Linee guida per lo sviluppo di un semplice simulatore particellare[1]

Donato D'Ambrosio
Department of Mathematics and Computer Science
University of Calabria, Italy

15 gennaio 2019

[1]Corso di Informatica per i Corsi di Laurea in Chimica, Fisica e Scienze dei Materiali, Università della Calabria

1 Specifiche del progetto

Sviluppare un programma C++ che simuli l'evoluazione dinamica di un sistema di particelle 2D. Il dominio (cioè lo spazio 2D in cui si evolve il fenomeno) è rappresentato da un quadrato di lato $l = 0.04$ m. Ogni particella ha massa $m = 0.001$ kg e raggio $r = 10^{-4}$ m. Un totale di n (ad esempio $n = 20$ o $n = 1000$) particelle sono posizionate in maniera casuale all'interno del dominio in modo che non vi siano sovrapposizioni particella-particella e particella-bordo dominio. Oltre alla posizione, che indicheremo come $\boldsymbol{p} = (p_x, p_y)$, lo stato di ogni particella è caratterizzato da una velocità $\boldsymbol{v} = (v_x, v_y)$ e da una forza $\boldsymbol{F} = (F_x, F_y)$. Ai fini della simulazione, la velocità iniziale di ogni particella deve essere impostata a zero.

Le particelle sono soggette alla forza di gravità, $m\boldsymbol{g}$ (che agisce lungo la direzione y), e alla forza di contatto risultante da eventuali urti partiella-particella e/o paricella-bordo dominio, $\boldsymbol{f_c}$. Se con n_c indichiamo il numero di contatti, per ogni particella si ha:

$$m\boldsymbol{a} = m\boldsymbol{g} + \sum_{j=1}^{n_c} \boldsymbol{f_{c,j}}$$

dove la forza di contatto è modellata tramite un sistema molla-smorzatore del tipo:

$$\boldsymbol{f_c} = -k_n\, \boldsymbol{\delta}_n - \eta_n\, \boldsymbol{v}_{rn} \tag{1}$$

In tale modello molla-smorzatore k_n ed η_n rappresentano rispettivamente la costante elastica e il coefficiente di smorzamento (che sono caratteristici del materiale). Ai fini della simulazione è possibile utilizzare i seguenti valori: $k_n = 5 \cdot 10^4$ e $\eta_n = 1.5 \cdot 10^3$. $\boldsymbol{\delta}_n$ rappresenta il livello di sovrapposizione. Nel caso di urto paricella-particella, esso è espresso come

$$\boldsymbol{\delta}_n = \big((r_i + r_j) - \|\boldsymbol{p}_i - \boldsymbol{p}_j\|\big)\, \boldsymbol{n}_{ij}$$

essendo $\boldsymbol{n}_{ij}$ il vettore normale che definisce la direzione tra il centro della particella i e il centro della particella j (vedi Figura 1). Nel caso di urto particella-bordo, che indicheremo con $\boldsymbol{b}$, la precedente espressione si riduce a:

$$\boldsymbol{\delta}_n = (r_i - \|\boldsymbol{p}_i - \boldsymbol{b}\|)\, \boldsymbol{n}_{ij}$$

La velocità relativa, $\boldsymbol{v}_{rn}$, è valutata esclusivamente lungo la direzione che congiuge i centri delle particelle ed è data dalla relazione:

$$\boldsymbol{v}_{rn} = (\boldsymbol{v}_i - \boldsymbol{v}_j) \cdot \boldsymbol{n}_{ij}\ \boldsymbol{n}_{ij}$$

La Figura 1 mostra un esempio di collisione particella-particella.

