Simulatore del movimento dei pianeti in un sistema solare

Gioele Mancino, Federico Mastroforti, Luca Tesei, Nico Tortorici

7 luglio 2023

1 Il Progetto

1.1 Istruzioni per la compilazione

Eseguire il pull della repository con il comando:

```
$ git clone https://github.com/NexganGH/gravity_simulator
```

Installare SFML v2.5.1 con il comando:

```
$ sudo apt-get install libsfml-dev=2.5.1
```

Installare TGUI, libreria estensione di SFML per create le interfacce utenti:

```
$ sudo add-apt-repository ppa:texus/tgui
$ sudo apt-get update
$ sudo apt-get install libtgui-1.0-dev
```

Il programma può essere compilato, *per la release*, con:

```
$ cmake -S. -B build -DBUILD_TESTING=Off
$ cmake --build build
$ build/gravity
```

Questo creerà un eseguibile `gravity` nella cartella `build`, da eseguire con il comando:

```
$ build/gravity
```

1.2 Obiettivo del simulatore

Lo scopo del programma quello di simulare un sistema di N corpi planetari o stellari secondo la fisica newtoniana. L'obiettivo principale era quello di creare un sistema solare stabile, in grado cioè di preservare le orbite nel tempo, senza che queste degenerino.

Oltre all'obiettivo principale, si sono volute implementare altre funzionalità utili all'utente, tra cui:

- finestra grafica per visualizzare i corpi, con relativa interfaccia grafica;
- possibilità di inserire nuovi corpi con una massa e direzione a scelta;
- possibilità di avviare, interrompere o ripristinare la simulazione;
- possibilità di scegliere tra diverse configurationi iniziali.

1.3 Interface

1.3.1 Elements on screen

Nell'angolo in alto a sinistra (Figura) troviamo 2 pulsanti cliccabili (che vedremo nel dettaglio nella prossima sezione), il tempo trascorso dall'inizio della simulazione (in anni, espresso con 7 cifre significative) e la velocità espressa in rapporto di tempo (anni al secondo)

1.3.2 User interaction

Nell'angolo alto a sinistra dell'interfaccia grafica sono presenti due tasti:
-il tasto Play/Pause (Figura , sinistra) che permette di far partire o fermare la simulazione;
-il tasto Reset (Figura , destra) che riporta tutto allo stato iniziale (i corpi già presenti ritornano nel punto in cui erano a inizio simulazione mentre quelli generati dall'utente vengono eliminati);
In tutta la finestra grafica è possibile cliccare con il tasto destro in un punto, facendo ciò comparirà una finestra in cui è possibile inserire la massa del corpo che stiamo inserendo (in unità di massa terrestre) e sarà poi possibile selezionare il verso di movimento iniziale del corpo

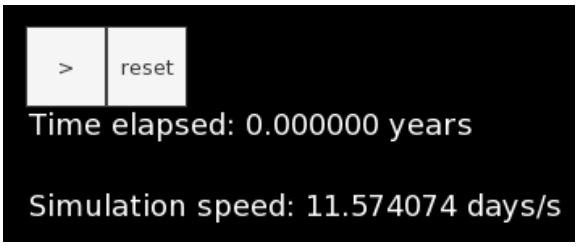

Figura 1: Il tasto (sinistra) Play/Pause permette di far partire e fermare la simulazione in ogni momento, mentre il tasto (destra) reset riporta la simulazione allo stato iniziale (i corpi già presenti ritornano nel punto in cui erano a inizio simulazione mentre quelli generati dall'utente vengono eliminati)

1.4 La fisica dietro al programma

La legge di gravitazione universale afferma che due punti materiali si attraggono con una forza di intensità direttamente proporzionale al prodotto delle masse dei singoli corpi e inversamente proporzionale al quadrato della loro distanza. Questa legge, espressa vettorialmente, diventa:

$$\mathbf{F}_{2,1}(\mathbf{r}) = -\frac{Gm_1m_2}{r^2}\hat{\mathbf{u}}_r \tag{1}$$

dove $\mathbf{F}_{2,1}$ è la forza con cui l'oggetto 1 è attratto dall'oggetto 2, G è la costante di gravitazione universale, che vale circa $6,67 \cdot 10^{-11} \frac{Nm^2}{kg^2}$, m_1 e m_2 sono le masse dei due corpi, $\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2$ è il vettore congiungente i due corpi (supposti puntiformi), r è il suo modulo e $\hat{\mathbf{u}}_r = \frac{\mathbf{r}}{r}$ rappresenta il versore che individua la retta congiungente i due punti materiali.
Il programma utilizza questa relazione per calcolare istante per istante le forze che agiscono su ogni corpo, e di conseguenza le accelerazioni:

$$\mathbf{F}_{j,tot} = \sum_{i=1, i\neq j}^{N_{bodies}} \mathbf{F}_i(\mathbf{r}) = m\ddot{\mathbf{r}} \tag{2}$$

dove $\mathbf{F}_{j,tot}$ è la forza risultante che agisce sul corpo j.
Integrando poi la precedente equazione si ottiene la legge oraria $\mathbf{r}(t)$ del corpo j. Non essendo risolvibile analiticamente si deve ricorrere a metodi di integrazione numerica.

2 Soluzioni matematiche

2.1 Metodo LeapFrog

3 Implementazione

3.1 Strumenti utilizzati

È stato utilizzato C++ 17, compilator tramite GCC.

Si sono utilizzate le librerie:

- SFML 2.5.1 per l'implementazione grafica.
- TGUI 1.0.

