L'ARTE DI DISEGNARE CON LaTeX

TOMMASO GORDINI & LORENZO PANTIERI*

27 febbraio 2014

INDICE

Non è raro che un documento composto con LaTeX richieda di realizzare un disegno. Se è molto complesso o se si ha poco tempo a disposizione per imparare un nuovo linguaggio, la soluzione più semplice e veloce è produrlo con un software di grafica vettoriale esterno, registrarlo in formato PDF e includerlo nel documento come al solito. In questo modo, però, viene meno la perfetta integrazione di testo e figure tipica di LaTeX e garantita invece da alcuni pacchetti specializzati nel disegno e compresi nelle principali distribuzioni.

I più importanti di essi sono TikZ e PSTricks, entrambi potentissimi e sostanzialmente equivalenti nell'uso comune. Il primo è direttamente compatibile con pdfLaTeX, mentre il secondo, basato sul linguaggio PostScript, lo è solo adottando qualche espediente, e anche per questo motivo, nonostante le maggiori attitudini al calcolo e al disegno tridimensionale, gli utenti preferiscono generalmente il suo diretto concorrente.

Quest'articolo, basato essenzialmente su [Tantau, 2013], introduce alle nozioni fondamentali necessarie per produrre semplici disegni con TikZ.

* Questo lavoro è stato scritto con la competente ed entusiasta collaborazione di Liverpool del GUIT. Grazie a Claudio Fiandrino, Claudio Beccari e Ivan Valbusa per le loro preziose osservazioni. A Thomas Sturm per la sua proverbiale pazienza.

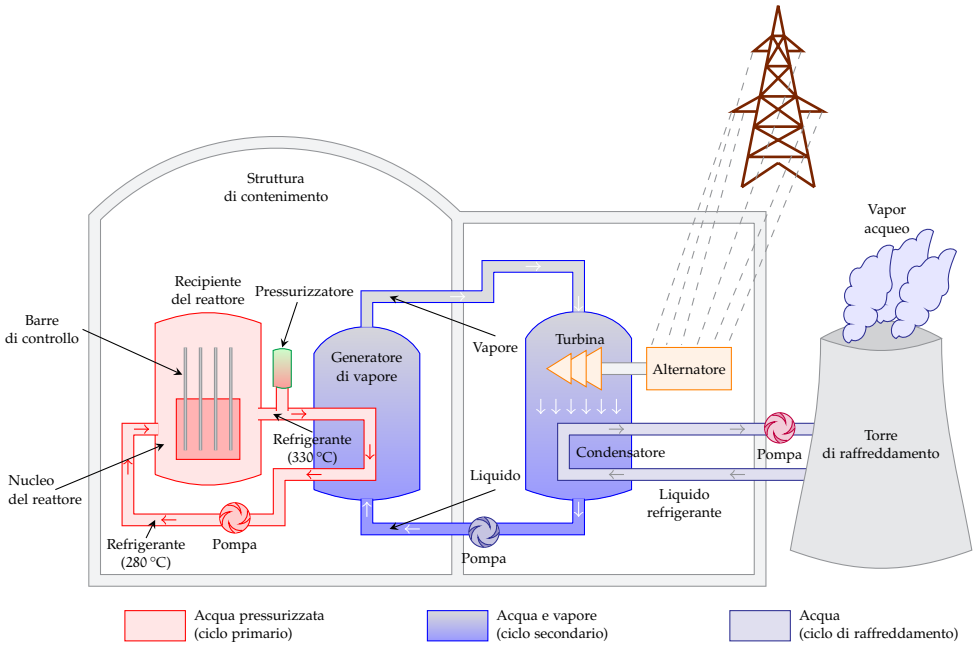


Figura 1: Schema del funzionamento di una centrale nucleare (l'esempio è tratto da ☞ TEXAMPLE).

1 INTRODUZIONE

Un'immagine *vettoriale* è un disegno i cui elementi sono memorizzati mediante una descrizione sintattica delle loro caratteristiche geometriche e delle loro proprietà (colore, spessore delle linee, eccetera), anziché mediante una conversione in matrice di punti colorati, come avviene invece in un'immagine *raster*. Ciò permette da un lato di ridurre le dimensioni del file corrispondente e dall'altro di ottenere un'elevata qualità indipendentemente dalla risoluzione.

TikZ è un pacchetto per produrre disegni vettoriali a partire da una descrizione geometrica, che fornisce comandi di alto livello basandosi a propria volta su PGF, un linguaggio di livello più basso (un po' come LaTeX fa rispetto a TeX, per capirci). PGF è l'acronimo di *Portable Graphics Format* ("formato di disegni portabile"), mentre TikZ è l'acronimo ricorsivo di TikZ *ist* kein *Zeichenprogramm* ("TikZ *non* è un programma di disegno").

L'idea di *programmare un disegno* potrebbe sembrare a prima vista esoterica ma, a pensarci bene, non lo è molto di più dell'idea di *programmare un documento* alla base di LaTeX. Si tratta di vincere una certa (e comprensibile) perplessità iniziale e di investire energie per imparare come fare, ma la fatica sarà ampiamente ripagata: se si desidera realizzare un lavoro di qualità senza compromessi, in cui testo e grafica si integrino perfettamente, TikZ è irrinunciabile e, non meno importante, permette di disegnare "nello spirito di LaTeX".

Ricorrere o meno a TikZ dipende dalle proprie esigenze di qualità tipografica e dal tipo di disegni da realizzare, ma prima di tutto dal *tempo* a disposizione. Per *padroneggiare* TikZ ci vuole *molto* tempo, e non è detto che l'utente ce l'abbia. In tal caso, appoggiarsi a un programma esterno per produrre ciò che serve e includerlo poi nel documento con LaTeX è un compromesso onorevole.

Se invece si opta per TikZ, si sappia che schemi semplici si realizzano rapidamente (anche grazie agli esempi contenuti nella documentazione e a quelli disponibili in Rete su ☞ TEXAMPLE, per esempio), ma per disegni complessi come lo schema del funzionamento di una centrale nucleare riportato nella figura 1, tanto per fare un esempio all'estremo opposto, ci vuole molto più tempo: si può fare anche quello, perché con TikZ si può disegnare virtualmente *qualunque* cosa.

Per il momento, le potenzialità di TikZ nel disegno tridimensionale sono piuttosto limitate, perciò quest'articolo si occuperà esclusivamente di disegni in due dimensioni.

2 FERRI DEL MESTIERE

2.1 Caricamento, librerie e ambienti

TikZ si carica nel solito modo:

```
\usepackage{tikz}
```

e a propria volta invoca automaticamente i seguenti pacchetti: graphicx, keyval e xcolor. Se fosse necessario caricarli con qualche opzione, dunque, si ricordi di farlo *prima*.

Data la sua versatilità, per organizzare meglio il codice e ridurre i tempi di composizione del documento l'autore ha pensato di dargli una struttura modulare basata sul concetto di libreria. Una *libreria* è una porzione di codice del pacchetto specializzata nell'eseguire una particolare funzione, da caricare solo se il disegno da realizzare lo richiede effettivamente. Le librerie si caricano *nel preambolo dopo* TikZ, scrivendone il ⟨*nome*⟩ nell'argomento di

```
\usetikzlibrary{⟨nome⟩}
```

che, come al solito, permette di caricarne più d'una, separandole con la virgola. Le librerie di TikZ sono numerose e si distinguono in *interne*, cioè già presenti nel pacchetto (se ne veda la documentazione) ed *esterne*, cioè veri e propri pacchetti che si caricano come si è appena visto. Di seguito si descrivono brevemente alcune librerie del primo tipo per scopi generici (altre, più specializzate, sono descritte nel paragrafo 8.1 a pagina 49):

- angles permette di marcare ed etichettare gli angoli;
- arrows.meta amplia la scelta predefinita degli stili di freccia e permette di personalizzarne l'aspetto nel dettaglio;
- calc permette di eseguire calcoli sulle coordinate;
- graphdrawing permette di migliorare il posizionamento automatico degli elementi di un grafo (richiede LuaLaTeX);
- graphs semplifica la sintassi per disegnare i grafi;
- intersections permette di determinare i punti di intersezione di due percorsi;
- matrix permette di realizzare matrici di nodi;
- patterns permette di riempire i percorsi con motivi di fantasia;
- plotmarks mette a disposizione ulteriori marcatori oltre a quelli predefiniti;
- quotes semplifica la sintassi delle etichette di testo;
- shadows permette di aggiungere le ombre agli elementi del disegno;
- la famiglia decorations permette di aggiungere elementi di decorazione al disegno;
- la famiglia shapes permette di inserire forme già pronte per l'uso.

L'ambiente principale di TikZ è `tikzpicture`, che presenta la seguente sintassi:

```
\begin{tikzpicture} [⟨opzioni globali⟩]
⟨istruzioni di TikZ⟩
\end{tikzpicture}
```

dove:

- le ⟨*opzioni globali*⟩, da separare con la virgola se più d'una, agiscono su *tutte* le istruzioni presenti nell'ambiente, se non ridefinite localmente;

- ⟨*istruzioni di* TikZ⟩ sono le istruzioni necessarie per realizzare il disegno.

Si noti inoltre che:

- TikZ sa riconoscere e ignorare gli spazi *non* significativi, che pertanto si possono aggiungere per aumentare la leggibilità del codice;
- le opzioni vengono applicate *rispettando l'ordine di scrittura*, perciò quando si dichiarano più opzioni incompatibili tra loro (come due colori diversi a uno stesso elemento del disegno) quella effettivamente assegnata è l'opzione dichiarata *per ultima*.

L'alternativa all'ambiente `tikzpicture` è il comando `\tikz`, con la sintassi:

```
\tikz {⟨istruzioni di TikZ⟩}
```

Si può trattare in modo particolare una determinata porzione di codice mettendola nell'ambiente `scope` (qui tradotto con *àmbito*), da annidare a propria volta in un ambiente `tikzpicture` insieme alle altre istruzioni. Le opzioni di `scope` saranno *locali*, cioè valide *solo al suo interno*:

```
\begin{tikzpicture} [⟨opzioni globali⟩]
...
\begin{scope} [⟨opzioni locali⟩]
⟨istruzioni di TikZ⟩
\end{scope}
...
\end{tikzpicture}
```

Poiché permettono di agire in una volta sola su porzioni anche ampie di codice, gli àmbiti sono molto utili per trattare come un'unità indivisibile un elemento complesso del disegno, cioè un particolare realizzato a propria volta tramite un insieme di istruzioni.

I disegni realizzati con TikZ sono oggetti in linea con il testo. Si rendono mobili inserendo l'ambiente `tikzpicture` o il comando `\tikz` in un normale ambiente `figure` come al solito:

```
\begin{figure}
\centering
\begin{tikzpicture} [⟨opzioni globali⟩]
⟨istruzioni di TikZ⟩
\end{tikzpicture}
\caption{⟨didascalia⟩}
\label{fig:⟨etichetta⟩}
\end{figure}
```

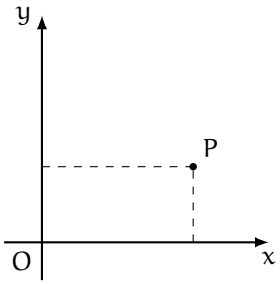
Documenti di soli disegni

Qualche circostanza particolare potrebbe richiedere un documento costituito esclusivamente da uno o più disegni. In tal caso si consiglia di ricorrere alla classe `standalone`, che si occupa di scontornare automaticamente le immagini (se ne consulti la documentazione).

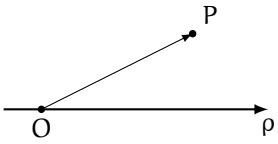
Altre volte può essere utile generare per ogni disegno del documento un file di output inserito automaticamente nel punto giusto (per ridurre i tempi di composizione, per esempio, oppure quando si vogliono riutilizzare i disegni in un poster). La libreria interna external permette di farlo automaticamente.

2.2 Sistemi di riferimento

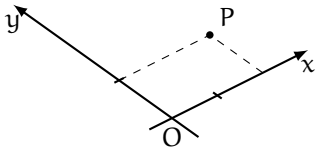
TikZ non richiede di specificare *a priori* un sistema di riferimento: infatti individua la posizione dei punti sulla *tela*, cioè la porzione della pagina riservata al disegno, tramite le loro coordinate e di volta in volta determina il sistema di riferimento adottato a partire dalla sintassi delle singole istruzioni. In questo modo, l'utente può usare i diversi sistemi



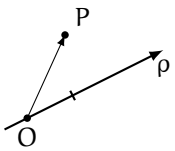
(a) Sistema di riferimento cartesiano associato alla tela.



(b) Sistema di riferimento polare associato alla tela.



