GUIT *meeting* 2012

Grafica ad oggetti con LuaTEX

Roberto Giacomelli

Napoli, 27 ottobre 2012

Le parole chiave

Grafica nel mondo TEX potenti pacchetti consentono di produrre disegni vettoriali, da PGF (Ti*k*Z), a *pstricks* fino a METAPOST, programmando il tracciamento degli enti geometrici;

ad oggetti il linguaggio macro impegna l'utente in un modo semplice, ma ci si chiede se egli non possa trarre maggiori benefici da un linguaggio ad oggetti, con il quale si rappresentano direttamente i concetti;

con LuaTEX il nuovo motore di composizione rende possibile lo sviluppo di linguaggi ad alto livello di astrazione compreso quelli con il paradigma OOP. Lua è un linguaggio di scripting efficiente che supporta più stili di programmazione.

Il linguaggio

Nel mondo TEX il documento finale è il risultato della compilazione di un sorgente attraverso l'uso di un *linguaggio*.

Caratteristiche dei sistemi asincroni:

- non è possibile interventire sul documento se non attraverso il codice scritto secondo il linguaggio;
- l'intuitività e l'efficienza espressiva del linguaggio sono essenziali per la produttività dell'utente.

Il linguaggio

Nel mondo TEX il documento finale è il risultato della compilazione di un sorgente attraverso l'uso di un *linguaggio*.

Paradigmi dal punto di vista dell'utente:

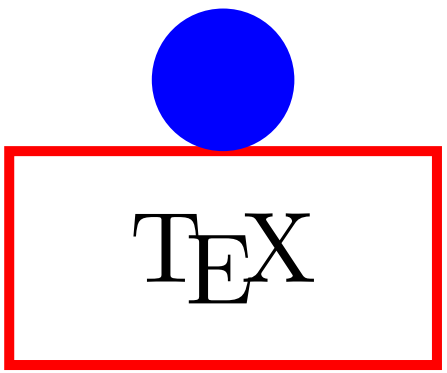
- Linguaggio macro — tradizionale
  - l'utente sostanzialmente pensa per comandi;
  - normalmente non è necessario scrivere codice.
- Linguaggio ad oggetti — da sperimentare
  - l'utente sostanzialmente pensa per entità strutturate;
  - è necessario programmare (utilizzando librerie OOP).

Il linguaggio grafico di Ti*k*Z

Nella grafica è richiesto un linguaggio particolarmente evoluto...
...sempre più utenti scelgono il pacchetto PGF di Till Tantau attraverso Ti*k*Z, l'interfaccia in LaTeX di PGF, che offre un potente linguaggio macro basato su un numero limitato di comandi che elaborano argomenti complessi ed adottano una struttura concettuale di stili ed opzioni chiave=valore:

```
\begin{tikzpicture}
\draw[red, line width=2pt]
         (0,0) rectangle (3,1.5);
\node[scale=2] at (3/2,1.5/2) {\TeX};

\draw[blue,fill] (3/2,1.5+0.5) circle (.5);
\end{tikzpicture}
```



Object Oriented Programming – Basi

Nella programmazione ad oggetti il linguaggio include alcuni concetti comuni:

- gli oggetti si riferiscono ad una classe (lo stampo);
- l'oggetto deve essere creato per mezzo di un costruttore;
- per ogni singola istanza di un oggetto possiamo chiamarne i metodi (si usa la *notazione punto* oppure in Lua la *notazione due punti*).

```
-- nuovo oggetto (chiamo il costruttore della classe Rectangle)
r = Rectangle:new(1,2)

-- chiamo un metodo dell'oggetto appena creato (semicolon-notation)
r:draw()
```

Non entreremo nel dettaglio di come la programmazione ad oggetti sia implementata in Lua (nell'articolo si trovano tutti i dettagli).