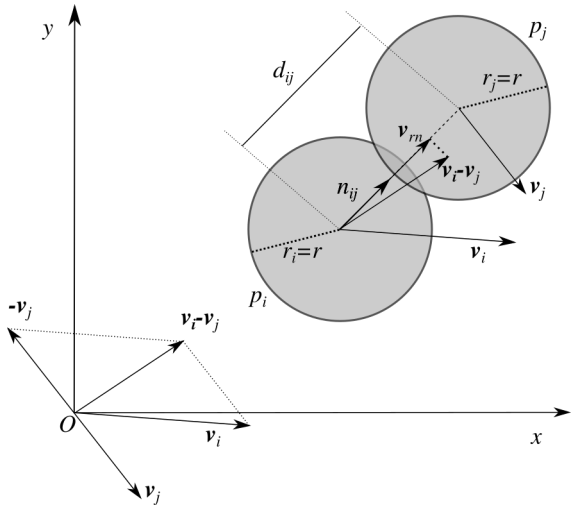

FIGURA 1: Esempio di collisione tra due particelle di forma circolare, p_i e p_j di raggio r. La collisione si verifica quando la distanza tra i centri delle particelle, d_{ij}, è minore di $r_i + r_j = 2r$. In tal caso, è applicata una forza repulsiva definita com in Equazione 1 ad entrambe le particelle lungo la direzione che congiunge i centri delle particelle stesse (detta direzione normale), $\boldsymbol{n}_{ij}$.

Nel caso di urto particella-bordo, la velocità relativa è semplicemente data dalla relazione seguente:

$$\boldsymbol{v}_{rn} = \boldsymbol{v}_i \cdot \boldsymbol{n}_{ib}\, \boldsymbol{n}_{ib}$$

dove $\boldsymbol{n}_{ib}$ rappresenta la direzione perpendicolare al bordo.

Durante la simulazione, il tempo corrispondente a un passo di calcolo deve essere sufficientemente piccolo per catturare diversi momenti dell'urto, il quale avviene generalmente in un tempo molto piccolo che indicheremo con τ. Se si vogliono catturare N_t momenti dell'urto, si può utilizzare la seguente formula empirica per ricavare il time-step Δt:

$$\Delta t = \frac{\tau}{N_t} = \frac{\pi\sqrt{m/k_n}}{N_t} \tag{2}$$

dove N_t varia tipicamente tra 10 e 50.

Una volta determinata la forza agente su ogni particella e il tempo corrispondente al passo di calcolo, è possibile utilizzare le seguenti formula cinematiche approssimate per determinare nuova posizione e velocità di ogni particella.

Sia $\boldsymbol{a}_i = \boldsymbol{f}_i/m$ l'accelerazione cui è soggetta l'$i-esima$ particella, allora si ha:

$$\begin{aligned} \boldsymbol{v}_i' &= \boldsymbol{v}_i + \boldsymbol{a}_i \Delta t \\ \boldsymbol{x}_i' &= \boldsymbol{x}_i + \boldsymbol{v}_i' \Delta t \end{aligned} \tag{3}$$

2 Organizzazione dei file sorgenti

Quando si sviluppa un'applicazione di una certa complessità è buona norma tenere separate la fisica del problema dalla parte di codice che gesitisce la simulazione e l'I/O. Per tale motivo, convine utilizzare più di un file sorgente, facendo riferimento al programma `make` per automatizzare la compilazione del programma.

```
[donato@vega parSim2D]$ ll
total 32
-rw-r--r-- 1 donato users 934 Jan  7 13:09 Makefile
-rw-r--r-- 1 donato users 729 Jan  7 13:06 particle.cpp
-rw-r--r-- 1 donato users 218 Jan  7 13:06 particle.h
-rw-r--r-- 1 donato users 153 Jan  7 13:06 simulation.cpp
[donato@vega parSim2D]$ make
g++    -c -o particle.o particle.cpp
g++    -c -o simulation.o simulation.cpp
g++ particle.o simulation.o -lm -o parSim2D
[donato@vega parSim2D]$ ll
total 76
-rw-r--r-- 1 donato users   934 Jan  7 13:09 Makefile
-rwxr-xr-x 1 donato users 17744 Jan  7 13:21 parSim2D
-rw-r--r-- 1 donato users   729 Jan  7 13:06 particle.cpp
-rw-r--r-- 1 donato users   218 Jan  7 13:06 particle.h
-rw-r--r-- 1 donato users  5640 Jan  7 13:21 particle.o
-rw-r--r-- 1 donato users   153 Jan  7 13:06 simulation.cpp
-rw-r--r-- 1 donato users  2640 Jan  7 13:21 simulation.o
```

Listato 1: Esempio suddivisione dell'applicazione in più file sorgenti (.cpp) e header (.h). Si noti anche la presenza del file `Makefile` (il cui contenuto è mostrato nel Listato 2), utile per la compilazione del progetto attraverso il comando `make`.