(c) Sistema di riferimento cartesiano associato a una base generica.



(d) Sistema di riferimento polare associato a una base generica.

Figura 2: Posizione di uno stesso punto nei diversi sistemi di riferimento riconosciuti da Ti*k*Z.

di riferimento riconosciuti dal pacchetto in uno stesso disegno e addirittura in una stessa istruzione. Quelli più comunemente utilizzati sono il sistema di riferimento *cartesiano* e quello *polare*, ciascuno dei quali presenta due varianti a seconda che sia associato alla tela o a una generica base vettoriale.

Appartengono alla prima variante il classico sistema cartesiano ortogonale (figura 2a) e il sistema polare generalizzato (figura 2b), ai quali l'unità di misura delle lunghezze non è assegnata e perciò va specificata con una delle abbreviazioni riconosciute da LaTeX quando si dichiarano le coordinate del punto.

I sistemi appartenenti alla seconda variante (figure 2c e 2d) coincidono con quelli appena descritti, a meno che non intervenga l'utente a modificarli, ma l'unità di misura delle lunghezze è assegnata ed è pari a 1 cm, perciò non va specificata. In questi sistemi l'utente può variare separatamente direzione, verso e unità di misura degli assi coordinati.

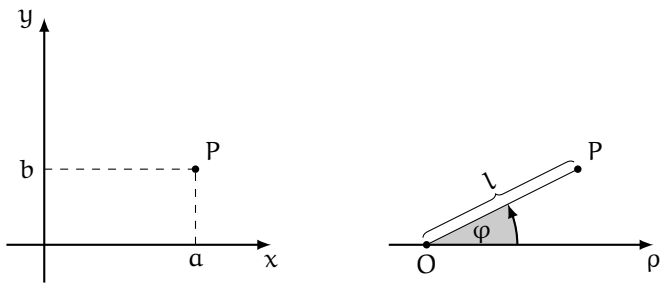
La figura 2 mostra come cambia la posizione di uno stesso punto nelle quattro varianti appena descritte. Si noti che Ti*k*Z accetta anche altri sistemi di riferimento (qui ignorati per semplicità) descritti nella documentazione del pacchetto.

La sintassi per individuare un punto del piano in coordinate cartesiane è

```
(⟨x⟩,⟨y⟩)
```

dove ⟨x⟩ e ⟨y⟩, separate *con la virgola* e racchiuse tra parentesi *tonde*, sono rispettivamente l'ascissa e l'ordinata del punto, cioè le proiezioni del punto sugli assi coordinati (si veda la figura 3a nella pagina seguente). Si noti quanto segue:

- se ⟨x⟩ e ⟨y⟩ sono misure di lunghezze come in `(1.5cm,2pt)`, le coordinate sono espresse nel sistema di riferimento associato alla tela;
- se ⟨x⟩ e ⟨y⟩ sono numeri come in `(2,1)`, le coordinate sono espresse nel sistema di riferimento associato a una base generica che, se non specificata, coincide con quella della tela;
- se ⟨x⟩ e ⟨y⟩ sono una misura di lunghezza e un numero, allora ogni coordinata è espressa in un sistema differente.



(a) Coordinate cartesiane P(a,b). **(b)** Coordinate polari P(φ:l).

Figura 3: Significato geometrico delle coordinate cartesiane e polari di uno stesso punto.

La sintassi per individuare un punto del piano in coordinate polari generalizzate è

```
(⟨φ⟩:⟨ρ⟩)
```

dove ⟨φ⟩ e ⟨ρ⟩, separate *con i due punti* e racchiuse tra parentesi tonde, sono rispettivamente la coordinata angolare in gradi sessadecimali e la coordinata radiale, come mostra la figura 3b. Si noti quanto segue:

- se ⟨ρ⟩ è una misura di lunghezza come in `(45.2:5mm)`, le coordinate sono espresse nel sistema di riferimento associato alla tela;
- se ⟨ρ⟩ è un numero come in `(45.2:0.5)`, le coordinate sono espresse nel sistema di riferimento associato a una base generica che, se non specificata, coincide con quella della tela;
- angoli che differiscono per un multiplo intero di 360° sono considerati equivalenti;
- il valore numerico di ⟨ρ⟩ può essere positivo, negativo o nullo; se negativo, la scrittura (φ:ρ) equivale a (φ + 180°:−ρ).

È appena il caso di osservare che nel linguaggio di Ti*k*Z *il separatore decimale è il punto*. Si noti che ⟨x⟩, ⟨y⟩, ⟨φ⟩ e ⟨ρ⟩ possono consistere anche di espressioni matematiche della forma descritta nel paragrafo 2.3 a fronte, alle quali viene sostituito automaticamente il risultato corrispondente.

Il confronto tra le figure 2b e 2d nella pagina precedente mostra chiaramente che nel riferimento polare associato a una base vettoriale generica diversa da quella della tela la coordinata radiale e quella angolare possono perdere il significato geometrico al quale si è abituati, perciò si presti particolare attenzione quando si modifica il sistema di riferimento predefinito. Di qui in avanti si considereranno esclusivamente i sistemi di riferimento associati alla tela e i corrispondenti generici con unità di misura pari a 1 cm. Il lettore è avvisato una volta per tutte che la forma e le proporzioni tra gli elementi del disegno potrebbero risultare alterate in altri sistemi di riferimento.

Oltre che rispetto all'origine, si possono esprimere le coordinate di un punto anche rispetto a un altro punto del disegno. Poiché l'argomento si presta a essere compreso più facilmente con degli esempi, sarà ripreso dettagliatamente nel paragrafo 3.2 a pagina 11.

Ti*k*Z stabilisce dimensioni della tela e posizione del disegno su di essa in base agli elementi del disegno, cercando di contenerli tutti nel minor spazio possibile. Ciò significa che di questi ultimi non conta tanto la posizione rispetto all'origine del sistema di riferimento, quanto le dimensioni e la posizione *relativa*. Per fare un esempio, un cerchio unitario nell'origine e uno nel punto P(2, 2) avranno esattamente lo stesso disegno e, nel secondo caso, l'origine non rientrerà nella tela. In qualche circostanza può essere utile, se non indispensabile, assegnare esplicitamente dimensioni e posizione della tela rispetto all'origine. In tal caso si rimanda il lettore alla documentazione del pacchetto per gli approfondimenti necessari.

Tabella 1: Alcune funzioni matematiche predefinite in Ti*k*Z.

Funzione	Effetto	Funzione	Effetto
`abs`	valore assoluto	`frac`	parte decimale
`acos`	arcocoseno	`int`	parte intera
`asin`	arcoseno	`ln`	logaritmo naturale
`atan`	arcotangente	`log10`	logaritmo decimale
`atan2`	arcotangente in due variabili	`max`	massimo
`ceil`	parte intera superiore	`min`	minimo
`cos`	coseno	`round`	arrotonda
`cot`	cotangente	`sin`	seno
`deg`	radianti → gradi	`sqrt`	radice quadrata
`floor`	parte intera inferiore	`tan`	tangente

Chiarimenti sulla terminologia adottata

Di qui in avanti, con i termini *misura*, *lunghezza* (*larghezza* o *altezza* quando opportuno) o, più in generale, *dimensione* si potrà indicare concisamente una misura (eventualmente algebrica) di lunghezza e si parlerà genericamente di *gradi* per indicare una misura di ampiezza in gradi sessadecimali. Inoltre, se non diversamente indicato, una coordinata angolare s'intenderà espressa rispetto a un asse polare parallelo ed equiverso all'asse delle ascisse della tela.

2.3 Espressioni coordinate

Oltre che nei modi appena visti, si possono indicare le coordinate di un punto (una sola o entrambe), con la stessa sintassi vista nel paragrafo 2.2 a pagina 4, anche tramite semplici espressioni matematiche, ricordandosi di racchiuderle tra graffe se contengono a propria volta parentesi tonde.

Ti*k*Z riconosce gli operatori più comuni e i seguenti degni di nota:

- l'operatore `^` di elevamento a potenza;
- l'operatore postfisso `r` che converte in gradi una misura di ampiezza in radianti;
- gli operatori di confronto per uguaglianza `==` e disuguaglianza `!=`;
- gli operatori logici di congiunzione `&&` e disgiunzione inclusiva `||`.

Si ricorda anche l'operazione condizionale `a ? b : c` (dove `a`, `b` e `c` rappresentano delle espressioni), che restituisce il risultato di `b` oppure di `c` a seconda che `a` sia vera oppure falsa (per esempio, `2<0 ? 1 : -1` restituisce `-1`). Inoltre il pacchetto riconosce le costanti matematiche pigreco `pi` e numero di Nepero `e`.

La tabella 1 elenca alcune funzioni predefinite di uso più frequente, tra cui si segnalano le funzioni circolari e le loro inverse, che richiedono e producono angoli in gradi e la funzione `deg`, che converte in gradi una misura in radianti. Gli argomenti delle funzioni vanno racchiusi tra parentesi *tonde* e, se più d'uno, separati con la *virgola*. Si noti che il valore di una funzione matematica è sempre un numero perciò, per trasformarlo in una dimensione, occorre moltiplicarlo per una lunghezza unitaria.

Ecco due esempi di espressioni, ciascuna delle quali può rappresentare una singola coordinata di un punto:

```
{sqrt(2)/2-sin(pi/4 r)}
```

oppure

```
{1cm*(sqrt(2)/2-sin(45))}
```

Si possono anche sommare lunghezze espresse in unità di misura differenti. In tal caso, i numeri adimensionali vengono automaticamente moltiplicati per la lunghezza di 1 pt, cioè `-5+1cm+2pt` equivale a `1cm-3pt`.

In aggiunta alle funzioni predefinite, è possibile definirne altre sfruttando quelle già esistenti mediante il comando

```
\pgfmathdeclarefunction{⟨nome⟩}{⟨num⟩}{\pgfmathparse{⟨espressione⟩}}
```

da dare nel preambolo, preferibilmente, oppure nel corpo del documento nel punto in cui serve. Si noti che:

- ⟨*nome*⟩ è il nome assegnato alla funzione, che può contenere lettere, numeri e linee basse (_), ma non può cominciare con un numero;
- ⟨*num*⟩ è il numero di argomenti, cioè di variabili;
- ⟨*espressione*⟩ è l'espressione analitica della funzione, nella quale gli argomenti sono identificati in modo analogo al comando `\newcommand` di LaTeX;
- `\pgfmathparse` è il comando che valuta l'⟨*espressione*⟩ sui valori attribuiti.

Per esempio, si può definire la funzione settore seno iperbolico nel modo seguente:

```
\pgfmathdeclarefunction{arsinh}{1}{\pgfmathparse{ln(#1 + sqrt(#1^2 + 1))}}
```

e la funzione media aritmetica tra due valori così:

```
\pgfmathdeclarefunction{media}{2}{\pgfmathparse{(#1 + #2)/2}}
```

per poterle utilizzare in maniera del tutto analoga a quelle predefinite:

```
{arsinh(2)+media(-1.5,4.5)}
```

Il risultato dell'⟨*espressione*⟩ è sempre un numero adimensionale, ma nel corso della valutazione tutti i termini vengono convertiti in punti. Poiché il motore matematico di TikZ si basa sulle dimensioni di TeX che hanno in 16 383,999 99 pt il valore massimo (circa 575,8 cm), ogni risultato di un'espressione matematica, anche parziale, *non deve* superare in valore assoluto il numero 16 383,999 99. Questo limite si estende a tutti i numeri e a tutte le dimensioni trattate dal pacchetto.

2.4 Percorsi

In TikZ un *percorso* è una successione di linee (segmenti, spezzate, curve aperte o chiuse) non necessariamente contigue, come quello mostrato nella figura 4 nella pagina successiva. Il comando generale per definire un percorso è `\path`, che presenta la seguente sintassi:

```
\path [⟨opzioni⟩] ⟨istruzioni⟩;
```

Si noti che:

- le ⟨*opzioni*⟩, da separare con la virgola se più d'una, agiscono solo sulle ⟨*istruzioni*⟩ che le seguono;
- ogni istruzione, che può svilupparsi anche su più righe per esigenze di spazio o per maggiore ordine nel codice, corrisponde a una diversa *linea*, che viene realizzata rispettando l'ordine di scrittura nel file sorgente;

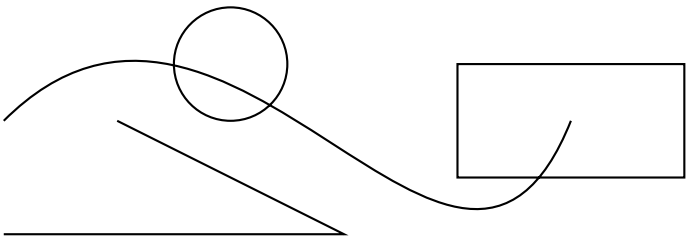


Figura 4: Esempio di percorso elaborato in Ti*k*Z.