La libreria GOL

Per questo lavoro è stata scritta una libreria chiamata GOL per sperimentare la OOP. L'utilizzo è molto semplice: all'interno di un ambiente `gol` si scrive direttamente il codice Lua avendo a disposizione per disegnare alcuni oggetti:

- il punto: classe `Point`;
- il rettangolo: classe `Rectangle`;
- il cerchio: classe `Circle`;
- la polilinea: classe `Line`.

Tutte le classi implementano il metodo `draw()`.
Struttura generale del codice:

```
\begin{gol}
   <codice Lua>
\end{gol}
```

Implementazione di GOL:

Il codice della libreria sperimentale GOL si trova in una serie di file in Lua che vengono caricati per mezzo di istruzioni `require()`. Ecco un estratto per l'implementazione della classe `Rectangle`:

```
-- oggetto rettangolo
Rectangle = {}                          -- init
setmetatable(Rectangle, Element)        -- eredito da 'Element'
Rectangle.__index = Rectangle           -- metametodo
...
-- metodo draw()
function Rectangle:draw(v)
   ...
   local frmt = '\\draw (%0.3f,%0.3f) rectangle (%0.3f,%0.3f);'
   ...
      tex.sprint(string.format(frmt, x1+vx,y1+vy,x2+vx,y2+vy))
   ...
end
```

Implementazione di GOL: l'interfaccia LuaLaTeX

L'interfaccia LuaLaTeX è fornita da un semplicissimo pacchetto che definisce un unico ambiente chiamato `gol`:

```
% lualatex package
\ProvidesPackage{gol}[2012/08/01 v1.0 Graphic Object Library]
\RequirePackage{luacode}
\RequirePackage{tikz}

\directlua{require "gol"}
\newenvironment{gol}
  {\tikzpicture
   \luacode}
  {\endluacode
   \endtikzpicture}
```

Implementazione di GOL: ereditarietà

Uno dei concetti introdotti dal paradigma OOP è la possibilità di mettere in relazione fra loro gli oggetti per mezzo del meccanismo dell'ereditarietà costruendo una gerarchia.
In GOL tutti gli oggetti grafici ereditano da una classe base chiamata `Element`, una sorta di oggetto geometrico virtuale.
Tutti i campi ed i metodi di questa classe faranno automaticamente parte di tutte le classi derivate, ovvero gli elementi geometrici veri e propri.

```
function Element:setLinewidth(lw)
   if lw then self.lw = lw end
end
function Element:setLineColor(col)
   if col then self.linecol = col end
end
function Element:setFillColor(col)
   if col then self.fillcol = col end
end
```

Confronto di linguaggi

Confrontiamo i linguaggio di Ti*k*Z con quello di GOL con un esempio semplice:

```
\begin{tikzpicture}
\draw  (0,0) rectangle (1,2.5);
\end{tikzpicture}
```

```
\begin{gol}
r = Rectangle:new{x=1,y=2.5}
r:draw()
\end{gol}
```

In GOL serve una riga di codice in più: prima occorre creare il rettangolo delle dimensioni desiderate e solo dopo si può creare il disegno.
Altra differenza sono le coordinate: in Ti*k*Z è necessario fornire le coordinate dei due vertici, in GOL invece il punto di origine locale dell'oggetto è definito implicitamente, infatti:

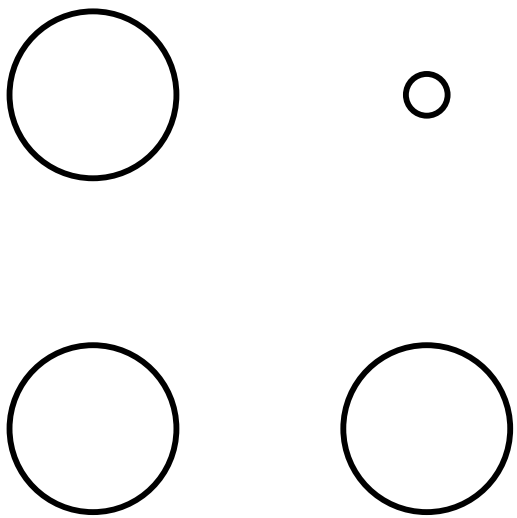
L'utente manipola enti geometrici e non direttamente un disegno

