Sistemi Multimediali e Laboratorio

Roberto Ranon
Esercitazione n.4 del 20/03/2018: trasformazioni

Nota: per svolgere questi esercizi, dovete aver scaricato e installato l'ultima versione di Processing (attualmente, la 3.3, ma qualsiasi versione dalla 2.x dovrebbe andare bene). Se state lavorando su una delle macchine del laboratorio, dovreste trovare Processing tra i programmi disponibili in ambiente Windows. Se ci sono problemi, avvisatemi durante la lezione o con un mail a roberto.ranon@uniud.it.

*Le slide che presentano brevemente i concetti e i comandi necessari sono disponibili all'indirizzo **https://users.dimi.uniud.it/~roberto.ranon/slides/processing-intro.pdf** (o sul sito **elearning.uniud.it**). Per maggiori informazioni, consultate la guida di riferimento di Processing: basta scrivere nell'editor l'istruzione, selezionarla, cliccare su di essa con il tasto destro, e scegliere "Find in Reference".*

Ci aspettiamo che, al termine del laboratorio, abbiate completato almeno gli esercizi fino al n. 5. Se non è così, cercate di completarli a casa.

1. Le trasformazioni sono uno strumento per spostare, ruotare e scalare gli oggetti disegnati. Iniziamo dallo spostamento (traslazione).

   Un'operazione di traslazione (funzione `translate(x,y)`) sposta l'origine del sistema di coordinate della finestra (che normalmente e' l'angolo in alto a sinistra) di `x` unità nella larghezza, e di `y` unità nell'altezza. Dopo l'operazione, le istruzioni che disegnano oggetti fanno riferimento al nuovo sistema di coordinate. Ad esempio, con le istruzioni:

```
translate(60, 80);
rect(20, 20, 40, 40);
```

   Il rettangolo non viene disegnato con l'angolo superiore sinistro alle coordinate (20,20), ma alle coordinate (60+20, 80+20), cioè alle coordinate (20,20) *in un sistema di coordinate con l'origine in (60,80)*.

   Se ora aggiungiamo una traslazione e un altro rettangolo, che cosa succede?

```
translate(60, 80);
rect(20, 20, 40, 40);
translate(60, 80);
rect(20, 20, 40, 40);
```

   Com'era prevedibile, il secondo rettangolo verrà disegnato alle coordinate (60+60+20, 80+80+20). In altre parole, le traslazioni si sommano tra loro, e ogni figura viene disegnata alle coordinate derivanti dalla somma delle traslazioni effettuate fino al momento del comando che la disegna. E se volessimo ora disegnare qualcosa usando il sistema di coordinate precedente alla seconda traslazione? O il sistema di coordinate originario?

   La funzione `pushMatrix()` "salva" la posizione corrente del sistema di coordinate, mentre la funzione `popMatrix()` riporta il sistema di coordinate nella situazione in cui era prima dell'ultimo `pushMatrix()`. Ad esempio:

```
pushMatrix();
translate(60, 80);
```

```
rect(20, 20, 40, 40);
popMatrix();
rect(20, 20, 40, 40);
```

Il primo rettangolo viene disegnato alle coordinate (60+20,80+20), mentre il secondo rettangolo viene disegnato alle coordinate (20,20) nel sistema di coordinate con l'origine in (0,0), cioè il sistema di coordinate che era attivo prima del comando `pushMatrix()`.

Processing possiede uno *stack* (una pila) di sistemi di coordinate, in cui quello che in si trova in cima è utilizzato per disegnare. Inizialmente, lo stack contiene un solo sistema di coordinate, quello di default (con l'origine in (0,0)). La funzione `pushMatrix()` copia il sistema di coordinate corrente in cima allo stack (creando un nuovo sistema di coordinate corrente identico al precedente); l'operazione `translate()` (o qualunque altra trasformazione) agisce sul sistema di coordinate in cima allo stack, modificandolo; la funzione `popMatrix()` elimina il sistema di coordinate in cima allo stack: quindi, quello che si trovava immediatamente sotto diventa il nuovo sistema di coordinate corrente. Il termine `Matrix` si riferisce al fatto che le trasformazioni vengono internamente rappresentate da matrici.

Partite dalla vostra soluzione dell'esercizio 6 dell'esercitazione 1 (disegnare un'automobile). Il vostro codice per disegnare l'automobile in una posizione arbitraria contiene istruzioni del tipo

```
rect(x,y-10,30,20);
line(x-15,y-10,x+15,y-10);
```

dove x e y sono, tipicamente, il centro dell'automobile o qualche punto di riferimento. Riscrivete l'esercizio in modo da eliminare tutte queste somme e sottrazioni, usando invece un'istruzione `translate` per posizionare le parti dell'automobile nel punto desiderato. Come potrete notare, `translate` vi consente di semplificare le istruzioni di disegno.