- il punto e virgola deve terminare *obbligatoriamente* la sequenza di istruzioni.

L'istruzione più semplice corrisponde allo spostamento dall'ultima posizione nel percorso, detta *posizione corrente*, in un altro punto eventualmente differente, che diventa la posizione corrente per l'istruzione successiva, e si realizza semplicemente dichiarando le coordinate della destinazione nel percorso. Per esempio, il comando `\path (1,0);` indica di spostarsi nel punto di coordinate $(1, 0)$. Ogni percorso comincia normalmente con un'istruzione di spostamento, che di fatto definisce la prima posizione corrente. Di qui in avanti, se non diversamente specificato, si assume che la posizione corrente all'inizio di un'istruzione sia il punto finale della linea precedente.

Il comando `\path` definisce il percorso e aggiorna dimensioni e posizione della tela, ma *non* realizza il disegno a meno che non lo si specifichi tra le opzioni, perciò di fatto si usa raramente. Le opzioni per disegnare, riempire e sfumare un percorso sono rispettivamente `draw`, `fill` e `shade`. Per ciascuna di esse Ti*k*Z definisce una scorciatoia di più agile utilizzo.

3 DISEGNARE IL PERCORSO

Il comando per disegnare un percorso è `\draw`, scorciatoia di `\path [draw]`, dal quale eredita la sintassi:

```
\draw [⟨opzioni⟩] ⟨istruzioni⟩;
```

Esso permette di tracciare direttamente segmenti, rettangoli, circonferenze, ellissi e loro archi, curve di Bézier, grafici di funzioni e, utilizzando le librerie, perfino forme geometriche più elaborate.

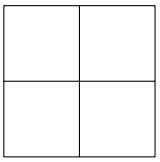
Si noti che, se non ridefinite attraverso le ⟨*opzioni*⟩, le caratteristiche del tratto (colore, spessore, eccetera) sono quelle predefinite. Proprio in virtù di tali caratteristiche, la differenza che passa tra due disegni realizzati l'uno con un comando diverso per ogni istruzione e l'altro con un unico comando per tutte le istruzioni è la stessa che corre tra due schizzi a mano libera eseguiti l'uno con un gesto differente per ogni elemento e l'altro con un unico gesto. Perciò, specie in presenza di linee contigue, si consiglia di definire un singolo percorso ogni volta che è possibile.

A lavoro in corso, può essere molto utile, ma non obbligatorio, aiutarsi con una griglia, che di solito si rimuove a disegno completato. L'istruzione

```
grid [⟨opzioni⟩] (⟨vertice finale⟩)
```

definisce la griglia rettangolare di vertici opposti coincidenti con la posizione corrente e il ⟨*vertice finale*⟩, individuato dalle sue coordinate con uno dei metodi esposti nel paragrafo 2.2 a pagina 4. Eccone un esempio:

```
\begin{tikzpicture}
\draw (0,0) grid (2,2);
\end{tikzpicture}
```



Per ottenere linee di colore grigio anziché nero si può utilizzare l'opzione `help lines`, come si è fatto in tutto quest'articolo.

Il *passo*, cioè la larghezza delle maglie della griglia, è pari a 1 cm per impostazione predefinita, ma lo si può modificare con la chiave `step`, da dichiarare indifferentemente tra le opzioni della griglia o del percorso, che presenta la sintassi seguente:

```
step=⟨valore⟩
```

dove ⟨*valore*⟩ può essere il passo oppure il suo valore numerico, espresso come multiplo o frazione dell'unità di misura degli assi coordinati, come mostrano i primi esempi del paragrafo 3.1. Poiché la griglia serve a individuare più facilmente i punti notevoli del disegno, il passo dovrebbe essere scelto in base alle dimensioni del disegno stesso e alla posizione dei suoi elementi.

Si noti che negli esempi di quest'articolo l'istruzione `grid` è stata ridefinita in maniera invisibile, in modo che il lettore possa individuare senza difficoltà i punti del disegno tramite etichette di riferimento. Il paragrafo 7.2 a pagina 44 spiega come realizzare per conto proprio una semplice griglia numerata con le istruzioni descritte nelle prossime sezioni.

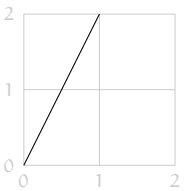
3.1 Segmenti

L'istruzione

```
-- (⟨punto finale⟩)
```

nella quale l'operatore `--` è costituito da due trattini in successione, definisce il *segmento* che unisce la posizione corrente con il ⟨*punto finale*⟩. Eccola all'opera:

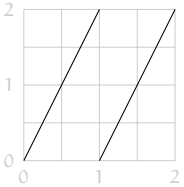
```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (0,0) -- (1,2);
\end{tikzpicture}
```



Si osservi che la griglia è tracciata *sotto* il segmento, perché le istruzioni vengono eseguite seguendo l'ordine di scrittura.

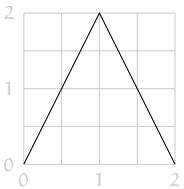
Un percorso appena più elaborato è costituito da più istruzioni all'interno dello stesso comando. Così, per disegnare due segmenti paralleli si può scrivere:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid [step=5mm] (2,2);
\draw (0,0) -- (1,2) (1,0) -- (2,2);
\end{tikzpicture}
```



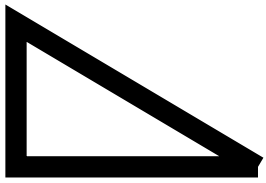
e per disegnare una spezzata:

```
\begin{tikzpicture}
\draw [help lines, step=0.5] (0,0) grid (2,2);
\draw (0,0) -- (1,2) -- (2,0);
\end{tikzpicture}
```

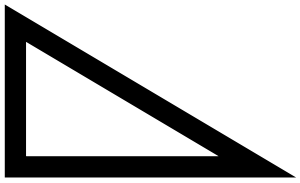


Si noti che i due segmenti vengono raccordati come se le linee fossero tracciate con un unico gesto.

Per disegnare una spezzata con i lati paralleli agli assi coordinati, anziché `--` si può usare l'operatore `-|` (o `|-`). In particolare, `-|` definisce la spezzata con il punto iniziale sul lato parallelo all'asse delle ascisse e quello finale sul lato parallelo all'asse delle ordinate, mentre `|-` agisce nel modo contrario:



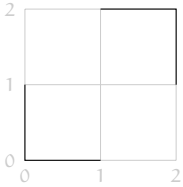
(a) Linea chiusa ottenuta ripetendo le coordinate iniziali.



(b) Linea chiusa ottenuta con l'operatore `-- cycle`.

Figura 5: Comportamento dell'operatore `-- cycle`.

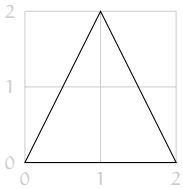
```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (1,2) -| (2,1)
      (0,1) |- (1,0);
\end{tikzpicture}
```



Si osservi che, quando la sequenza di istruzioni si sviluppa su più righe, soltanto l'*ultima* di esse va terminata con il punto e virgola.

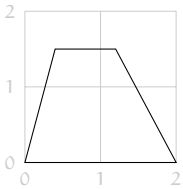
Per disegnare un percorso chiuso, anziché ripetere le coordinate del punto iniziale va usata l'istruzione `-- cycle`, che garantisce la perfetta chiusura della linea, come mostra la figura 5. Per esempio, un triangolo si disegna in questo modo:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (0,0) -- (1,2) -- (2,0) -- cycle;
\end{tikzpicture}
```



e un trapezio così:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (0,0) -- (0.4,1.5) -- (1.2,1.5) --
      (2,0) -- cycle;
\end{tikzpicture}
```



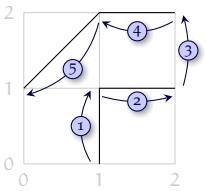
3.2 Coordinate relative

Come si è accennato nel paragrafo 2.2 a pagina 4, oltre che rispetto all'origine del sistema di riferimento si può individuare la posizione di un punto anche rispetto a un altro punto del percorso già definito (che qui si chiamerà *cardine*), operando una traslazione *temporanea* del sistema di riferimento. Il cardine è la posizione corrente al termine dell'ultima istruzione eseguita all'interno dello stesso comando nella quale essa sia espressa in coordinate assolute, cioè rispetto all'origine, o tramite l'opzione `turn` descritta di seguito, o ancora tramite la sintassi

`++(`⟨*coordinate*⟩`)`

che esprime le ⟨*coordinate*⟩ del punto finale dell'istruzione in esecuzione rispetto alla posizione del cardine, *che viene aggiornata*: il cardine dell'istruzione successiva, cioè, coincide con la sua posizione corrente. Un esempio chiarirà le idee. Il codice

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (1,0) -- ++(0,1) -- ++(1,0)
      ++(90:1) -- (1,2) -- (0,1);
\end{tikzpicture}
```



va letto così, (a ogni istruzione dell'elenco si è associata una freccia nel disegno, non desumibile dalla lettura del codice sorgente, esclusivamente per fini didattici):

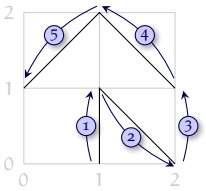
1. `(1,0) -- ++(0,1)` disegna il segmento da $(1,0)$ a $(1,1)$;
2. `++(0,1) -- ++(1,0)` disegna il segmento da $(1,1)$ a $(2,1)$;
3. `++(1,0) ++(90:1)` si sposta da $(2,1)$ in $(2,2)$;
4. `++(90:1) -- (1,2)` disegna il segmento da $(2,2)$ a $(1,2)$ (coordinate assolute);
5. `(1,2) -- (0,1)` disegna il segmento da $(1,2)$ a $(0,1)$.

La sintassi

`+(`⟨*coordinate*⟩`)`

esprime le ⟨*coordinate*⟩ del punto finale dell'istruzione in esecuzione rispetto alla posizione del cardine, *che in questo caso non viene aggiornata*: il cardine dell'istruzione successiva, cioè, coincide con quello dell'istruzione appena eseguita. L'esempio seguente

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (1,0) -- +(0,1) -- ++(1,0)
      +(90:1) -- (1,2) -- (0,1);
\end{tikzpicture}
```



contiene le stesse coordinate dell'esempio precedente e va letto così:

1. `(1,0) -- +(0,1)` disegna il segmento da $(1,0)$ a $(1,1)$: cardine $(1,0)$;
2. `+(0,1) -- ++(1,0)` disegna il segmento da $(1,1)$ a $(2,0)$, perché il cardine è ancora il punto di coordinate $(1,0)$;
3. `++(1,0) +(90:1)` si sposta da $(2,0)$ in $(2,1)$, perché il cardine $(2,0)$ è stato aggiornato al termine dell'istruzione precedente;
4. `+(90:1) -- (1,2)` disegna il segmento da $(2,1)$ a $(1,2)$ (coordinate assolute);
5. `(1,2) -- (0,1)` disegna il segmento da $(1,2)$ a $(0,1)$: il cardine è il punto di coordinate $(1,2)$ aggiornato al termine dell'istruzione precedente.

Indipendentemente dalla posizione del cardine, la sintassi

`([turn] ⟨`φ`⟩:⟨`ρ`⟩)`

esprime le coordinate del punto finale dell'istruzione in esecuzione nel sistema di riferimento polare generalizzato in cui il polo coincide con la posizione corrente e l'asse polare con la tangente nel polo al percorso, orientata in modo da costituirne il naturale prolungamento, come mostra la figura 6 nella pagina successiva. L'espressione non funziona correttamente quando non è possibile determinare la direzione della tangente al percorso nella posizione corrente o quando viene usata in combinazione con una delle altre due sintassi per le coordinate relative. Si noti che la posizione del cardine *viene aggiornata*: il cardine dell'istruzione successiva coincide perciò con la sua posizione corrente. Il codice

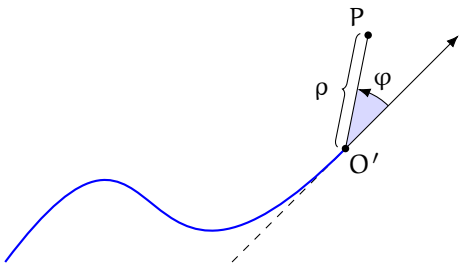
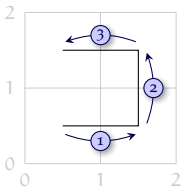


Figura 6: Sistema di riferimento utilizzato dall'opzione `turn`: O′ è la posizione corrente e P è il punto finale dell'istruzione in esecuzione.

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (0.5,0.5) -- (1.5,0.5) --
      ([turn] 90:1) -- ++(-1,0);
\end{tikzpicture}
```



va letto così:

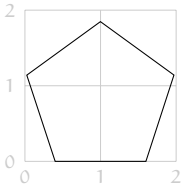
1. `(0.5,0.5) -- (1.5,0.5)` disegna il segmento da $(0.5, 0.5)$ a $(1.5, 0.5)$: il cardine è il punto di coordinate $(1.5, 0.5)$;

2. `(1.5,0.5) -- ([turn] 90:1)` disegna il segmento da $(1.5, 0.5)$ a $(1.5, 1.5)$: il cardine è il punto di coordinate $(1.5, 1.5)$;

3. `([turn] 90:1) -- +(-1,0)` disegna il segmento da $(1.5, 1.5)$ a $(0.5, 1.5)$: il cardine è ancora il punto di coordinate $(1.5, 1.5)$.

