Relazione del progetto d'esame

Francesca Fusco Riccardo Antonio Larocca

Febbraio 2024

Contents

0 Aggiornamento

Le modifiche effettuate al programma sono le seguenti:

1. tutti i $vector2f$ della libreria $SFML$ sono stati cambiati in $vector2 < double >$, di conseguenza tutte le variabili $float$ sono state cambiate in $double$;
2. le variabili globali sono state dichiarate locali;
3. alcune $free\ function$ sono state rese metodi;
4. sostituzione dei vettori contenenti le energie con funzioni membro della classe *chain* che ne calcolano il valore;
5. rimozione del metodo $chain :: initial\_config$. Il setting iniziale è implementato con il costruttore;
6. rivisti i parametri passati alle funzioni by value e by reference;
7. rimosso l'uso di "$alias$" per il valore di π;
8. Suggerimento: si consigliano valori in input della costante elastica k e della massa m tali che l'ordine di grandezza di $k/m < 10^5$, per evitare comportamenti divergenti.

1 Introduzione al modello implementato

L'obiettivo del progetto è la simulazione del comportamento di una corda elastica circolare, costruita su un piano che viene messo in rotazione con velocità angolare ω attorno all'asse y su cui giace il diametro della corda. I punti che intercettano l'asse y, detti polo nord e polo sud, saranno vincolati a scorrere lungo quest'asse.

Una volta in moto, su ogni elemento della corda sarà applicata una forza centrifuga, contrapposta alla forza elastica che tiene coesi i punti alla corda. Per via della forza centrifuga la corda andrà ad allungarsi, spostando i poli ovest ed est (ossia l'estremo sinistro e destro rispettivamente) lungo l'asse x. Qualora la velocità angolare diminuisse, quindi la rotazione rallentasse o cessasse, la corda si restringerà.

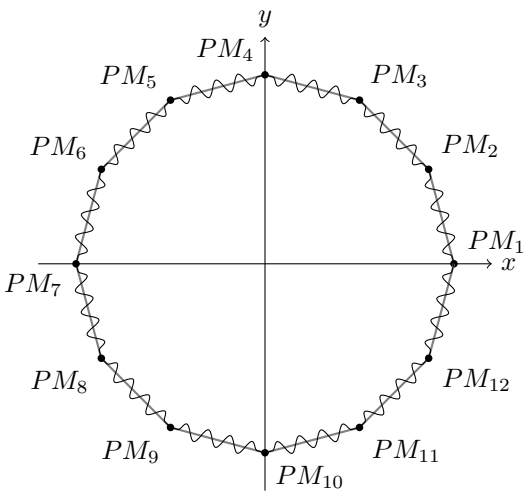

Figure 1: Implementazione della corda

La corda è stata implementata come un'insieme di punti materiali (in figura "PM") di uguale massa, collegati da molle di uguale lunghezza a riposo e costante elastica.

Il codice si propone di calcolare e rappresentare, tramite la libreria *SFML*, la posizione dei punti rispetto al sistema di riferimento collocato sul piano di rotazione.

Sistema fisico: calcolo delle energie e delle forze in gioco

Le forze agenti su un punto materiale sono quindi la forza elastica, agente tra punti materiali adiacenti, e la forza centrifuga lungo l'asse x:

$$\overrightarrow{F}_{el} = k \cdot \| \overrightarrow{r} - \overrightarrow{l}_v \|$$
$$\overrightarrow{F}_{cf} = m\omega^2 \overrightarrow{x}$$

dove con $\overrightarrow{r}$ si intende il vettore che collega due punti successivi della catena e con $\overrightarrow{x}$ si intende la posizione del punto.

Sia PM_j l'elemento j-esimo della catena, questo sarà soggetto a una forza data da

$$\overrightarrow{F} = \overrightarrow{F}_{el}^{j+1} + \overrightarrow{F}_{el}^{j-1} + \overrightarrow{F}_{cf}^{j}$$

dove $\overrightarrow{F}_{el}^{j-1}$ e $\overrightarrow{F}_{el}^{j+1}$ sono le forze elastiche agenti sul punto j esercitate dai punti $j-1$ e $j+1$ e $\overrightarrow{F}_{cf}$ è la forza centrifuga a cui è soggetto l'elemento j-esimo.

Nel codice il calcolo delle forze sarà svolto dalle funzioni "*apply_hooke*", e "*apply_CF*". Le forze $\overrightarrow{F}_{el}^{j-1}$ saranno indicate con *f_prev*, e le forze $\overrightarrow{F}_{el}^{j+1}$ con f.

