Práctica 1 – Parte 1: Masa-Muelle y ODEs

(Asignación 5 de Febrero; Entrega 18 de Febrero a las 23:59)

¿Cómo entregar la práctica?

Enviar una copia del fichero Exercise.cpp por email a miguel.otaduy@urjc.es, antes del 18 de Febrero a las 23:59.

Especificaciones Generales:

Se proporciona un proyecto sobre MS VisualStudio 2010 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 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 el cálculo de un paso de simulación en dos escenas diferentes, y con varios parámetros de entrada. Las rutinas para la ejecución de cada paso de simulación están incluidas en el fichero Exercise.cpp. Todo nuevo código ha de ser añadido en el fichero Exercise.cpp. No se han de enviar más ficheros, **sólo** Exercise.cpp. La práctica está preparada para que no necesite librerías adicionales. Todo el código adicional que se precise ha de estar incluido en el fichero Exercise.cpp. La práctica se evaluará intercambiando el fichero Exercise.cpp, compilando y ejecutando, con lo cual no compilará si se precisan otros ficheros.

Las escenas están simuladas en el plano XY, y la gravedad ha de aplicarse en la dirección –Y (hacia abajo). Las fuerzas de amortiguamiento se deberán programar como amortiguamiento en los nodos (proporcionales a su velocidad)

Las escenas tienen múltiples parámetros modificables: propiedades mecánicas como masa, rigidez y amortiguamiento; paso de tiempo; método de integración... Estos parámetros se pueden modificar en la inicialización de la escena en Scene.cpp, o se pueden introducir por comando (probar a hacer AAPractica1a.exe –help para ver las opciones).¡Se recomienda probar las simulaciones con parámetros distintos para analizar el comportamiento!

Recursos Adicionales:

Se proporcionan clases de C++ para vectores y matrices en 2D, que deberían ser útiles para las escenas con simulación en 2D. Estas clases implementan operaciones básicas como suma, resta, multiplicación, producto escalar y vectorial, etc. Se encuentran en el fichero Vec2.h. Como se ha dicho antes, cualquier funcionalidad adicional ha de programarse en el fichero Exercise.cpp, ¡no se ha de modificar Vec2.h!

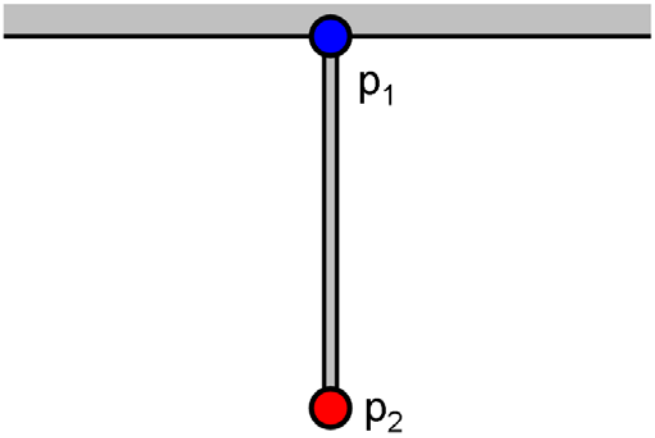
También se proporcionan clases (MatrixMN, Vector) y métodos (GaussElimination) para resolver sistemas de ecuaciones lineales por eliminación de Gauss. En el fichero Exercise.cpp se incluye un ejemplo de utilización de dicho código. Las clases y métodos se encuentran en el fichero LinSys.h. Esta funcionalidad sólo será necesaria para el problema 2.

Problema 1: Nodo con masa colgando del techo.

Este ejercicio consiste en la simulación de un muelle en 1 dimensión. Un extremo del muelle se encuentra fijado al techo, mientras que el otro cae por efecto de la gravedad. Además, el extremo inferior puede llegar a chocar con el suelo, en cuyo caso hay que aplicar una fuerza de contacto.

Los parámetros como masa (m), rigidez (k), amortiguamiento (d), rigidez de la colisión (kc), paso de simulación (dt), y longitud inicial del muelle (L) se pasan como argumentos.