Nell'esempio seguente si è disegnato un pentagono regolare di lato 1,2 cm, sapendo che ogni angolo esterno è di $180° - 108° = 72°$:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (0.4,0) -- +(0:1.2) -- ([turn] 72:1.2) --
      ([turn] 72:1.2) -- ([turn] 72:1.2) -- cycle;
\end{tikzpicture}
```



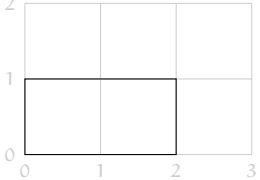
3.3 Rettangoli

L'istruzione

```
rectangle (⟨vertice finale⟩)
```

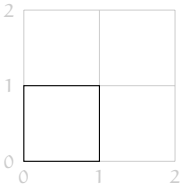
definisce il rettangolo con i lati paralleli agli assi coordinati, con un vertice nella posizione corrente e quello opposto nel ⟨*vertice finale*⟩, che diventa la posizione corrente dell'istruzione successiva. La scelta dei vertici è indifferente, purché siano opposti. Eccola all'opera:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (0,0) rectangle (2,1);
\end{tikzpicture}
```



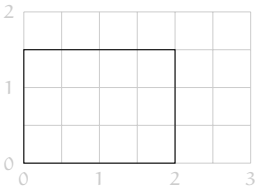
Con la stessa istruzione si può disegnare anche un quadrato:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (0,1) rectangle (1,0);
\end{tikzpicture}
```



Per disegnare un rettangolo con i lati di lunghezza assegnata è preferibile utilizzare la sintassi delle coordinate relative, come mostra l'esempio seguente per il rettangolo di lati 2 cm e 1,5 cm:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid [step=0.5] (3,2);
\draw (0,0) rectangle ++(2cm,1.5cm);
\end{tikzpicture}
```



Si noti che non è necessario dichiarare le unità di misura.

3.4 Circonferenze ed ellissi

L'istruzione

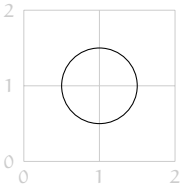
```
circle [radius=⟨raggio⟩]
```

definisce la circonferenza di ⟨*raggio*⟩ assegnato con centro nella posizione corrente. Essa è del tutto equivalente all'istruzione

```
circle (⟨raggio⟩)
```

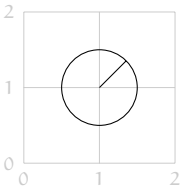
sconsigliabile perché meno leggibile e coerente, ma più compatta e perciò regolarmente utilizzata in questo documento. Eccola all'opera per disegnare una circonferenza con centro nel punto $(1, 1)$ e raggio di 5 mm:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (1,1) circle (5mm);
\end{tikzpicture}
```



La posizione corrente rimane invariata al termine dell'istruzione, perciò con il codice

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (1,1) circle (0.5) -- +(45:0.5);
\end{tikzpicture}
```



si disegna un raggio partendo direttamente dal centro. Si noti che, non essendo specificata, l'unità di misura del raggio è il centimetro per impostazione predefinita.

L'istruzione

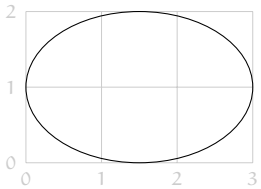
```
ellipse [x radius=⟨semiasse x⟩, y radius=⟨semiasse y⟩]
```

definisce l'ellisse con gli assi paralleli agli assi coordinati e centro nella posizione corrente, della quale sono assegnate le lunghezze dei due semiassi. In particolare, il ⟨*semiasse x*⟩ è la lunghezza del semiasse parallelo all'asse delle ascisse e il ⟨*semiasse y*⟩ quella del semiasse parallelo all'asse delle ordinate. Anche `ellipse` conosce una sintassi alternativa, per la quale valgono le osservazioni appena esposte su `circle`:

```
ellipse (⟨semiasse x⟩ and ⟨semiasse y⟩)
```

Eccola all'opera per disegnare un'ellisse con centro nel punto (1,5 ; 1) e semiassi orizzontale e verticale di 1,5 cm e 1 cm rispettivamente:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (1.5,1) ellipse (1.5 and 1);
\end{tikzpicture}
```



Anche in questo caso la posizione corrente rimane invariata al termine dell'istruzione. Si noti che `circle` e `ellipse` sono sinonimi, perciò permettono di disegnare indifferentemente circonferenze o ellissi a seconda della sintassi utilizzata.

3.5 Archi

Archi di circonferenza e di ellisse

L'istruzione

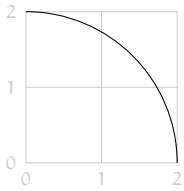
```
arc [start angle=⟨angolo di partenza⟩, end angle=⟨angolo d'arrivo⟩, radius=⟨raggio⟩]
```

definisce l'arco di circonferenza di ⟨*raggio*⟩ assegnato che parte dalla posizione corrente, per il quale gli angoli di partenza e d'arrivo sono le coordinate angolari dei due estremi rispetto al *centro di curvatura*. In questo documento, tuttavia, si è preferito utilizzare la forma compatta

```
arc (⟨angolo di partenza⟩:⟨angolo d'arrivo⟩:⟨raggio⟩)
```

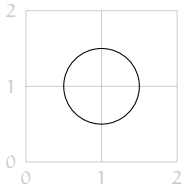
L'esempio seguente traccia l'arco di 2 cm di raggio, che parte da (2, 0) a est (0°) del centro di curvatura e termina a nord (90°) di esso:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (2,0) arc (0:90:2);
\end{tikzpicture}
```



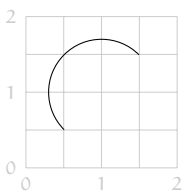
In questo caso, angoli che differiscono per un multiplo intero di 360° *non* sono considerati equivalenti, perciò il codice seguente disegna un'intera circonferenza:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (1.5,1) arc (0:360:0.5);
\end{tikzpicture}
```



Quando è noto il centro di curvatura anziché il punto di partenza, le coordinate relative si rivelano particolarmente utili. L'esempio seguente traccia l'arco con centro di curvatura nel punto di coordinate (1, 1), raggio di 7 mm e angoli di partenza e d'arrivo di 45° e 225° rispettivamente:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid [step=0.5] (2,2);
\draw (1,1) ++(45:7mm) arc (45:225:7mm);
\end{tikzpicture}
```



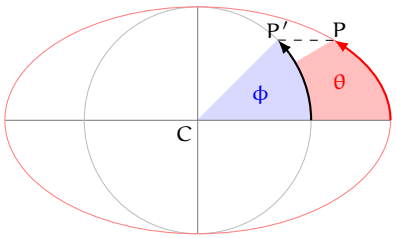


Figura 7: Significato geometrico dell'angolo di partenza dell'arco ellittico.

Analogamente, l'istruzione

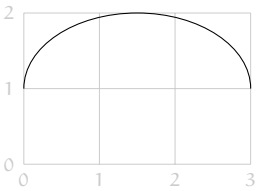
```
arc [start angle=⟨angolo di partenza⟩, end angle=⟨angolo d'arrivo⟩,%
     x radius=⟨semiasse x⟩, y radius=⟨semiasse y⟩]
```

definisce l'arco che parte dalla posizione corrente, appartenente all'ellisse con le caratteristiche geometriche descritte nel paragrafo precedente. Si presti particolare attenzione al fatto che in questo caso gli angoli di partenza e d'arrivo non sono le *coordinate angolari dei due estremi* rispetto al centro dell'ellisse, ma quelle delle *proiezioni dei due estremi*, parallelamente all'asse maggiore, sulla circonferenza concentrica inscritta, come mostra la figura 7 per l'angolo di partenza. Anche per gli archi di ellisse l'alternativa compatta è più pratica:

```
arc (⟨angolo di partenza⟩:⟨angolo d'arrivo⟩:⟨semiasse x⟩ and ⟨semiasse y⟩)
```

Eccone un esempio:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```



Anche qui, angoli che differiscono per un multiplo intero di 360° *non* sono considerati equivalenti.

La relazione che lega gli angoli al centro dell'ellisse con quelli al centro della circonferenza concentrica permette di individuare i valori corretti da inserire nell'istruzione per ottenere il risultato desiderato, se non fossero noti:

$$\tan\theta = \frac{b}{a}\tan\phi$$

dove:

- a e b sono il ⟨*semiasse x*⟩ e il ⟨*semiasse y*⟩ rispettivamente;
- θ e ϕ sono gli angoli al centro dell'ellisse e della circonferenza rispettivamente.

A differenza di quanto si è detto per l'arco circolare, le coordinate relative non sono particolarmente utili quando è noto il centro dell'ellisse anziché l'estremo di partenza e perciò bisogna ricavare la relazione tra le coordinate dei due punti:

$$(x_P, y_P) = (x_C + a\cos\phi,\ y_C + b\sin\phi)$$

A titolo d'esempio, di seguito si disegna l'arco di ellisse con centro nel punto $(1{,}5\,;1)$ e semiassi di 1,5 cm e 1 cm rispettivamente, con gli angoli di partenza e d'arrivo al centro dell'ellisse di 150° e 30° rispettivamente, utilizzando le espressioni coordinate:

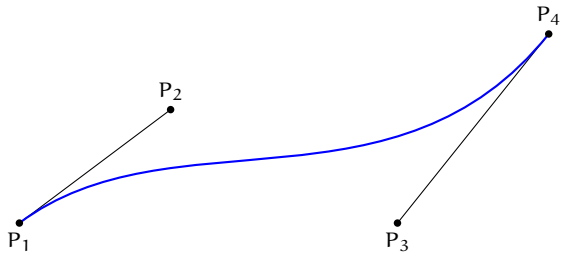
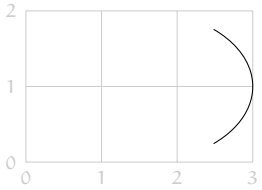


Figura 8: Curva di Bézier cubica.

```
\pgfmathdeclarefunction{map}{1}{%
  \pgfmathparse{atan2(cos(#1), 1.5*sin(#1))}}
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw ({1.5*(1+cos(map(150)))},
       {1+sin(map(150))})
      arc (map(150):map(30):1.5cm and 1cm);
\end{tikzpicture}
```



La funzione personale `map` permette di associare l'angolo al centro della circonferenza all'angolo assegnato.

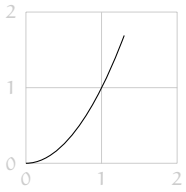
Archi di parabola

L'istruzione

```
parabola (⟨punto finale⟩)
```

definisce l'arco di parabola che parte dalla posizione corrente e termina nel ⟨*punto finale*⟩, con vertice nel punto iniziale e asse di simmetria parallelo all'asse delle ordinate. Eccola all'opera:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (0,0) parabola (1.3, 1.3^2);
\end{tikzpicture}
```

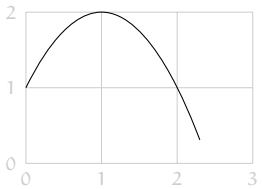


L'istruzione più completa

```
parabola bend (⟨vertice⟩) (⟨punto finale⟩)
```

definisce, purché esista, l'arco di parabola di ⟨*vertice*⟩ e ⟨*punto finale*⟩ assegnati, con asse parallelo all'asse delle ordinate e punto iniziale nella posizione corrente:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (0,1) parabola bend (1,2)
      (2.3, 2-1.3^2);
\end{tikzpicture}
```



3.6 Curve di Bézier

Una *curva di Bézier cubica* è una particolare curva piana definita univocamente da quattro punti (si veda la figura 8): i due estremi della curva P_1 e P_4 e i due *punti di controllo* P_2 e P_3, che godono della proprietà che i segmenti P_1P_2 e P_3P_4 sono tangenti alla curva nei

suoi estremi. TikZ permette di disegnare curve di Bézier in diversi modi, descritti nelle prossime sezioni.