Ad esempio, è possibile adottare uno schema come quello riportato nel Listato 1 in cui il file `particle.h` contiene le definizioni dei nuovi tipi di dato (ad esempio `struct`, `typedef` ed `enum`) utili a modellare il sistema di interesse. Il file `particle.cpp` contiene, invece, l'implementazione delle funzioni che operano sui dati (ad esempio per la definizione delle condizioni iniziali del sistema, delle condizioni al contorno - per caratterizzare il comportamento del sistema ai confini del dominio di simulazione -, per il calcolo delle forze agenti sulle particelle e per il calcolo della nuova posizione delle particelle stesse). Il file `simulation.cpp` dovrebbe contenere tutto ciò che ha a che fare con la simulazione tra cui l'oggetto necessario

per definire l'isieme di particelle. Nel nostro caso è possibile utilizzare un unico array di `Particelle`, essendo `Particelle` una `struct` definita nel file `simulation.cpp` con i campi necessari a caratterizzare una particella in termini di posizione, velocità e forza applicata. Infine, potrebbe essere una buona idea considerare un ulteriore file per le funzioni di I/O, cosa che però non è considerata in questa guida.

```
# definisce la macro CPPC
ifndef CPPC
  CPPC=g++
endif

# definisce la macro LIBS alla libreria matematica
LIBS = -lm

# definisce la macro EXEC al nome dell'eseguibile
EXEC = parSim2D

# definisce il target di default a $(EXEC)
# in caso di invocazione di make senza parametri
default: $(EXEC)

# esegue il linkaggio dei file oggetto
# generando l'eseguibile
# $^ rappresenta una macro predefinita di make che
# corrisponde a quello che compare prima del
# simbolo : ($(EXEC) nel caso specifico)
# S@ rappresenta la macro predefinita che corrisponde
# a quello che compare dopo il simbolo :
# (particle.o simulation.o nel caso specifico)
$(EXEC): particle.o simulation.o
  $(CPPC) $^ $(LIBS) -o $@

# genera il file oggetto del sorgente cpp
simulation: simulation.cpp
  $(CPPC) $^ -c

# genera il file oggetto del sorgente cpp
particle: particle.cpp
  $(CPPC) $^ -c
# elimina l'eseguibile e i file oggetto

clean:
  rm -f $(EXEC) *.o
```

LISTATO 2: Esempio di file di configurazione di `make` per la compilazione del progetto. La sintassi è descrtitta tramite commenti (righe che iniziano col simbolo #).

Una volta definiti i file sorgenti (*source files*) e di intestazione (*header files*) è possibile compilare il tutto lanciando il comando `make` da terminale, come illustrato nel Listato 1. Al fine di semplificare lo sviluppo del progetto,

i file di cui al Listato 1 possono essere scaricati qui. I suddetti file possono essere utilizzati come punto di partenza per sviluppare il progetto.

3 Visualizzazione con gnuplot

Per visualizzare l'andamento della simulazione è possibile usare il programma gnuplot. A tal fine è necessario salvare su file, ogni volta che si vuole visualizzare lo stato del sistema (ad esempio a ogni passo di calcolo o a intervalli regolari tipo ogni 100 passi), le coordinate e il raggio di ogni particella. Coordinata x, y e raggio devono essere separati da uno spazio e gli stati delle particelle vanno salvati su righe consecutive, come mostrato nel Listato 3. Si noti che altre informazioni che concorrono a definire lo stato della particella come, ad esempio, la velocità, non sono tenute in considerazione. Lo studente, tuttavia, può arricchire il file di output con le informazioni che ritiene opportune, a patto che esse siano utilizzate ai fini della visualizzazione (di cui vedremo un esempio a breve) o per valutare determinare informazioni sullo stato del sistema (come, ad esempio, l'energia totale).

```
0.0255653  0.0282956  0.0005
0.0298076  0.0205495  0.0005
0.0221685  0.0156014  0.0005
0.00796418  0.0099712  0.0005
0.0207736  0.0163346  0.0005
0.0176561  0.0133246  0.0005
0.0280166  0.0164294  0.0005
0.0254514  0.0173138  0.0005
0.0213257  0.023621  0.0005
0.020957  0.0134877  0.0005
0.019163  0.00801469  0.0005
0.0157925  0.0211059  0.0005
0.0199523  0.0220452  0.0005
...
```

LISTATO 3: Esempio di file di output. Lo stato di ogni particella è salvato su una riga del file in termini di coordinata x, y e raggio.

