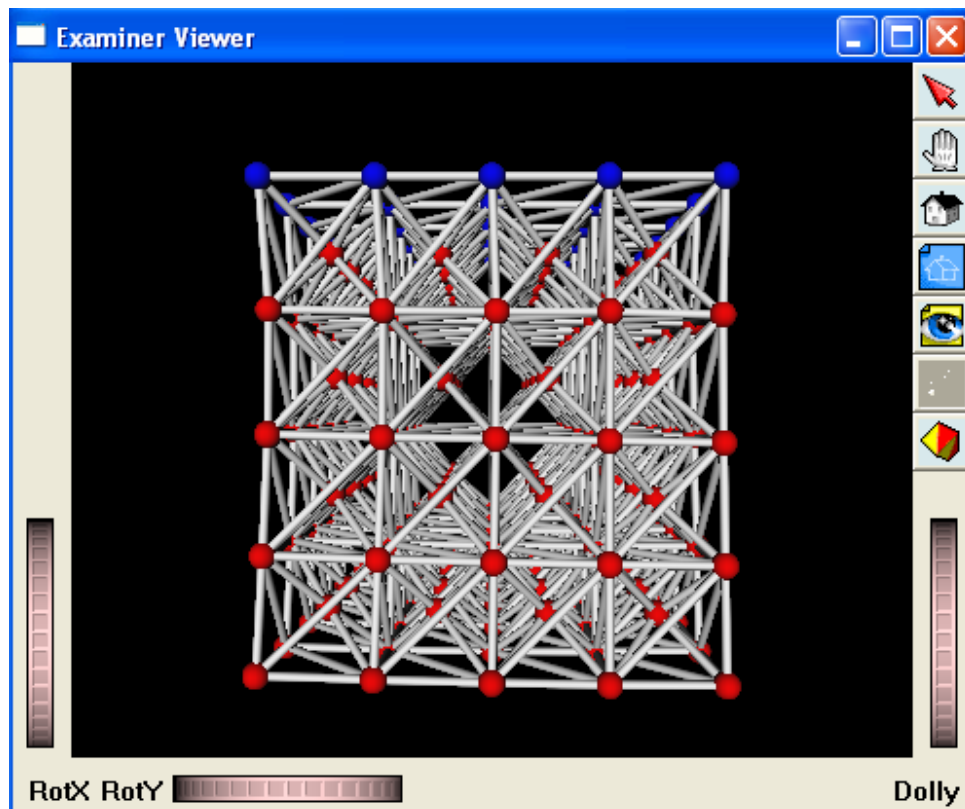
### ¿Cómo entregar la práctica?

Enviar una copia de los ficheros ExercisePoint.cpp y ExerciseSpring.cpp por email a [miguel.otaduy@urjc.es](mailto:miguel.otaduy@urjc.es), antes del 13 de Octubre a las 23:59.

### Especificaciones Generales:

Se proporciona un proyecto sobre MS VisualStudio 2005 que se encarga de inicializar y configurar la simulación, ejecutar el bucle de simulación, y visualizar los resultados utilizando Coin3D. Previamente, se han de instalar Coin3D (versión 2) y SoWin. La práctica también se puede realizar en otras plataformas (Linux, Mac...), ya que el fichero Exercise.cpp no hace referencia a SoWin.

El ejercicio consiste en la simulación de una malla deformable mediante el método masa-muelle y el método de integración Euler semi-implícito. En concreto, se han de programar (1) el cálculo de fuerzas y sus derivadas de una clase muelle, y (2) la inserción de los términos de masa y rigidez en una matriz dispersa. Las rutinas a programar se incluirán en los ficheros ExercisePoint.cpp y ExerciseSpring.cpp. Todo nuevo código ha de ser añadido en dichos ficheros. No se han de enviar más ficheros, **sólo** ExercisePoint.cpp y ExerciseSpring.cpp. La práctica está preparada para que no necesite librerías adicionales. La práctica se evaluará intercambiando los ficheros indicados, compilando y ejecutando, con lo cual no compilará si se precisan otros ficheros.



**Explicación de la Demo y el Código:**

La demo carga una malla con elementos Point y Spring, que son visualizados en Coin mediante esferas y cilindros respectivamente. La inicialización de la simulación ya está realizada, así como el bucle de simulación que va actualizando las posiciones de los nodos.

Al iniciar la demo, se encuentra activa la opción de movimiento de cámara en la ventana principal de Coin (icono ‘mano’). La demo comienza en pausa. Para iniciar la demo, se ha de seleccionar la opción de interacción de Coin (icono ‘flecha’), y se ha de pinchar sobre la ventana. Entonces, la tecla ‘s’ permite activar y parar la simulación. Pinchando y arrastrando sobre una esfera roja se aplica una fuerza externa a la masa asociada a dicha esfera.

Se pueden modificar los siguientes parámetros de la demo: (1) paso de simulación, (2) rigidez de los muelles, y (3) número de nodos en cada dirección XYZ. Estos parámetros se pueden modificar por la línea de comandos como argumentos del programa, o en los valores por defecto de la clase Scene. La masa total es por defecto 1.0, y se reparte uniformemente entre todos los nodos. No se ha programado amortiguamiento/damping.

Se aconseja iniciar los experimentos con una malla con muy pocos nodos, para facilitar la depuración. El mínimo es 2x2x2, con un nodo central.