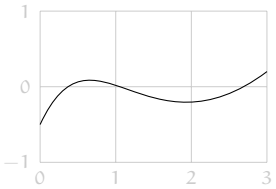
Curve definite dalla posizione dei punti notevoli

Il più completo di essi richiede di dichiarare i quattro punti notevoli nell'istruzione

```
.. controls (⟨punto di controllo 1⟩) and (⟨punto di controllo 2⟩) .. (⟨punto finale⟩)
```

con ovvio significato dei parametri, nella quale il punto iniziale, non espresso, coincide con la posizione corrente. Eccone un esempio:

```
\begin{tikzpicture}
\draw [help lines] (0,-1) grid (3,1);
\draw (0,-0.5) .. controls (0.7,1) and (1.4,-1)
               .. (3,0.2);
\end{tikzpicture}
```



Combinando più linee in un percorso, si possono costruire curve più complesse:

```
\begin{tikzpicture}
\draw [help lines] (0,-1) grid (3,1);
\draw (0,0)   .. controls (0.8,1) and (1.6,-1) ..
      (2.4,0) .. controls ([turn] 0:0.6) and
      (1.9,0.5) .. (0.8,-0.5);
\end{tikzpicture}
```

Curve definite tramite le proprietà geometriche

In alternativa al metodo appena esaminato, si può definire la curva più intuitivamente dichiarandone le proprietà geometriche con una serie di opzioni, attraverso l'istruzione

```
to [⟨opzioni⟩] (⟨punto finale⟩)
```

dove il punto iniziale coincide con la posizione corrente e l'operatore `to` sostituisce `--`. Ogni opzione non dichiarata è sostituita con il proprio valore predefinito, perciò più opzioni compatibili si dichiarano, più accuratamente si potrà controllare la forma della curva. Di seguito si descrivono le opzioni più importanti (per il loro elenco completo si rimanda il lettore alla documentazione del pacchetto).

Le opzioni

```
out=⟨angolo di partenza⟩, in=⟨angolo d'arrivo⟩
```

definiscono le direzioni delle due semirette tangenti alla curva nei suoi estremi, espresse in forma di coordinata angolare rispetto a essi. I valori predefiniti, utilizzati *solo se* c'è almeno un parametro che definisce la curva, sono 45° e 135° rispettivamente. Il codice

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,1);
\draw (0,0) to [out=30, in=120] (2,0);
\end{tikzpicture}
```

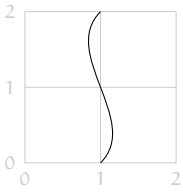
disegna una curva che parte da $(0,0)$ con un ⟨*angolo di partenza*⟩ (`out`) di 30° e termina in $(2,0)$ con un ⟨*angolo d'arrivo*⟩ (`in`) di 120°.

L'opzione

```
relative
```

indica che gli angoli di partenza e d'arrivo vanno considerati rispetto al segmento che unisce i due estremi, cioè assumendo come asse polare la semiretta orientata diretta dal punto iniziale al ⟨*punto finale*⟩:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (1,0) to [out=-45, in=135, relative] (1,2);
\end{tikzpicture}
```

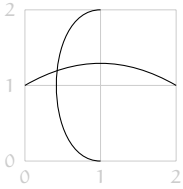


Le opzioni

```
bend left=⟨α⟩, bend right=⟨β⟩
```

sono scorciatoie per `out=`⟨α⟩`, in=180-`⟨α⟩`, relative` e `out=-`⟨β⟩`, in=180+`⟨β⟩`, relative` rispettivamente. Quando il valore non è dichiarato, T*i*k*Z* utilizza in entrambe l'ultimo valore specificato o, per difetto, 30°. Eccole all'opera:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (0,1) to [bend left]     (2,1);
\draw (1,2) to [bend right=90] (1,0);
\end{tikzpicture}
```

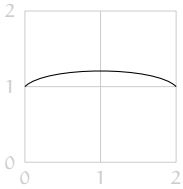


L'opzione

```
looseness=⟨valore⟩
```

è una misura di quanto è "tesa" la curva. Il ⟨*valore*⟩ predefinito è 1:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (0,1) to [looseness=0.5] (2,1);
\end{tikzpicture}
```



Per comprendere l'azione di `looseness`, si confrontino gli ultimi due disegni. Si noti che, non avendo specificato gli angoli di partenza e d'arrivo, T*i*k*Z* ha usato i valori predefiniti, perché la curva è stata definita almeno da un parametro.

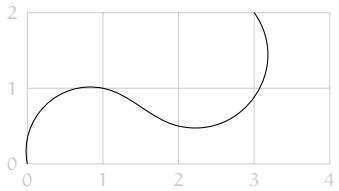
Come al solito, combinando più linee in un percorso si possono costruire curve più complesse:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,1);
\draw (0,0) to [out=90, in=180] (1,1)
      to [out=0, in=180]  (3,0)
      to [out=0, in=-135] (4,1);
\end{tikzpicture}
```

Curve composte mediante l'algoritmo di Hobby

Infine, un ulteriore metodo consiste nell'appoggiarsi alla libreria esterna `hobby`, che per determinare la curva di Bézier composta passante per i punti assegnati mette a disposizione l'algoritmo di Hobby. Ogni coppia di punti consecutivi individua un differente tratto della curva che ha la proprietà di condividere le direzioni delle tangenti negli estremi con i due tratti adiacenti. Per attivare l'algoritmo, dopo averne caricato la libreria come al solito basta dichiarare l'opzione `use Hobby shortcut` e sostituire l'operatore `--` con `..`, come mostra l'esempio seguente:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\draw [use Hobby shortcut]
      (0,0) .. (1,1) .. (2,0.5) .. (3,2);
\end{tikzpicture}
```



La libreria permette di modificare singolarmente i parametri della curva. Si rimanda il lettore a [Stacey, 2014] per approfondire l'argomento.

3.7 Grafici di funzione

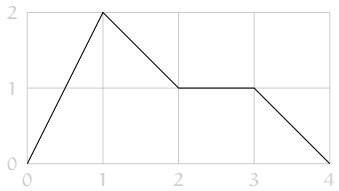
Mediante l'istruzione `plot`, Ti*k*Z permette di disegnare anche grafici di funzione, dove i punti iniziale e finale della linea corrispondono al primo e all'ultimo punto del grafico rispettivamente.

L'istruzione

`plot [`⟨*opzioni*⟩`] coordinates {(`⟨*punto* $_1$⟩`) (`⟨*punto* $_2$⟩`) (`⟨*...*⟩`) (`⟨*ultimo punto*⟩`)}`

definisce la poligonale sui punti assegnati. Eccone un esempio:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\draw plot coordinates {(0,0) (1,2) (2,1)
                        (3,1) (4,0)};
\end{tikzpicture}
```



L'istruzione

`plot [`⟨*opzioni*⟩`] (`⟨*espressioni coordinate*⟩`)`

definisce la poligonale sui punti individuati attraverso una coppia di ⟨*espressioni coordinate*⟩ della forma descritta nel paragrafo 2.3 a pagina 7 e da esprimere in funzione di un parametro (per impostazione predefinita la macro `\x`) che rappresenta la variabile indipendente. Si può personalizzare il grafico mediante opzioni specifiche da indicare *dopo* `plot`. Di seguito se ne descrivono le principali.

Opzioni di plot

L'opzione

`domain=`⟨*valore iniziale*⟩`:`⟨*valore finale*⟩

limita la variabilità del parametro tra il ⟨*valore iniziale*⟩ e il ⟨*valore finale*⟩ (il valore predefinito è `domain=-5:5`).

L'opzione

`samples=`⟨*numero*⟩

imposta il ⟨*numero*⟩ di valori da assegnare al parametro, uniformemente distribuiti nel dominio (il valore predefinito è `samples=25`).

L'opzione

`samples at={`⟨*elenco*⟩`}`

attribuisce direttamente al parametro i valori presi ordinatamente da un ⟨*elenco*⟩ di numeri separati dalla virgola e racchiusi tra parentesi graffe. Si può comprimere l'elenco mediante i puntini di sospensione, come mostrano le due scritture seguenti, del tutto equivalenti:

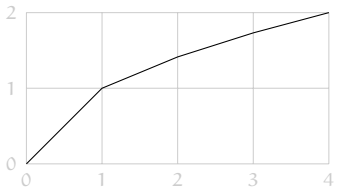
```
{1,1.5,...,3}
{1, 1.5, 2, 2.5, 3}
```

L'opzione

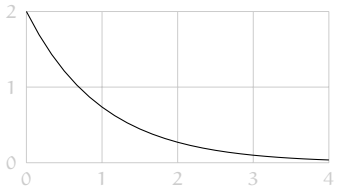
```
variable=⟨macro⟩
```

imposta il nome della macro (\x per impostazione predefinita). Gli esempi seguenti mostrano all'opera le opzioni appena descritte.

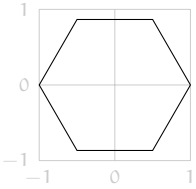
```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\draw plot [domain=0:4, samples=5]
           (\x, {sqrt(\x)});
\end{tikzpicture}
```



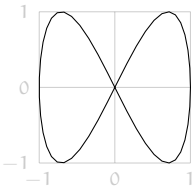
```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\draw plot [domain=0:4] (\x, {2*exp(-\x)});
\end{tikzpicture}
```



```
\begin{tikzpicture}
\draw [help lines] (-1,-1) grid (1,1);
\draw plot [samples at={0,60,...,360}] (\x:1);
\end{tikzpicture}
```



```
\begin{tikzpicture}
\draw [help lines] (-1,-1) grid (1,1);
\draw plot [variable=\t, domain=0:360, samples=50]
           ({cos(\t)}, {sin(2*\t)});
\end{tikzpicture}
```



Si noti che `plot` riconosce le coordinate polari e che permette di disegnare curve parametriche.

Infine, si descrivono altre due opzioni accettate da entrambe le forme dell'istruzione. L'opzione

```
mark=⟨simbolo⟩
```

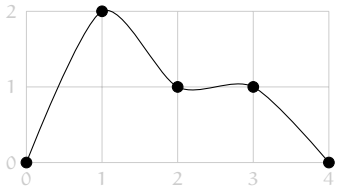
inserisce un marcatore in ogni punto assegnato del grafico. I simboli predefiniti sono `*`, `+` e `x` per ottenere un circoletto pieno, una croce greca e una *x* rispettivamente. La libreria `plotmarks` (descritta nella documentazione del pacchetto) ne fornisce molti altri.

L'opzione

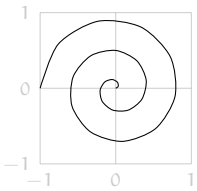
```
smooth
```

anziché con segmenti, raccorda i punti del grafico mediante curve di Bézier cubiche che condividono a coppie consecutive la direzione della tangente negli estremi. Di fatto, si tratta di un ulteriore metodo per disegnare curve di Bézier. Eccone due esempi:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\draw plot [mark=*, smooth] coordinates
           {(0,0) (1,2) (2,1) (3,1) (4,0)};
\end{tikzpicture}
```

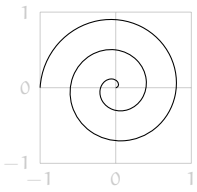


```
\begin{tikzpicture}
\draw [help lines] (-1,-1) grid (1,1);
\draw plot [variable=\t, domain=0:900, smooth]
           (\t: \t/900);
\end{tikzpicture}
```



Una buona alternativa a `smooth`, il cui algoritmo non funziona molto bene con angoli di curvatura superiori a 30°, è il più efficiente algoritmo di Hobby, attivabile con l'opzione `hobby` dell'omonima libreria. Si confronti l'ultimo esempio con il seguente:

```
\begin{tikzpicture}
\draw [help lines] (-1,-1) grid (1,1);
\draw plot [variable=\t, domain=0:900, hobby]
           (\t: \t/900);
\end{tikzpicture}
```



Si ricordi che Ti*k*Z può contare su potenza di calcolo e numero di funzioni matematiche predefinite relativamente limitati, il che potrebbe costituire un problema quando il grafico richieda calcoli particolarmente onerosi. In tal caso, l'istruzione `plot` può avvalersi efficacemente del motore di calcolo del programma gnuplot (da installare a parte), come descritto nella documentazione del pacchetto, che spiega anche come tracciare un grafico a partire da una sequenza di dati raccolti in un file esterno.