La colisión con el suelo sólo deberá estar activa si así lo indica el flag 'collision'. Se activa/desactiva con la tecla 'C'.



INTERFAZ DE LA FUNCIÓN:

```
void AdvanceTimeStep1(float k, float m, float d, float L, float kc, float dt,
     int meth, float p1, float v1, float& p2, float& v2, float& p2old, bool collision);
```

REQUISITOS: Programar la actualización en cada paso de simulación de la posición p2 y la velocidad v2 del nodo inferior, para los siguientes métodos de integración: Euler Explícito (meth = 1), Euler Simpléctico (meth = 2), Punto Medio (meth = 3), Euler Implícito (meth = 4), Verlet (meth = 5). Escribir la posición y velocidad actualizadas en p2 y v2.

Nota: la posición del suelo se encuentra en Y=0.0. La fuerza de contacto se aplicará como una fuerza de penalti con rigidez kc, sólo si está activado el flag 'collision'.

Nota 2: la posición y velocidad anterior se pasan en los valores de entrada de p2 y v2-

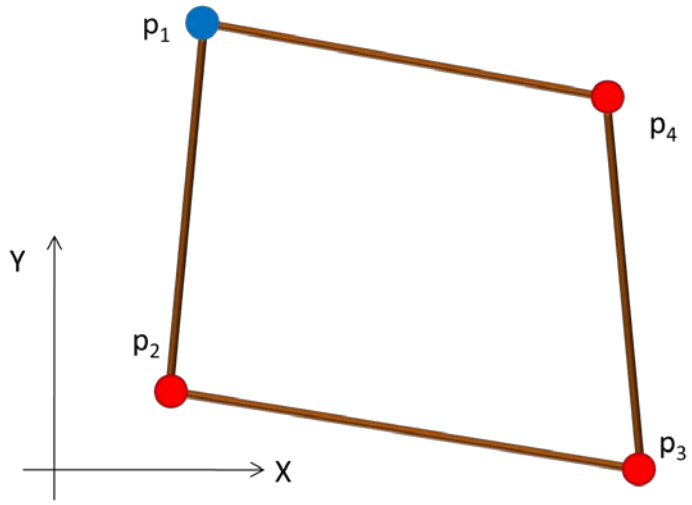
Nota 3: para el método de Verlet, la posición del paso previo (t – dt) se pasa en p2old, y también se ha de devolver el valor de p2old actualizado.

Problema 2: Cuadrado con conservación de área.

Este ejercicio consiste en simular el movimiento de un cuadrado. Cada arista está modelada como un muelle, el extremo superior izquierdo está fijado, y el cuadrado ha de conservar su área. Los parámetros de los nodos y los muelles (masa, etc.) se pasan como argumentos de forma similar al Problema 1.

Además, se pasan los siguientes argumentos para la conservación de área: área de reposo (A) y la rigidez de la energía (kA).

Se pasan dos flags, 'springs' y 'area' para determinar si se usan las fuerzas de muelles y la fuerza de conservación de área o no. Esto permite testear las fuerzas de muelles y la conservación de área por separado. Estas opciones se activan/desactivan con las teclas 'S' y 'C'.



INTERFAZ DE LA FUNCIÓN:

```
void AdvanceTimeStep2(float k, float m, float d, float L, float kA, float A, float dt,
      int method, const Vec2& p1, const Vec2& v1, Vec2& p2, Vec2& v2,
      Vec2& p3, Vec2& v3, Vec& p4, Vec& v4, bool springs, bool area);
```

REQUISITOS: Programar la actualización en cada paso de simulación de las posiciones y velocidades (p1, v1, p2, v2, p3, v3) de los nodos, para los métodos Euler Simpléctico (method = 2) y Euler Implícito (method = 4).

Nota: El área del cuadrado se ha de preservar usando dos triángulos (p1,p2,p3) y (p3,p4,p1), cada uno de ellos con la mitad del área total.