L'energia del sistema invece è calcolata come somma delle energie dei punti costituenti del sistema:

$$E = \sum_{j}^{N_{pm}} E_j$$

Ciascun punto possiede una energia pari alla somma di energia cinetica e potenziale elastica.

$$E_j = K_j + U_j$$

dove l'energia cinetica e potenziale di ciascun punto sono date da, rispettivamente,

$$K_j = \frac{1}{2} \| \overrightarrow{v}_j \|^2 m$$

$$U_j = \frac{1}{2} k \| \overrightarrow{r}_j - \overrightarrow{l}_v \|^2$$

Nota: Nel codice sarebbe stato possibile implementare, per esempio, anche la forza peso, ma poiché si è notato che influenzasse poco la dinamica dell'evoluzione è stata esclusa dall'integrazione.

2 Struttura del progetto

Il progetto è suddiviso in più file, un *main.cpp* e quattro *translation unit*, costituite dai seguenti *header* e *source* file:

1. **vec.hpp** e **vec.cpp**: si implementa il concetto di spazio vettoriale attraverso la definizione di una classe *vec*, che ha come unico membro un vettore bidimensionale di *float* della libreria di *SFML*;

2. **pm.hpp** e **pm.cpp**: si implementa la classe PM per il punto materiale come elemento massivo della corda. I suoi membri sono due variabili di tipo *vec* rappresentanti la posizione e la velocità e una di tipo *float* per la massa;

3. **hooke.hpp** e **hooke.cpp**: si implementa una classe $Hooke$ per la molla che collega i punti. I suoi membri sono due variabili di tipo *float* per la costante elastica e la lunghezza a riposo, e una di tipo *vec* che rappresenta la "lunghezza a riposo vettoriale", il vettore uscente da un punto e indirizzato al punto successivo (fig. 2), il cui modulo è la lunghezza a riposo.

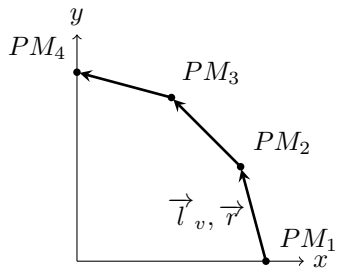

Figure 2: vettore $\overrightarrow{r}$ e lunghezza a riposo vettoriale

4. **chain.hpp** e **chain.cpp**: si implementa la classe della corda, *Chain*, come insieme di punti collegati da molle in una configurazione iniziale che andrà a evolversi. I suoi membri sono una variabile di tipo *std::vector*, che contiene tutti i punti materiali, e una di tipo *Hooke*. Inoltre fuori dalla classe sono presenti gli *std::vector* *Kinetic_energies* e *Potential_energies*, contenenti le energie cinetiche e potenziali dei singoli elementi della catena, con cui in seguito sarà ricavata l'energia totale.

3 Funzionamento del codice

Nel presente paragrafo si illustrano le *free function* e i principali metodi delle classi appena presentate, dichiarate nei rispettivi header file e definite nei source file.

vec.hpp e vec.cpp

Sono implementate le funzioni *get* e *update* che permettono di accedere alle coordinate del vettore e aggiornarne il valore, le funzioni che permettono i calcoli tra vettori e scalari, e gli operatori che torneranno utili nella scrittura del codice.

pm.hpp e pm.cpp

Sono implementate le funzioni *get* e *update* che permettono di accedere ai membri della classe, quali coordinate spaziali, velocità e massa, e di aggiornarne il valore. Con i costruttori oltre a inizializzare i valori dei membri della classe, sono creati anche i punti per l'implementazione grafica.

hooke.hpp e hooke.cpp

Sono implementate le funzioni *get* e *update* che permettono di accedere ai membri della classe.

chain.hpp e chain.cpp

Nei file *chain.hpp* e *chain.cpp* sono implementate delle *free function* che calcolano le forze:

1. *vec apply_hooke*: prende in argomento due punti e una molla, e restituisce la forza elastica esercitata dal secondo punto sul primo, calcolata come

$$\vec{F}_{hooke} = k\left(\|\vec{r}\|\hat{r} - l\cdot\frac{\vec{r}}{|\vec{r}|}\right) = k\left(\|\vec{r}\| - l\right)\hat{r}$$

   dove $\vec{r}$ è il vettore che esce dal primo punto e "punta" il secondo punto e l la lunghezza a riposo della molla;

2. *vec apply_CF*: prende come argomento un punto e la velocità angolare, restituisce la forza centrifuga dovuta alla velocità angolare agente sul punto. Ha componente solo lungo l'asse x;