Grafici troppo grandi e grafici elaborati

Sottovalutare le dimensioni di un grafico realizzato con Ti*k*Z potrebbe causare qualche problema: se non si cambiano le unità di misura, due punti di coordinate $(0,0)$ e $(10,100)$ richiedono una tela di almeno $10\,\text{cm} \times 1\,\text{m}$, chiaramente troppo grande per un comune foglio di formato A4. Al prezzo di una sintassi un po' macchinosa, la libreria interna `datavisualization` permette di assegnare esplicitamente le dimensioni del diagramma e di semplificarne la personalizzazione (aggiungere gli assi, la legenda, eccetera) che potrebbe rivelarsi facilmente laboriosa se realizzata con le sole istruzioni di base. In alternativa, il pacchetto `pgfplots` costituisce un'ottima soluzione dalla sintassi più amichevole per entrambi i problemi e permette di disegnare anche grafici tridimensionali (si consulti [Pantieri e Gordini, 2012a] per gli approfondimenti).

3.8 Nodi

Un *nodo* è una forma (di solito un cerchio o un rettangolo) che può contenere un testo al proprio interno: la sua funzione primaria, infatti, è quella di aggiungere scritte al disegno. Nella sua forma più semplice, la sintassi di un nodo è

```
\node [opzioni] (⟨nome⟩) at (⟨punto⟩) {⟨testo⟩};
```

dove:

- `\node` è l'abbreviazione di `\path [node];`
- ⟨*nome*⟩ è un'etichetta di riferimento *facoltativa* che può contenere lettere, numeri e spazi ma *non* segni d'interpunzione;
- ⟨*punto*⟩ è per impostazione predefinita la posizione del centro del nodo;
- il ⟨*testo*⟩ va racchiuso *obbligatoriamente* tra parentesi graffe *anche se vuoto* e può consistere di testo puro (si veda anche il paragrafo 7.3 a pagina 45) o di espressioni matematiche.

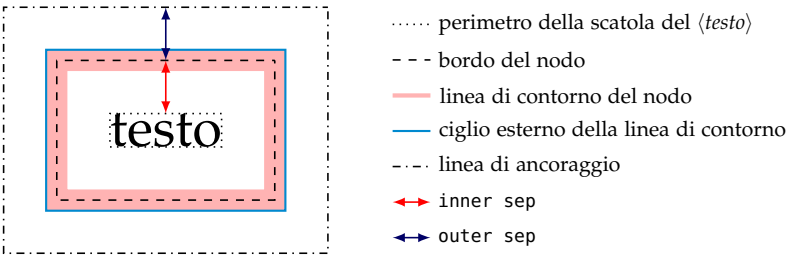


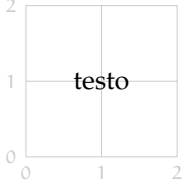
Figura 9: Elementi di un nodo.

Si noti ancora che, per impostazione predefinita:

- un nodo ha forma rettangolare, ma si possono ottenere nodi circolari con l'opzione `circle`;
- un nodo è invisibile, a meno che non si specifichi di volerne disegnare il contorno o di riempirlo con le opzioni opportune;
- la dimensione del nodo si adatta automaticamente al ⟨*testo*⟩ che contiene.

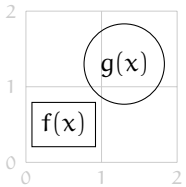
Ecco due esempi che mostrano quanto si è appena descritto. Un nodo invisibile intorno al proprio contenuto (una parola):

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\node at (1,1) {testo};
\end{tikzpicture}
```



e due nodi di forme diverse (con dentro espressioni matematiche):

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\node [draw] at (0.5,0.5) {$f(x)$};
\node [draw, circle] at (1.3,1.3) {$g(x)$};
\end{tikzpicture}
```



La figura 9 illustra in dettaglio gli elementi di un nodo. Si possono riconoscere:

- la scatola del ⟨*testo*⟩, cioè il rettangolo più interno che racchiude la parola *testo*;
- il *bordo* del nodo, sul quale viene tracciata la *linea di contorno*, qui ingrossata per maggiore chiarezza;
- la *linea di ancoraggio* ideale (spiegata nelle prossime sezioni), che per impostazione predefinita coincide con il ciglio esterno della linea di contorno (in celeste) e qui distanziata per maggiore chiarezza.

Opzioni dei nodi

Il comando `\node` accetta diverse opzioni, raccolte nella documentazione del pacchetto, a cui si rimanda il lettore. Di seguito se ne descrivono le più importanti.

L'opzione

```
font=⟨dichiarazioni⟩
```

permette di personalizzare lo stile del font in uso mediante le opportune ⟨*dichiarazioni*⟩ standard di LaTeX, eventualmente racchiudendole tra parentesi graffe. L'opzione non influenza le eventuali dimensioni degli elementi del nodo espresse in rapporto alle dimensioni del font usato (`em` e `ex`), al contrario dell'opzione `node font`, che ne condivide la finalità.

L'opzione

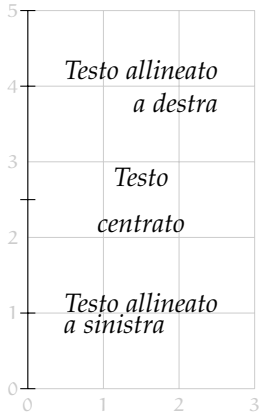
`align=`⟨*tipo di allineamento*⟩

specifica il tipo di allineamento per il ⟨*testo*⟩ e accetta i seguenti valori:

- `left`, `right`, `center` allineano il ⟨*testo*⟩ a sinistra, a destra e al centro rispettivamente, eventualmente sillabandolo;
- `flush left`, `flush right` e `flush center` agiscono come i valori precedenti ma senza sillabazione;
- `justify` giustifica il testo.

Si noti che si può mandare a capo il testo in un punto qualunque mediante il solito `\\` solo *dopo* aver dichiarato esplicitamente il tipo di allineamento desiderato. Inoltre, si può aumentare (o ridurre) l'avanzamento tra la riga terminata da `\\` e quella successiva specificando tra parentesi quadre la misura positiva (o negativa) desiderata. Ecco un esempio:

```
\begin{tikzpicture} [font=\itshape]
\draw [help lines] (0,0) grid (3,5);
\draw (0,0)    -- (0,5)
      (-0.1,0) -- (0.1,0)
      (-0.1,1) -- (0.1,1)
      (-0.1,2.5) -- (0.1,2.5)
      (-0.1,4) -- (0.1,4)
      (-0.1,5) -- (0.1,5);
\node [align=right] at (1.5,4)
      {Testo allineato\\ a destra};
\node [align=center] at (1.5,2.5)
      {Testo\\[1ex] centrato};
\node [align=left] at (1.5,1)
      {Testo allineato\\[-1ex] a sinistra};
\end{tikzpicture}
```



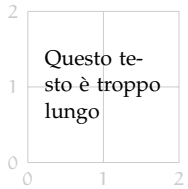
Si osservi che la chiave `font` è stata assegnata all'ambiente `tikzpicture` nel suo complesso, quindi agisce su *tutti* gli elementi dell'ambiente sui quali può aver effetto, cioè su tutti i nodi.

L'opzione

`text width=`⟨*larghezza*⟩

imposta a una dimensione fissa la larghezza della scatola del ⟨*testo*⟩ all'interno del nodo. A meno che non si intervenga sulla chiave `align`, il testo è allineato a sinistra e sillabato, come nell'esempio seguente:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\node [font=\footnotesize, text width=15.5mm]
      at (1,1)  {Questo testo è troppo lungo};
\end{tikzpicture}
```



Quando si assegna esplicitamente una larghezza alla scatola del testo, si può usare l'operatore `\\` per forzare un'interruzione di linea.

Infine le opzioni

`minimum width=`⟨*dimensione*⟩`, minimum height=`⟨*dimensione*⟩`, minimum size=`⟨*dimensione*⟩

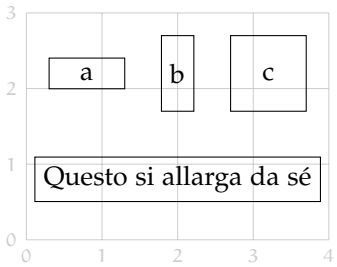
permettono di impostare rispettivamente la larghezza minima e l'altezza minima del bordo del nodo (`minimum size` le imposta contemporaneamente) in modo tale che il nodo non sia

Tabella 2: Ancore notevoli di un nodo espresse come valori della chiave `anchor` (colonne dispari) e come opzioni (colonne pari).

Valore di `anchor`	Scorciatoia	Valore di `anchor`	Scorciatoia
`east`	`left`	`west`	`right`
`north east`	`below left`	`south west`	`above right`
`north`	`below`	`south`	`above`
`north west`	`below right`	`south east`	`above left`
`center`		`base`	

più piccolo di una certa dimensione, indipendentemente dal ⟨*testo*⟩, come mostra l'esempio seguente:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,3);
\node [draw, minimum width=1cm]  at (0.8,2.2) {a};
\node [draw, minimum height=1cm] at (2,2.2)   {b};
\node [draw, minimum size=1cm]   at (3.2,2.2) {c};
\node [draw, minimum width=1cm]  at (2,0.8)
      {Questo si allarga da sé};
\end{tikzpicture}
```



Ancore

Un'*àncora* è un punto sulla linea di ancoraggio o all'interno del nodo, che si può far corrispondere al ⟨*punto*⟩ assegnando l'opportuno valore alla chiave `anchor` con la sintassi:

```
anchor=⟨direzione⟩
```

dove ⟨*direzione*⟩ può essere:

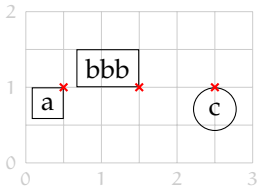
- un numero, e in tal caso rappresenta la coordinata angolare che individua un punto della linea di ancoraggio rispetto al centro del nodo;
- un'espressione analoga a quelle mostrate nella tabella 2 e spiegate qui di seguito.

La maggior parte dei nodi possiede almeno dieci ancore notevoli alle quali sono associati altrettanti valori della chiave `anchor`:

- Un'ancora centrale (predefinita), a cui è associato il valore `center`.
- Un'ancora all'intersezione della linea di base del testo con la sua perpendicolare per il centro del nodo, a cui è associato il valore `base`.
- Otto ancore sulla linea di ancoraggio, corrispondenti alle direzioni della rosa dei venti rispetto al centro del nodo, a cui sono associati i rimanenti valori della tabella 2. Per ognuno di essi è disponibile anche una scorciatoia in forma di *opzione* (si veda la tabella appena menzionata) che indica intuitivamente la posizione del centro del nodo rispetto al ⟨*punto*⟩. Quindi `left`, equivalente a `anchor=east` o a `anchor=0`, indica che il centro del nodo si trova *a sinistra* del ⟨*punto*⟩.

Nei nodi rettangolari, le ancore associate alle quattro direzioni della rosa diverse dai punti cardinali corrispondono per impostazione predefinita ai quattro vertici del rettangolo celeste nella figura 9 a pagina 23. Ecco un esempio in cui si sono evidenziate le ancore per chiarezza:

```
\begin{tikzpicture}
\draw [help lines, step=0.5] (0,0) grid (3,2);
\node [draw, anchor=45]         at (0.5,1) {a};
\node [draw, anchor=south east] at (1.5,1) {bbb};
\node [circle, draw, below]     at (2.5,1) {c};
\end{tikzpicture}
```

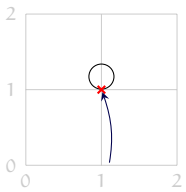


Se usate come *chiavi*, le scorciatoie appena viste accettano un valore facoltativo che quantifica lo spostamento nella direzione indicata con la sintassi

⟨*scorciatoia*⟩=⟨*distanza*⟩

Per esempio, `above=1cm` (o `above=-1cm`) àncora il nodo in modo che il lato inferiore della linea di ancoraggio si trovi 1 cm sopra (o sotto) al ⟨*punto*⟩:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\node [circle, draw, above=1cm] (a) at (1,0) {};
\end{tikzpicture}
```



L'opzione

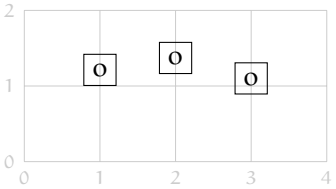
`outer sep=`⟨*distanza*⟩

(corrispondente alla freccia blu nella figura 9 a pagina 23) imposta la ⟨*distanza*⟩ tra il bordo del nodo e la linea di ancoraggio. Si noti che:

- il valore predefinito, pari a metà spessore di linea, fa coincidere la linea di ancoraggio con il ciglio esterno della linea di contorno, in modo che le ancore perimetrali si trovino sul rettangolo celeste della figura 9 a pagina 23;
- una ⟨*distanza*⟩ pari a zero può essere utile per collocare correttamente le ancore in un nodo riempito ma non tracciato;
- un valore negativo fa rientrare le ancore nel nodo.

Il prossimo esempio mostra all'opera quanto si è appena descritto:

```
\begin{tikzpicture} [above]
\draw [help lines] (0,0) grid (4,2);
\node [draw]                    at (1,1) {o};
\node [draw, outer sep=1ex]     at (2,1) {o};
\node [draw, outer sep=-0.3em] at (3,1) {o};
\end{tikzpicture}
```

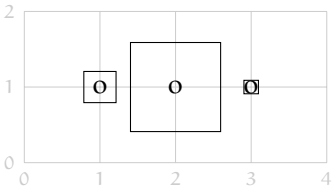


L'opzione

`inner sep=`⟨*distanza*⟩

imposta la ⟨*distanza*⟩ tra il perimetro della scatola del ⟨*testo*⟩ e il bordo del nodo, pari a 0,3333 em per impostazione predefinita (corrisponde alla freccia rossa nella figura 9 a pagina 23), permettendo di aumentare le dimensioni del nodo rispetto a quelle del suo contenuto:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\node [draw]                 at (1,1) {o};
\node [draw, inner sep=5mm] at (2,1) {o};
\node [draw, inner sep=0pt] at (3,1) {o};
\end{tikzpicture}
```



Nodi coordinata, etichette e forme

Un nodo particolare è il *nodo coordinata*, che si può intendere come un nodo senza dimensione (un punto geometrico, per capirci). Serve principalmente come *alias* delle coordinate di un punto da richiamare successivamente nel disegno. La sua sintassi è:

```
\coordinate [opzioni] (⟨nome⟩) at (⟨punto⟩);
```

dove:

- `\coordinate` è l'abbreviazione di `\path [coordinate];`
- il ⟨*nome*⟩ della coordinata è un'etichetta di riferimento *obbligatoria* che può contenere lettere, numeri e spazi ma *non* segni d'interpunzione.