Una volta salvato su file lo stato di ogni particella del sistema secondo la convenzione del Listato 3 è possibile visualizzare il sistema di particelle eseguendo da terminale il programma gnuplot dalla directory in cui si trova il file di output (il cui nome supporremo sia `out.txt`) e inserendo successivamente il seguente comando:

```
plot 'out.txt' using 1:2:3 with circles
```

Dovrebbe essere visualizzato qualcosa di simile a quanto mostrato in Figura 2. Si noti che nella figura il bordo del dominio è circondato da particelle. Nel sistema visualizzato, infatti, si è scelto di modellare il bordo con particelle fittizie in cui è registrato il valore del vettore normale allo spigolo nei

campi relativi alla velocità (che ovviamente è zero per quelle particelle). Ad esempio, si consideri una generica particella del bordo in basso. Per ognuna di esse, il campo velocità è valorizzato con il vettore normale $\boldsymbol{n} = (0, 1)$. Il vettore normale puo essere qundi utilizzato per valutare la distanza tra una generica particella e lo spigolo usando la relazione della distanza tra punto (centro della prticella generica) e piano (individuato dal vettore normale). Soluzioni alternative possono ovviamente essere adottate.

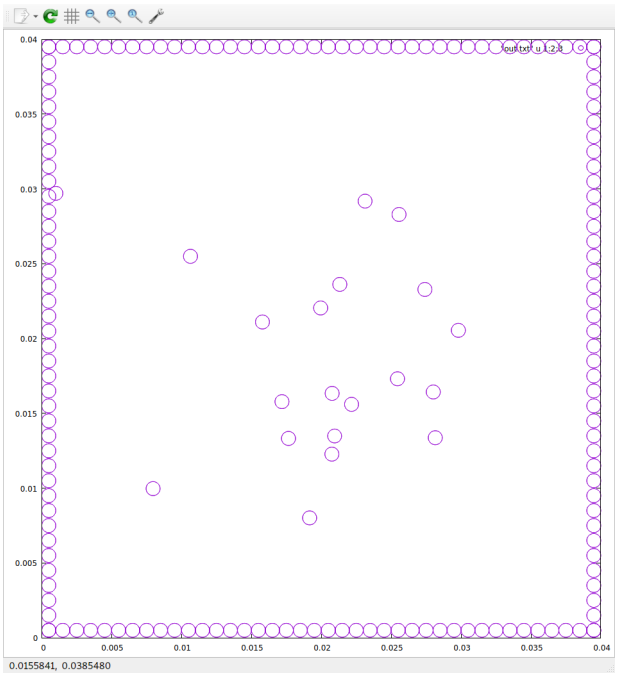

FIGURA 2: Sistema di 20 particelle visualizzato con gnuplot.

Si noti che ogni volta che si vuole aggiornare la visualizzazione, ad esempio dopo che una nuova configurazione del sistema è stata salvata su file (sovrascrivendo ad esempio il file `out.txt`), è necessario ridare il comando

```
plot 'out.txt' using 1:2:3 with circles
```

Tuttavia, è possile fare in modo di rieseguire il comando a intervalli regolari di tempo, ad esempio ogni secondo, salvando su file (di nome `loop.plot`) quanto riportato di seguito

```
#!/usr/bin/gnuplot
plot 'out.txt' using 1:2:3 with circles
pause 1
reread
```

e invocando una sola volta da terminale il comando

```
gnuplot loop.plot
```

Si noti che quest'ultimo comando bloccherà il temrinale per cui si consiglia di utilizzare un secondo terminale diverso da quello utilizzato per eseguire la simulazione. Per terminare gnuplot sarà quindi sufficiente chidere il terminale secondario.

4 Raccomandazioni finali

Sentitevi liberi di apportare le modifiche che ritenete utili e di effettuare le analisi che ritenete opportune per capire se la simulazione è plausibile dal punto di vista fisico. Ad esempio, sarebbe interessante capire (magari plottando un grafico gnuplot) se l'energia totale del sistema varia o si mantiene costante e, nel primo caso, qual è il problema e se è possibile risolverlo. Inoltre sarebbe interessante visualizzare il campo di velocità del sistema particellare, cosa che è possibile fare con gnuplot a patto di salvare su file anche il vettore velocità di ogni singola perticella.