Le principali *member function* servono alla costruzione della configurazione iniziale della catena e alla sua evoluzione, e sono

1. *PM solve*: aggiorna la posizione e la velocità di un elemento della catena dati il punto, la forza totale agente su di esso e un intervallo di tempo;

2. *void initial_config*: calcola la posizione iniziale di ogni punto dati il raggio r della catena, la massa dei punti, un angolo θ - calcolato opportunamente in base al numero di punti in input - e il numero di punti N_{pm} costituenti della catena. La coordinata angolare di ciascun punto è ottenuta moltiplicando

$$\theta = \frac{2\pi}{N_{pm}} \tag{1}$$

   per un indice i assegnato a ciascun punto, in ordine, a partire dal primo $i = 0$ (fig 3).

$$\theta_i = \theta \cdot i$$

   Chiamata tale funzione, la catena sarà costruita in forma circolare, ferma nel sistema di riferimento del piano su cui giace.

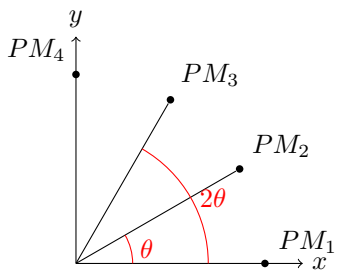

Figure 3: Angoli nella posizione di riposo

3. *void evolve*: preso come argomento un intervallo di tempo *dt*, comincia il calcolo delle energie e dello stato dei punti in ogni istante, chiamando le funzioni *f_prev* e *f* che calcolano le forze agenti su un punto, a partire dal primo.

   Prima di tutto viene creata una copia della catena, *std::vector ch_copy*, affinchè conservi immutato lo stato della catena in quel determinato istante. Se non fosse creata la copia della catena, le posizioni dei punti sarebbero aggiornate in ordine, e il calcolo delle forze elastiche esercitate tra di essi avverrebbe prima che le posizioni di tutti i punti siano aggiornate allo stesso istante.

   Nel calcolo delle forze dunque, verranno passati in argomento alle funzioni gli elementi della **copia della catena** e verranno aggiornate le posizioni e velocità degli elementi della **catena**. In questo modo le forze

f_prev, calcolate alla fine di ogni ciclo, saranno calcolate sulla base delle coordinate aggiornate di tutti i punti nello stesso momento.

Il calcolo delle coordinate dei punti viene compiuto attraverso dei cicli *for*. Prima di entrarvi però, viene calcolata la forza *f_prev* applicata dall'ultimo punto sul primo, per imporre la condizione di circolarità della catena.

Entrando nel ciclo, è calcolata la forza *f* applicata dal secondo punto sul primo della copia della catena. Nel codice sono indicati i punti della copia che applicano la forza, ovvero il precedente e il successivo, con, rispettivamente, **state_last_copy* e **state_it_next_copy* (iteratori dereferenziati), e il punto considerato con **state_it_copy*.

Successivamente è chiamata la funzione *solve*, che risolve le equazioni del moto del primo punto della copia, data la forza appena calcolata, e ritorna lo stato da assegnare al primo punto della catena, indicato con **state_it*.

Contemporaneamente avviene il calcolo dell'energia cinetica e potenziale di ciascun punto fino al penultimo in quell'istante, contenute nei *std::vector Kinetic_energies* e *Potential_energies*.

Prima che il ciclo si concluda, è calcolata la forza che il primo punto applica sul secondo, che sarà assegnata alla variabile *f_prev*, pronta per il ciclo successivo.

Il ciclo si ripete per ogni punto della catena fino al penultimo. Una volta concluso il ciclo sono risolte le equazioni del moto per l'ultimo punto in maniera analoga al primo punto, imponendo la circolarità della catena, e sono calcolate le sue energie cinetiche e potenziali.

Per via della scarsa precisione dei *float*, dopo un certo intervallo di tempo la catena non sarebbe risultata più perfettamente simmetrica all'asse di rotazione, portando quindi la forza centrifuga a svincolarla dall'asse e allontanarla. Ragion per cui si è ritenuto necessario imporre a ogni ciclo che il polo nord e il polo sud della catena siano vincolati all'asse di rotazione.

L'operazione di integrazione è ripetuta due volte: la prima volta le equazioni del moto sono risolte a partire dal primo punto per concludere con l'ultimo, quindi in senso **antiorario**. La seconda volta sono risolte in senso **orario**. Questo espediente è stato utilizzato per compensare un'anomalia osservata nel corso della costruzione del codice: integrando solo in senso antiorario la posizione dei punti della catena risultava sfasata verso destra già dal primo istante. Integrando anche in senso orario prima di passare all'istante di tempo successivo, la posizione è sfasata anche verso sinistra, annullando ogni traslazione.