Si noti che, non avendo dimensione, un nodo coordinata è anche privo del ⟨*testo*⟩ e perciò tutte le opzioni per i nodi fino a qui descritte non hanno alcun effetto su di esso.

Si può assegnare un'etichetta testuale a un nodo, e quindi a un nodo coordinata in particolare, mediante l'opzione `label`, che nella sua forma completa presenta la sintassi:

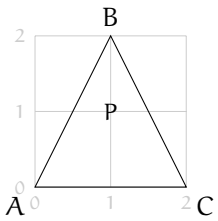
```
label={[⟨opzioni⟩]⟨direzione⟩:⟨testo⟩}
```

dove:

- le ⟨*opzioni*⟩ agiscono solo sul ⟨*testo*⟩ dell'etichetta;
- la ⟨*direzione*⟩, *facoltativa* (compresi i due punti), può essere una delle scorciatoie elencate nella tabella 2 a pagina 25 (il valore predefinito è `above`) oppure `center` o ancora un angolo rispetto al ⟨*punto*⟩;
- se la ⟨*direzione*⟩ non è un angolo, *non* va preceduta da spazi;
- le parentesi graffe si possono omettere se non ci sono ⟨*opzioni*⟩ e se il ⟨*testo*⟩ non contiene virgole e segni di uguaglianza.

L'esempio seguente mostra all'opera quanto si è appena descritto:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\coordinate [label=below left:$A$] (A) at (0,0);
\coordinate [label=$B$]            (B) at (1,2);
\coordinate [label=-45:$C$]        (C) at (2,0);
\coordinate [label={[font=\small]center:$P$}]
                                   (P) at (1,1);
\draw (A) -- (B) -- (C) -- cycle;
\end{tikzpicture}
```



Si osservi che nel percorso il ⟨*nome*⟩ sostituisce il ⟨*punto*⟩ corrispondente. La dichiarazione di più opzioni `label` per uno stesso nodo permette di assegnargli più di un'etichetta di testo.

La libreria quotes permette di semplificare la dichiarazione delle etichette rendendo disponibile la sintassi equivalente (si notino i segni "⟨. . . ⟩", necessari):

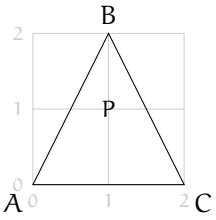
```
"⟨direzione⟩:⟨testo⟩" {⟨opzioni⟩}
```

dove:

- la ⟨*direzione*⟩, *facoltativa* (compresi i due punti), può essere inserita in alternativa tra le ⟨*opzioni*⟩ se espressa con una delle scorciatoie elencate nella tabella 2 a pagina 25;
- il ⟨*testo*⟩ va messo tra parentesi graffe se contiene in generale segni d'interpunzione;
- se più d'una, le ⟨*opzioni*⟩ vanno racchiuse tra parentesi graffe.

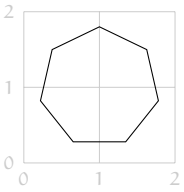
Con la sintassi alternativa, l'esempio precedente si presenta così:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\coordinate ["$A$" below left] (A) at (0,0);
\coordinate ["$B$"]             (B) at (1,2);
\coordinate ["-45:$C$"]         (C) at (2,0);
\coordinate ["center:$P$" font=\small]
                                (P) at (1,1);
\draw (A) -- (B) -- (C) -- cycle;
\end{tikzpicture}
```



Le librerie interne della famiglia shapes mettono a disposizione molte altre forme di nodo. La libreria shapes.geometric, per esempio, permette di utilizzare forme geometriche direttamente nel disegno, come si vede nell'esempio seguente:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\node [regular polygon, regular polygon sides=7,
       minimum size=16mm, draw] at (1,1) {};
\end{tikzpicture}
```



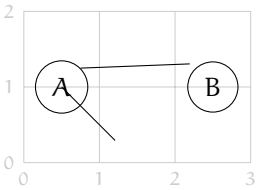
Sistema di riferimento dei nodi

Quando un nodo è definito mediante un ⟨*nome*⟩, ci si può riferire alle sue ancore direttamente nelle coordinate di un percorso mediante la sintassi

```
(⟨nome⟩.⟨direzione⟩)
```

dove ⟨*direzione*⟩, separata dal ⟨*nome*⟩ con il punto (`.`), può essere uno dei valori accettati dalla chiave `anchor` (se un angolo, deve essere espresso in gradi interi). Ecco un esempio:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\node [circle, draw] (A) at (0.5,1) {$A$};
\node [circle, draw, outer sep=1mm]
      (B) at (2.5,1) {$B$};
\draw (A.north east) -- (B.135)
      (A.center) -- +(-45:1);
\end{tikzpicture}
```



Se la ⟨*direzione*⟩ non è specificata, la sintassi

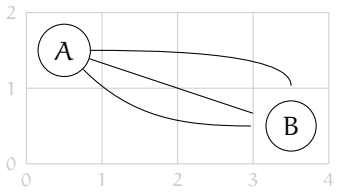
```
(⟨nome⟩)
```

individua il centro del nodo per archi, grafici di funzione e punti di controllo delle curve di Bézier, oppure un punto sulla linea di ancoraggio diverso a seconda dell'istruzione in cui la si adopera. In particolare:

- se l'istruzione definisce un segmento o una spezzata dichiarata con gli operatori `-|` o `|-`, ⟨*nome*⟩ individua il punto che giace sulla linea passante per il centro del nodo;
- se l'istruzione definisce una curva di Bézier a partire dai suoi punti notevoli, ⟨*nome*⟩ individua il punto che giace sul segmento che congiunge il centro del nodo al punto di controllo adiacente;
- se l'istruzione definisce una curva di Bézier attraverso le sue proprietà geometriche, allora ⟨*direzione*⟩ coincide con l'⟨*angolo di partenza*⟩ o l'⟨*angolo di arrivo*⟩ a seconda che il nodo cominci o termini la linea.

Qualche esempio chiarirà le idee:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\node [circle, draw] (A) at (0.5,1.5) {$A$};
\node [circle, draw, outer sep=2mm]
      (B) at (3.5,0.5) {$B$};
\draw (A) -- (B) [out=-45, in=180] (A) to (B)
   (A) .. controls (1.5,1.5) and (3.5,1.5) .. (B);
\end{tikzpicture}
```



Si noti che la sintassi (⟨*nome*⟩) oppure (⟨*nome*⟩.⟨*direzione*⟩) costituisce un ulteriore metodo per specificare le coordinate di un punto, che si aggiunge di fatto a quelli già esposti nel paragrafo 2.2 a pagina 4.

Nodi definiti direttamente in un percorso

Si può aggiungere un nodo direttamente nella posizione corrente di un percorso mediante l'istruzione

```
node [⟨opzioni⟩] (⟨nome⟩) {⟨contenuto⟩}
```

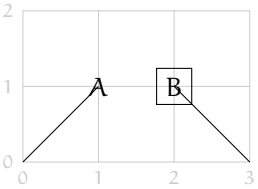
Lo stesso si può fare con un nodo coordinata tramite l'istruzione

```
coordinate [⟨opzioni⟩] (⟨nome⟩)
```

In tal caso, quanto detto finora riferito al ⟨*punto*⟩ si intende riferito alla posizione corrente.

Si noti che i nodi definiti in un percorso *non* ne fanno parte, ma normalmente gli sono aggiunti solo *dopo* averlo tracciato, perciò si possono sovrapporre alle altre linee. Eccone un esempio:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) node {$A$}
      (3,0) -- (2,1) node [draw] {$B$};
\end{tikzpicture}
```



Si osservi che anche se aggiunto a un percorso tracciato, un nodo rimane invisibile a meno di non esplicitarne il disegno (con `draw`, per esempio) tra le sue opzioni.

L'opzione

```
behind path
```

permette di inserire un nodo *dietro* al percorso nel quale è definito, cioè *prima* che questo sia tracciato.

L'opzione

```
pos=⟨valore⟩
```

nella quale il ⟨*valore*⟩ è generalmente un numero compreso tra 0 e 1, permette di collocare il nodo in un punto della linea precedentemente tracciata intermedio tra la posizione attuale (corrispondente a `pos=1`) e la posizione corrente della precedente istruzione (`pos=0`), come mostra l'esempio seguente:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (1,0) -- (0,1) -- (2,1)
      node [pos=0.5, above] {$d$};
\end{tikzpicture}
```

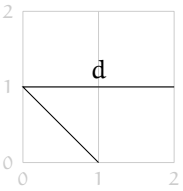
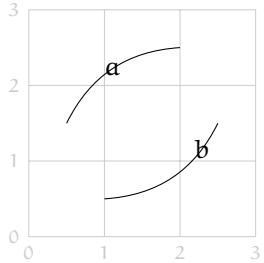


Tabella 3: Spessori di linea predefiniti da TikZ.

Opzione	Spessore	Risultato	Opzione	Spessore	Risultato
`ultra thin`	0,1 pt		`thick`	0,8 pt	
`very thin`	0,2 pt		`very thick`	1,2 pt	
`thin`	0,4 pt		`ultra thick`	1,6 pt	
`semithick`	0,6 pt				

La linea deve essere un segmento, una spezzata definita mediante gli operatori `-|` o `|-`, un arco circolare o ellittico, oppure una curva di Bézier definita attraverso i suoi punti di controllo. Per le curve di Bézier definite mediante le proprietà geometriche, si può dichiarare il nodo subito dopo l'operatore `to` (cioè prima del ⟨*punto finale*⟩). In tal caso, l'impostazione predefinita è `pos=0.5`, come mostra l'esempio seguente:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,3);
\draw [bend left] (0.5,1.5) to node {$a$} (2,2.5)
      [bend right] (1,0.5) to node [pos=0.8]
      {$b$} (2.5,1.5);
\end{tikzpicture}
```