I parametri del metodo `draw()`

Continuamo l'esplorazione di GOL con un esempio in cui si utilizzano le potenzialità espressive del metodo `draw()`. Grazie ad un tipo particolare di costruttore delle tabelle in Lua (l'unica struttura dati presente nel linguaggio), è immediato implementare un sistema di opzioni chiave=valore:

```
\begin{gol}
c = Circle:new{}

c:draw()
c:draw{dx=4}
c:draw{dy=4}
c:draw{dx=4, dy=4, scale=0.25}
\end{gol}
```

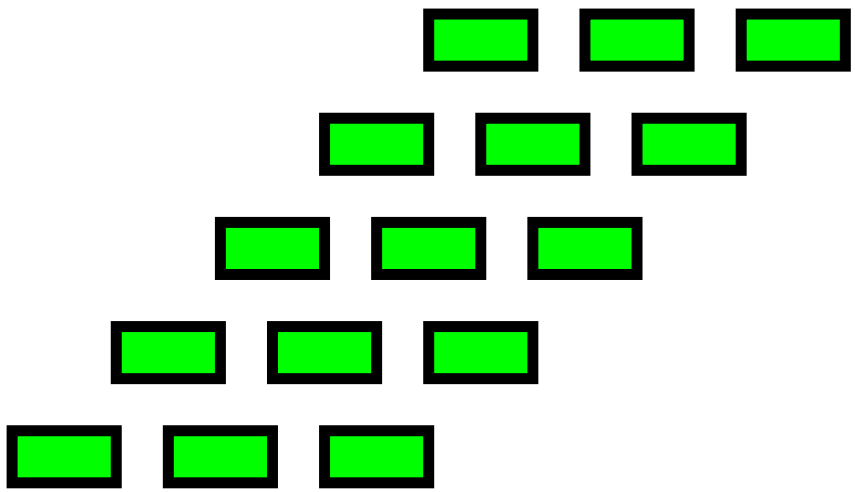


draw() ed il disegno a griglia

Grazie a Lua è facile inventare nuovi parametri per il metodo draw(), per esempio il disegno su griglia:

```
\begin{gol}
r = Rectangle:new{y=0.5}
r:setLinewidth(3)
r:setFillColor "green"

-- disegno oggetto su griglia
r:draw{grid={rows=5, cols=3,
             angle=45,
             dx=1.5
            }
      }
\end{gol}
```



Costruzione del disegno: `RefPoint`

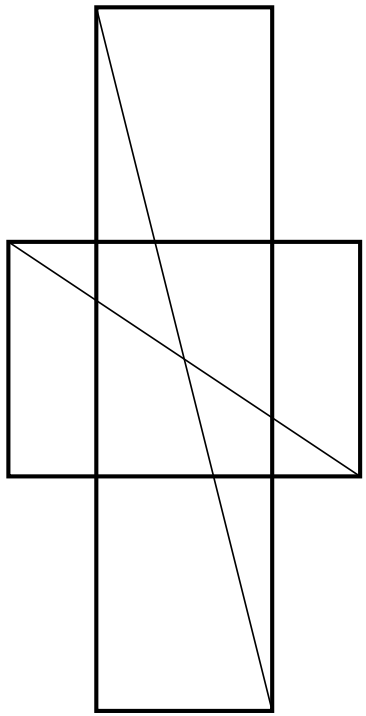
La principale linea guida concettuale della libreria GOL è quella di facilitare la costruzione grafica con *riferimenti* e parametri al fine di ottenere il massimo vantaggio dal disegno programmato.
Infatti, possiamo riferirci a punti di un oggetto tramite i `RefPoint`:

```
\begin{gol}
r = Rectangle:new{x=3,y=2}
nw = r:getPoint{vertex='nw'}
se = r:getPoint{vertex='se'}

l = Line:new{nw,se}
r:draw(); l:draw()

r:setX(1.5); r:setY(6)

r:draw(); l:draw()
\end{gol}
```