2. Modificate l'esercizio precedente in modo da disegnare 20 automobili in posizioni casuali. Le automobili devono tutte apparire, almeno parzialmente, all'interno della finestra.

3. Modificate l'esercizio 1 in modo da disegnare una riga di automobili lungo tutta la larghezza della finestra.

Nota: gli esercizi 2 e 3 servono per farvi capire quando usare (e quando non usare) le istruzioni `pushMatrix()` e `popMatrix()`. In particolare, l'esercizio 2 è molto più semplice usandole, l'esercizio 3 risulta un po' più semplice facendone a meno, cioè accumulando le traslazioni.

4. E' possibile anche scalare il sistema di coordinate corrente con la funzione `scale(x)`, dove x è il fattore di scala. Se 0<x<1, il sistema di coordinate viene rimpicciolito; per x=1, resta uguale, mentre un valore maggiore di 1 ingrandisce proporzionalmente tutto ciò che viene disegnato dopo.

Attenzione a quando combinate insieme due trasformazioni: l'ordine è importante. Per verificarlo, provate a disegnare un quadrato centrato in (0,0) e poi a:

- traslarlo di (width/2, height/2), e poi scalarlo con fattore 2, oppure

- scalarlo con fattore 2, e poi traslarlo di (width/2, height/2)

Come vedrete, il risultato è diverso. Nel secondo caso, infatti, la scalatura con fattore 2, applicata prima della traslazione, ha l'effetto di raddoppiare l'entità della traslazione! Dunque, bisogna stare attenti all'ordine con cui si effettuano le trasformazioni.

Modificate ora l'esercizio 2 in modo che le 20 automobili, oltre ad essere disegnate in posizione casuale, vengano disegnate con un fattore di scala casuale tra 0.5 e 2.

5. Il comando `rotate(x)` ruota il sistema di coordinate di x radianti intorno all'origine (potete usare i gradi con la funzione `radians(x)` di Processing, cioè `rotate(radians(x))`, dove x è ora in gradi). Un angolo positivo ruota in senso orario. Modificate l'esercizio 2 in modo che le 20 automobili, oltre ad essere disegnate in posizione casuale, con un fattore di scala casuale tra 0.5 e 2, vengano ruotate con un angolo casuale tra 0 e 2**π** intorno al proprio centro.

6. Il videogioco *Asteroids* del 1979 (https://it.wikipedia.org/wiki/Asteroids) consentiva al giocatore di comandare una navicella (rappresentata da un triangolo) modificando la sua orientazione tramite due tasti e usando un terzo tasto per accendere i motori (il gioco prevedeva anche la distruzione di asteroidi, ma lasceremo perdere questo aspetto nel nostro esercizio).

   Questo esercizio vi chiede di riprodurre questi aspetti del gioco in uno sketch Processing:

   - la navicella deve essere rappresentata da un triangolo isoscele, inizialmente al centro dello schermo e con "la punta" rivolta verso l'alto

   - il giocatore deve poter modificare l'orientazione della "punta" della navicella, facendola ruotare verso destra con il tasto freccia destra, e verso sinistra con il tasto freccia sinistra. Per gestire la pressione dei tasti, potete usare la variabile `keyPressed` all'interno del metodo `draw()` (https://processing.org/reference/keyPressed.html), insieme alle variabili `key` (https://processing.org/reference/key.html) e `keyCode` (https://processing.org/reference/keyCode.html).

   - il giocatore, mediante il tasto 'z', "accende" i motori imprimendo alla navicella una forza nella direzione della punta. Il moto della navicella è controllato poi dalle equazioni di Newton e del moto che abbiamo visto nell'esercitazione precedente. Attenzione: cercate di stabilire un valore opportuno per il modulo della forza, in modo che l'accelerazione non sia troppo elevata o troppo bassa; inoltre, è probabilmente opportuno limitare la velocità della navicella per evitare che diventi incontrollabile, e magari inserire un piccolo attrito.

   - ogni volta che la navicella esce dallo schermo, deve rientrare dalla parte opposta.

   - quando i motori vengono accesi, visualizzate i "fumi di scarico" mediante un particle system che genera una decina di particelle (sfere) che si muovono con accelerazione opposta alla direzione della forza che sposta la navicella, e con velocità iniziale casuale. Le particelle devono sparire quando escono dal bordo dello schermo.