Per il linking dei file e la creazione degli executable, come i test, è stato usato CMake 3.16.

Per lo sviluppo è stato utilizzato l'IDE Visual Studio Code, usato in ambiente WSL con Ubuntu 20.04. È stato adottato, per la formattazione, un file .clang-format basato sullo stile di Google.

Per effettuare il controllo delle versioni, è stato utilizzato Git 2.5.1; come repository remota è stato adottato GitHub. La repository è presente al link: `https://github.com/NexganGH/gravity_simulator/tree/master`

3.2 Struttura dei file

La directory principale contiene è divisa nelle seguenti sotto-directory:

- `docs/` contiene questo file, il suo sorgente in Latex e altri file necessari alla compilazione.
- `include/` contiene gli header (`.hpp`).
- `src/` contiene i file `.cpp` con le definizioni.
- `test/` contiene i file di testing.

3.3 Struttura generale delle classi

L'obiettivo di design del progetto è stato quello di sfruttare al massimo la OOP per garantire la massima riusabilità delle classi, in egual modo aumentando la mantenibilità e scalabilità del codice.

La classe cardine del programma è `SimulationState`, la quale rappresenta lo stato corrente della simulazione. Essa contiene:

- la lista dei corpi presenti nello spazio (vettore di `Body`);
- un'istanza di `Configuration`, che rappresenta la configurazione iniziale attualmente in uso;
- un'istanza di `PhysicsEngine`, classe che si occupa di effettuare i calcoli circa le posizioni dei pianeti;
- un'istanza di `Renderer`, il cui scopo è disegnare a schermo i pianeti.

`Body` è una classe che rappresenta un corpo nello spazio, dotato di una posizione, velocità, forza e massa. Tale classe è astratta poiché alcuni metodi dipendono dal tipo del corpo, distinzione eseguita tramite ereditarietà (attualmente le classi figlie sono `Planet` e `Star`).

`Configuration` rappresenta uno specifico stato iniziale della simulazione: è quindi caratterizzato da una lista di corpi con le loro posizioni iniziali. A tal propositio, la funzione `getConfigurations` fornisce una lista di configurazioni, le quali sono mostrate all'utente in alto a sinistra nell'interfaccia.

`PhysicsEngine` è la classe che esegue i calcoli: il suo metodo pubblico più importante è `evolve`, il quale permette di evolvere una lista di corpi nell'arco di tempo `dt`. Esso implementa il metodo di integrazione LeapFrog.

`Renderer` è la classe che permette di disegnare a schermo i corpi e i componenti dell'interfaccia. Nel caso dei corpi, è effettuata un automatico una conversione dalle coordinate dei corpi, espressi in coordinate reali (metri), alle coordinate sullo schermo (pixel). Tale conversiene avviene, ovviamente, in base alla larghezza dell'universo in visione (variabile a seconda della configurazione).

`GuiManager` è una classe che crea e gestisce l'interfaccia utente.

3.4 PhysicsEngine

Il metodo principale, `evolve`, oltre ha come parametro la lista dei corpi da evolvere e l'intervallo di tempo `dt`. Tale intervallo deve corrispondere all'intervallo di tempo che è trascorso *nella realtà* dalla scorsa chiamata di `evolve`.

Al fine di simulare eventi che avvengono in periodi estesi, come anni, è necessario però utilizzare una `timescale`, ovvero un valore che velocizzi la simulazione. Tale valore è moltiplicato per il `dt` del metodo evolve. Si incorre nella seguente relazione:

$$T_{simulazione} = n \cdot dt \cdot timescale$$

dove $T_{simulazione}$ è il tempo trascorso, in secondi, nella simulazione durante n chiamate di `evolve` di un intervallo di tempo dt ciascuno.

Se $n \cdot dt = 1s$, cioè è passato 1 secondo nella realtà, si ha che $T_{simulazione} = timescale$; da ciò si conclude che timescale, espresso in secondi, rappresenta il numero di secondi passato nella simulazione durante un secondo nella realtà.

Si noti inoltre che, siccome dt dipende dalla velocità con cui viene eseguito il ciclo, quindi dall'hardware dell'utente, il tempo della simulazione è normalizzato: *la velocità della simulazione è costante in tutti gli ambienti/dispositivi.*

Questo implica il fatto, però, che un utente con un hardware lento, a parità di timescale, avrà degli intervalli di calcolo maggiori, diminuendo l'accuratezza dei calcoli. A tal proposito può essere ridotto il timescale.

3.5 Renderer

La classe `Renderer` ha lo scopo di disegnare i corpi e l'interfaccia a schermo. Poiché le posizioni dei corpi sono espressi in coordinate reali (metri), è necessario che prima di disegnarli a schermo le loro coordinate sono convertite. A tal proposito, il metodo statico `fromUniverseWidth` permette di calcolare una scala a partire dalla grandezza dell'universo e della finestra:

$$scale = \frac{universe\ width}{window\ width}$$

Per convertire una coordinata `x` da coordinate reali a coordinate sullo schermo, si usa la relazione:

$$x_{schermo} = \frac{x_{reale}}{scale}$$

3.6 Configuration

La classe `Configuration` contiene lo stato iniziale del progetto

4 Testing e debugging

5 Risultati e problemi riscontrati

Si è riuscito a ottenere una simulazione dei primi quattro pianeti del sistema solare,. Le orbite, grazie all'implementazione del metodo Leapfrog

A Calcoli e approssimazioni