Un mazzo di fiori casuale

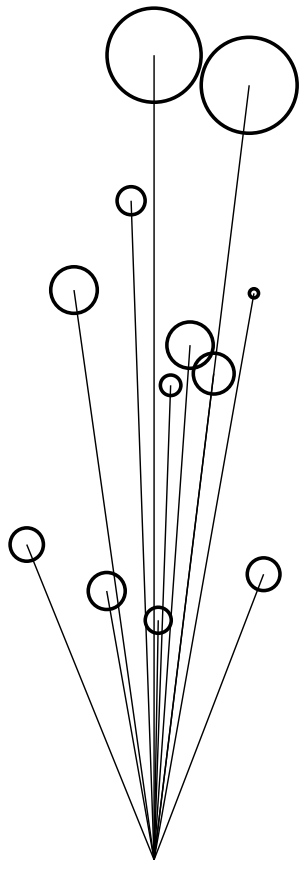
Un altro esempio di utilizzo di oggetti `RefPoint` è il seguente:

```
\begin{gol}
-- dozzina di fiori casuale
fiore = Circle:new()
centroFiore = fiore:getPoint{'center'}

gambo = Line:new{{0,0}, centroFiore}
maxangle, maxdim = 24, 8  -- parametri

for i=1,12 do
  local alfa = math.random(-maxangle,maxangle)
  local dist = maxdim*math.random()+2

  fiore:setxy{polar={90-alfa, dist}}
  fiore:draw{scale=0.5*math.random()}
  gambo:draw()
end
\end{gol}
```



Assemblare oggetti grafici

Un `Assembly` rappresenta un *gruppo* di oggetti grafici.
Il concetto di `Assembly` in GOL è una conseguenza della struttura ad oggetti nella quale trovano diretta rappresentazione non disegni ma concetti geometrici.

Per costruirlo abbiamo bisogno di due tipi di informazioni:

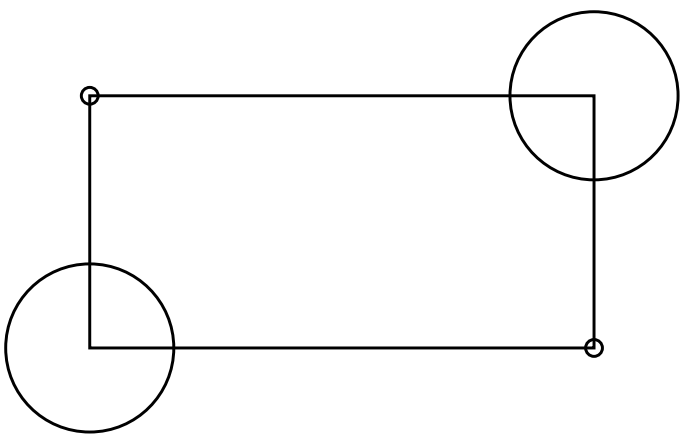
- gli oggetti che fanno parte del gruppo;
- la posizione di ciascuno di questi oggetti.

A ciascuna di queste informazioni corrisponde un'azione:

- inserire un oggetto nel gruppo: metodo `install()`;
- disporre gli oggetti sul piano: metodo `deploy()`.