4 Main file: input e output

Nel file *main.cpp* sono presenti le seguenti free function:

1. *void evolve*: passati come argomenti un intervallo di tempo *dt*, un numero intero chiamato *steps_per_evolution* e la catena, fa evolvere lo stato della catena nel tempo. Questa funzione chiama la funzione *chain::evolve* passandole come argomento l'intervallo di tempo *dt* e successivamente stampa a schermo la catena.

   Il numero *steps_per_evolution* è dato dal rapporto N/fps dove N è numero intero e *fps* sta per *frames per second*. In questo modo lo stato della corda non sarà disegnato a ogni *dt*, poichè l'operazione richiederebbe molto tempo. Piuttosto sarà disegnato un numero di volte ragionevole affinchè la simulazione sia continua e arbitrariamente veloce. Per prendere dimestichezza con questo concetto e osservarne la sua utilità, si noti che all'aumentare degli *steps_per_evolution*, aumenta il numero di evoluzioni calcolate prima che il nuovo stato della corda sia stampato a schermo. Ciò comporta che sull'interfaccia grafica sarà osservato, nello stesso intervallo di tempo reale, un'evoluzione della catena più veloce, come se il tempo fosse velocizzato, a discapito della fluidità.

2. *std::string to_string_with_precision*: dato un *float* e un numero intero n, la funzione converte il *float* in *std::string* con n cifre decimali. Si rimanda alla [Fonte]. Serve per poter stampare a schermo i valori di alcuni parametri e dell'energia.

All'utente sono chiesti in input i parametri principali quali la massa dei punti materiali **m**, la costante elastica della molla **k**, il numero di punti materiali **NoPM**, il raggio della configurazione iniziale **r** e la velocità angolare **w**.

Nota: Si consiglia un valore di $r \in [200, 300]$ e di $w = 2$. Inoltre, si consiglia di non superare il valore di $k = 10^6$ (per valori di $m \sim 10$), poichè il comportamento potrebbe divergere.

É stata inoltre data all'utente la possibilità di modificare i parametri quali **w** e **Steps_per_evolution** in tempo reale con i tasti

1. W: aumenta di 1 la velocità angolare;
2. S: diminuisce di 1 la velocità angolare;
3. D: aumenta di 1 steps_per_evolution;
4. A: diminuisce di 1 steps_per_evolution.

Una volta eseguito il codice verrà aperta una finestra di *SFML* sulla quale verrà visualizzata la catena nella posizione iniziale e i valori di alcuni parametri.

5 Risultati e unit testing

Il comportamento della catena sembra rispecchiare solamente in parte il comportamento reale di una catena elastica. Questo è dovuto al fatto che una catena elastica reale ha una certa *rigidità*: man mano che si piega un elemento di lunghezza della catena, risulterebbe sempre più difficile piegarlo ulteriormente. Inoltre una catena reale non può compenetrarsi.

Infine una catena elastica reale ha una lunghezza di estensione massima finita, a differenza di quella implementata in questo progetto che può estendersi indefinitamente.

Tuttavia, se l'obiettivo fosse stato quello di simulare una corda ideale senza le condizioni sopracitate, la simulazione sarebbe realistica.

L'unità di testing ha confermato che lo spazio vettoriale è stato implementato correttamente, rispecchiando le regole di uno spazio vettoriale matematico, e che le forze sono calcolate nel modo corretto.

6 Compilazione

Su Windows si suggeriscono i seguenti comandi: Per compilare il codice

```
$ g++ vec.cpp pm.cpp hooke.cpp chain.cpp main.cpp -lsfml-graphics
-lsfml-window -lsfml-system -Wall -Wextra -fsanitize=address -o main
```

Per eseguirlo

```
$ ./main
```

Per i test:

```
$ g++ vec.cpp pm.cpp hooke.cpp chain.cpp tests.cpp -lsfml-graphics
-lsfml-window -lsfml-system -o tests
$ ./tests
```

NOTA. La compilazione con -Wall -Wextra -fsanitize=address non riporta alcun errore. Tuttavia a fine esecuzione vengono segnalati alcuni errori ai quali però non si è riusciti a dare una spiegazione.

NOTA. Per la realizzazione del progetto è stata usato ***Github***, caricando il progetto alla seguente Repository.