**Clases de Álgebra Lineal:**

Se ha utilizado la clase para vectores 3D de Coin (SbVec3f). Se puede encontrar ayuda en: <http://doc.coin3d.org/Coin/annotated.html>

Si se desea, se pueden utilizar otras clases con vectores, matrices, etc., pero todo el código ha de incluirse en los ficheros a entregar.

**Clase CGSolver:**

Esta clase almacena un sistema de ecuaciones lineal disperso (matriz y vector derecho) y contiene métodos para resolver dicho sistema utilizando el método del gradiente conjugado. Utiliza otras dos clases importantes, ABlock (bloque 3x3 de la matriz A) y bBlock (bloque 3x1 del vector b), para pasarle los datos.

Entre los datos de CGSolver cabe destacar:

`ABlock *diagA` Es un vector con bloques 3x3 de la diagonal de la matriz A.

`ABlock *sparseA` Es un vector con los bloques 3x3 que no están en la diagonal. Ya que la matriz es dispersa, estos bloques también almacenan la fila y columna a la que hacen referencia dentro de la matriz. Los bloques  $A[i][j]$  y  $A[j][i]$  son iguales, al ser la matriz simétrica, por lo que sólo hace falta almacenar uno de ellos.

`bBlock *b` Es un vector con los bloques 3x1 del vector A.

Entre los métodos de CGSolver cabe destacar:

`void AddToDiagBlock(const ABlock& val, int i);` Suma un bloque 3x3 al bloque 3x3 que se encuentra en la posición 'i' de la diagonal de A.

`void AddToSparseBlock(const ABlock& val, int i, int r, int c);` Suma un bloque 3x3 al bloque 3x3 que se encuentra en la posición 'i' de los que no están en la diagonal de A.

`void AddToVectorBlock(const bBlock& val, int i);` Suma un bloque 3x1 en la posición 'i' del vector b.

### **Clase Point:**

Contiene datos para la posición y velocidad de un nodo, así como la fuerza total que actúa sobre dicho nodo. En cada paso de simulación, la fuerza se resetea a 0, y se van sumando las distintas fuerzas que actúan sobre el nodo.

También almacena si el nodo está fijo (fixed) o no. Si está fijo, no es necesario realizar cálculos con él.

Por último, 'simIndex' indica el índice del nodo entre los nodos que no están fijos. Este índice también sirve para saber en qué posición de la matriz A y el vector b se han de almacenar los datos referidos al nodo en cuestión.

### **Clase Spring:**

Contiene punteros a los dos nodos de los extremos del muelle, así como valores de longitud actual (length), longitud de reposo (rest), y rigidez (stiffness). El vector 'position' almacena la posición del punto medio del muelle, y se utiliza en la visualización.

Incluye métodos para saber si ambos nodos están fijos (FullFixed) o si al menos uno está fijo (HalfFixed).

Por último, 'simIndex' indica el índice del muelle, que sirve para indexar los bloques de la parte dispersa de la matriz A, es decir, sparseA.

### **Tarea 1: Cálculo de la fuerza de un muelle.**

#### INTERFAZ DE LA FUNCIÓN:

```
void Spring::AddForces(void);
```

REQUISITOS: Programar el cálculo de la fuerza de un muelle, y la suma de dicha fuerza a los dos nodos conectados al muelle.

#### OTROS DATOS:

- Al llamar a este método ya se ha calculado la longitud del muelle.

### **Tarea 2: Inclusión de los términos referidos a nodos en el sistema lineal.**

#### INTERFAZ DE LA FUNCIÓN:

```
void Point::AddToLinearSystem(CGSolver *solver);
```

#### REQUISITOS:

1. Para un nodo, calcular el bloque de la matriz de masas referido a ese nodo, e incluirlo en la matriz del sistema lineal.
2. Para un nodo, calcular el término del lado derecho de la ecuación lineal, e incluirlo en el vector del sistema lineal.

#### OTROS DATOS:

- Al llamar a este método ya se ha calculado la fuerza total sobre el nodo.
- El paso de integración es un miembro de la clase Scene, Scene::step.
- El índice 'simIndex' indica la posición en el sistema lineal de los nodos que no están fijos.

### **Tarea 3: Inclusión de los términos referidos a muelles en el sistema lineal.**

#### INTERFAZ DE LA FUNCIÓN:

```
void Spring::AddToLinearSystem(CGSolver *solver);
```

#### REQUISITOS:

1. Para un muelle, calcular los términos de la matriz de rigidez implicados.
2. Para un muelle, incluir los términos de la matriz de rigidez en el sistema lineal, teniendo en cuenta si están en la diagonal o no.

#### OTROS DATOS:

- Al llamar a este método ya se ha calculado la longitud del muelle.
- El paso de integración es un miembro de la clase Scene, Scene::step. Recordar que la matriz de rigidez se ha de multiplicar por  $h^2$ .
- Un muelle define, en principio, 4 bloques de la matriz de rigidez (si ninguno de los nodos está fijo). En la parte diagonal, la posición de los bloques viene determinada por Point::simIndex. En la parte fuera de la diagonal, que es dispersa, la posición de un bloque viene determinada por Spring::simIndex. Recordar que CGSolver aprovecha que la matriz de rigidez es simétrica, y los términos de fuera de la diagonal sólo han de añadirse una vez, no dos.