Esempio di assemblaggio/1

Vogliamo realizzare questo disegno:



Creiamo un nuovo oggetto `Assembly` ed installiamo gli oggetti disponendoli sul piano:

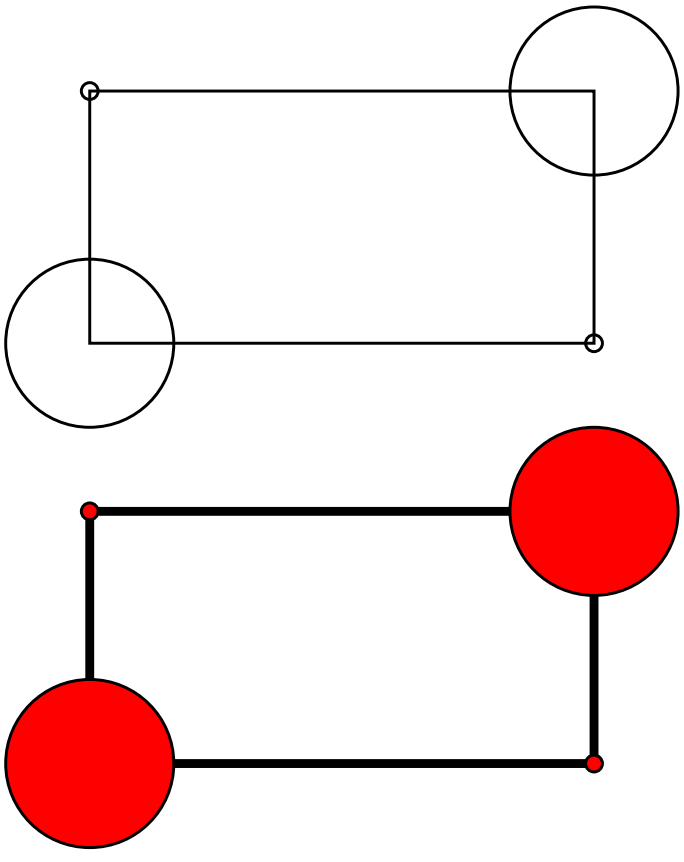
```
\begin{gol}
r = Rectangle:new{x=6,y=3}
c = Circle:new()

fig = Assembly:new{}
fig:install(r)
fig:install(c)

fig:deploy(r, {x=3, y=1.5})
fig:deploy(c, {x=0, y=0})
fig:deploy(c, {x=6, y=3})
fig:deploy(c, {x=6, y=0, scale=0.1})
fig:deploy(c, {x=0, y=3, scale=0.1})
fig:draw()
\end{gol}
```

Esempio di assemblaggio/2

A questo punto modificando le proprietà del singolo oggetto. . .



```
r = Rectangle:new{x=6,y=3}
c = Circle:new()
fig = Assembly:new{}
fig:install(r)
fig:install(c)
fig:deploy(r, {x=3, y=1.5})
fig:deploy(c, {x=0, y=0})
fig:deploy(c, {x=6, y=3})
fig:deploy(c, {x=6, y=0, scale=0.1})
fig:deploy(c, {x=0, y=3, scale=0.1})

fig:draw()

c:setFillColor "red"
r:setLinewidth(3)
fig:draw{dy=-5}
```

Ulteriori sviluppi

Disegno parametrico:
È possibile far dipendendere la geometria degli oggetti da *parametri*. Questa funzionalità permette all'utente di affinare il disegno semplicemente cambiando il valore dei parametri.

Organizzare il disegno:
Una nuova proprietà `Layer` potrebbe essere associata ai singoli oggetti. Potremo così facilamente regolare i dettagli del disegno in funzione della scala di rappresentazione, spegnendo o viceversa attivando i `Layer`.

Conclusioni

Partendo dalla considerazione che dal linguaggio dipende in buona parte la produttività dell'utente, si è sperimentato una libreria per la grafica in LuaTEX (LuaLATEX) basata sulla programmazione ad oggetti da parte dell'utente.
Vantaggi e potenzialità (ancora da dimostrare con progetti reali):

- lavorare con gli oggetti è più intuitivo;
- le strutture gerarchiche rendono il linguaggio più coerente.

Svantaggi e limitazioni:

- l'utente deve saper programmare in Lua;
- è obbligatorio l'uso di LuaTEX od uno dei sui formati.

→ *il paradigma ad oggetti è un sicuro candidato per lo sviluppo di pacchetti futuri!*

Grazie per l'attenzione.
Domande?