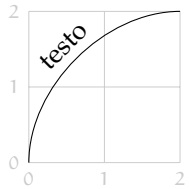


Infine l'opzione

```
sloped
```

ruota automaticamente il nodo in modo tale che il ⟨*testo*⟩ sia parallelo alla tangente alla linea nel punto in cui il nodo è aggiunto (si veda anche il paragrafo 6.3 a pagina 41), come mostra l'esempio seguente:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw (0,0) ..  controls (0,1) and (1,2) .. (2,2)
      node [pos=0.5, sloped, above] {testo};
\end{tikzpicture}
```



Le operazioni che si possono compiere sui nodi sono davvero tante, e molte di esse esulano dagli obiettivi di quest'articolo. Perciò, al lettore che abbia acquisito una discreta padronanza degli argomenti qui trattati e con esigenze che non vi trovino immediato riscontro, si consiglia di consultare la documentazione del pacchetto.

4 PERSONALIZZARE IL TRATTO

Le istruzioni per disegnare il percorso descritte finora non hanno contemplato la possibilità di personalizzare il disegno. Questa sezione mostra come sia possibile modificare le caratteristiche del tratto tramite le opzioni dei comandi.

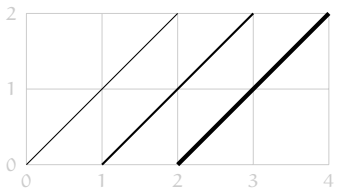
4.1 Spessori di linea

Lo spessore di linea utilizzato da TikZ è di 0,4 pt (circa 0,14 mm) per impostazione predefinita e corrisponde all'opzione `thin` mostrata nella tabella 3, ma lo si può cambiare scegliendo una delle altre opzioni lì raccolte. Il prossimo esempio ne mostra tre all'opera:

Tabella 4: Tipi di linea predefiniti da Ti*k*Z.

Opzione	Risultato	Opzione	Risultato
`solid`	———	`dash dot`	_._._._._
`dotted`	.....	`densely dash dot`	_._._._._
`densely dotted`	.....	`loosely dash dot`	_ . _ . _ .
`loosely dotted`	. . . . .	`dash dot dot`	_.._.._..
`dashed`	_ _ _ _ _	`densely dash dot dot`	_.._.._..
`densely dashed`	_ _ _ _ _	`loosely dash dot dot`	_ . . _ . .
`loosely dashed`	_ _ _ _ _		

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\draw [thin]        (0,0) -- (2,2);
\draw [thick]       (1,0) -- (3,2);
\draw [ultra thick] (2,0) -- (4,2);
\end{tikzpicture}
```

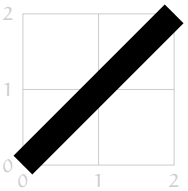


Se ancora non dovessero bastare, si può definire uno spessore a piacere con l'opzione

```
line width=⟨grossezza⟩
```

dove ⟨*grossezza*⟩ è una misura da esprimere con una delle abbreviazioni riconosciute da LaTeX. Eccone un esempio:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw [line width=10pt] (0,0) -- (2,2);
\end{tikzpicture}
```

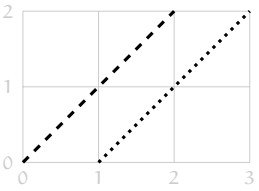


Si osservi che le linee vengono sempre tracciate a cavallo della linea ideale di spessore nullo definita nel percorso.

4.2 Tipi di linea

Per impostazione predefinita, Ti*k*Z traccia il disegno con una linea continua, che si può cambiare scegliendo tra quelle raccolte nella tabella 4. Il prossimo esempio ne mostra un paio all'opera:

```
\begin{tikzpicture} [very thick]
\draw [help lines] (0,0) grid (3,2);
\draw [dashed] (0,0) -- (2,2);
\draw [dotted] (1,0) -- ++(2,2);
\end{tikzpicture}
```



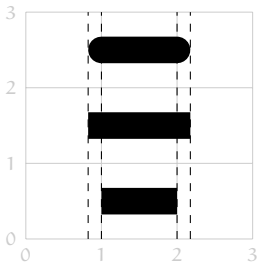
Si noti che l'opzione `very thick` è assegnata all'ambiente `tikzpicture` nel suo complesso e perciò agisce su *tutti* gli elementi del disegno. Ti*k*Z permette di personalizzare anche la sequenza di tratti e punti mediante la chiave `dash pattern` (si veda la documentazione del pacchetto).

L'opzione

```
line cap=⟨stile⟩
```

permette di specificare lo ⟨*stile*⟩ delle estremità delle linee e accetta uno dei seguenti tre valori: `round` ("arrotondato"), `rect` ("rettangolare") e `butt` ("mozzato", stile predefinito). Il prossimo esempio la mostra all'opera:

```
\begin{tikzpicture} [line width=10pt]
\draw [help lines] (0,0) grid (3,3);
\draw [thin, dashed]
      (1,0) -- (1,3)
      ++(-5pt,-3) -- ++(90:3)
      (2,0) -- (2,3)
      ++( 5pt,-3) -- ++(90:3);
\draw [line cap=round] (1,2.5) -- (2,2.5);
\draw [line cap=rect]  (1,1.5) -- (2,1.5);
\draw [line cap=butt]  (1,0.5) -- (2,0.5);
\end{tikzpicture}
```



Si noti che:

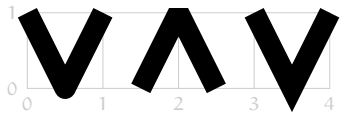
- gli stili `round` e `rect` allungano la linea di metà ⟨*grossezza*⟩ da entrambe le estremità;
- l'opzione `thin` data al secondo percorso scavalca localmente quella data all'ambiente `tikzpicture` per impostare lo spessore delle linee del disegno.

Invece l'opzione

```
line join=⟨stile⟩
```

determina lo ⟨*stile*⟩ del punto di raccordo tra le linee contigue del percorso e accetta uno dei seguenti tre valori: `round` ("arrotondato"), `bevel` ("smussato") e `miter` ("spigoloso", stile predefinito). Eccola in azione:

```
\begin{tikzpicture} [line width=8pt]
\draw [help lines] (0,0) grid (4,1);
\draw [line join=round]
      (0,1) -- (0.5,0) -- (1,1);
\draw [line join=bevel]
      (1.5,0) -- (2,1) -- (2.5,0);
\draw [line join=miter]
      (3,1) -- (3.5,0) -- (4,1);
\end{tikzpicture}
```

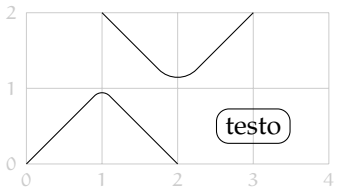


L'opzione

```
rounded corners=⟨misura⟩
```

dove ⟨*misura*⟩ è pari a 4 punti per impostazione predefinita e può essere omesso, permette di raccordare dolcemente le linee contigue di un percorso. La ⟨*misura*⟩ è direttamente collegata al raggio di curvatura del raccordo, come mostra l'esempio seguente:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\draw [rounded corners]
      (0,0) -- (1,1) -- (2,0);
\node [rounded corners=1ex, draw]
      at (3,0.5) {testo};
\draw [rounded corners=10pt]
      (1,2) -- (2,1) -- (3,2);
\end{tikzpicture}
```



4.4 Frecce

L'opzione generale

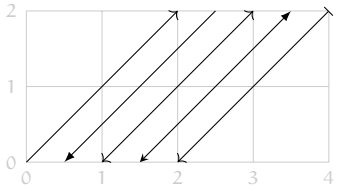
⟨*stile iniziale*⟩-⟨*stile finale*⟩

disegna due punte alle estremità dell'ultima linea di un percorso aperto. Si noti che:

- se uno dei due ⟨*stili*⟩ è omesso, la punta verrà disegnata solo dall'altra parte;
- gli stili predefiniti più utilizzati sono `To`, `latex`, `stealth` e `|` (linea verticale);
- lo stile `>` (`<`, se iniziale) è un *alias* per `To`, a meno che non venga ridefinito dall'utente;
- le frecce predefinite non sporgono oltre l'estremità della linea, per non alterarne la lunghezza;
- la dimensione della punta si accorda automaticamente con lo spessore della linea per impostazione predefinita.

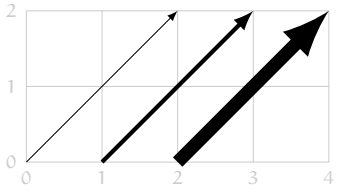
Ecco le opzioni appena descritte all'opera:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\draw [-To] (0,0) -- (2,2);
\draw [latex-] (0.5,0) -- +(2,2);
\draw [<->] (1,0) -- +(2,2);
\draw [stealth-latex] (1.5,0) -- +(2,2);
\draw [<-|] (2,0) -- +(2,2);
\end{tikzpicture}
```



Nell'esempio seguente si è ridefinito l'alias `>`:

```
\begin{tikzpicture} [>=latex]
\draw [help lines] (0,0) grid (4,2);
\draw [->] (0,0) -- (2,2);
\draw [->, ultra thick] (1,0) -- (3,2);
\draw [->, line width=5pt] (2,0) -- (4,2);
\end{tikzpicture}
```



Esistono altri stili predefiniti (qui omessi per brevità) e numerosi altri ne offre la libreria `arrows.meta`, che permette di personalizzare l'aspetto delle punte delle frecce regolandone finemente dimensioni, proporzioni, forma e colore.

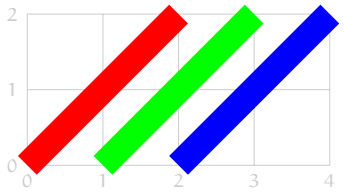
4.5 Colori

TikZ carica il pacchetto `xcolor` *senza opzioni*, perciò ne mette a disposizione solo i comandi e i colori di base mostrati nella tabella 5 nella pagina successiva. Per colorare un percorso basta scrivere il nome del colore scelto tra le sue opzioni. Il prossimo esempio mostra tre spesse linee colorate con colori puri:

Tabella 5: Colori predefiniti da TikZ.

black	darkgray	lime	pink	violet
blue	gray	magenta	purple	white
brown	green	olive	red	yellow
cyan	lightgray	orange	teal	

```
\begin{tikzpicture} [line width=10pt]
\draw [help lines] (0,0) grid (4,2);
\draw [red]   (0,0) -- (2,2);
\draw [green] (1,0) -- (3,2);
\draw [blue]  (2,0) -- (4,2);
\end{tikzpicture}
```



Si possono mescolare due colori per ottenerne altri in questo modo:

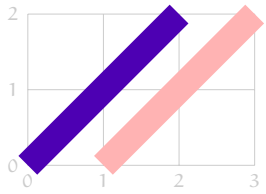
⟨*colore*$_1$⟩!⟨*valore*⟩!⟨*colore*$_2$⟩

dove ⟨*valore*⟩ è la percentuale di ⟨*colore*$_1$⟩ da mescolare con la rimanente percentuale di ⟨*colore*$_2$⟩ per raggiungere il 100 %, oppure così:

⟨*colore*⟩!⟨*valore*⟩

dove ⟨*valore*⟩ è la percentuale di ⟨*colore*⟩ da mescolare con la rimanente percentuale di bianco per raggiungere il 100 %. Il prossimo esempio mostra all'opera quanto si è appena descritto:

```
\begin{tikzpicture} [line width=10pt]
\draw [help lines] (0,0) grid (3,2);
\draw [red!30!blue] (0,0) -- (2,2);
\draw [red!30]      (1,0) -- (3,2);
\end{tikzpicture}
```



dove:

- `red!30!blue` mescola il 30 % di rosso con il 70 % di blu;
- `red!30` mescola il 30 % di rosso con il 70 % di bianco.

Si noti che l'eventuale colore dichiarato tra le opzioni di un percorso viene applicato su *tutte* le azioni eseguite sul percorso stesso, se più d'una (per esempio, la tracciatura del contorno, il riempimento con un colore, l'aggiunta di un testo). Si può assegnare il colore *solo a un'azione* dichiarandola nella forma ⟨*chiave*⟩=⟨*valore*⟩ ed esprimendo il colore come ⟨*valore*⟩. Per esempio, per colorare di rosso solo il tratto in un percorso tracciato e riempito, l'opzione è `draw=red` (si veda anche il paragrafo 5.1 a pagina 36).

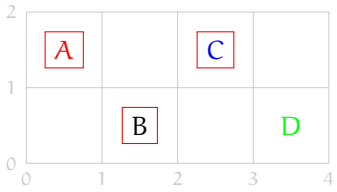
Analogamente accade con i nodi: l'eventuale colore dichiarato tra le opzioni colora *tutti* gli elementi del nodo (testo, bordo e riempimento).

Per colorare solo il testo, per esempio, l'opzione è

`text=`⟨*colore*⟩

Il colore dell'etichetta di un nodo coordinata, invece, va specificato tra le opzioni dell'etichetta stessa:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\node [draw, red] at (0.5,1.5) {$A$};
\node [draw=red] at (1.5,0.5) {$B$};
\node [draw, red, text=blue] at (2.5,1.5) {$C$};
\coordinate [label={[green]center:$D$}]
            (C) at (3.5,0.5);
\end{tikzpicture}
```



Per estendere il supporto ai comandi e ai colori di base, si carichi il pacchetto xcolor con le opzioni opportune *prima* del pacchetto Ti*k*Z. Quest'operazione permette, tra le altre cose:

- di utilizzare gli altri set di colori predefiniti di xcolor;
- di utilizzare un modello di colore differente da `rgb` e `gray` (gli unici modelli disponibili in Ti*k*Z) forzando la conversione dei colori mediante l'opzione `rgb` (si veda la documentazione di xcolor).

4.6 Definire uno stile personale

Uno *stile* è un insieme di opzioni grafiche identificato da un nome. Ti*k*Z permette di ridefinire gli stili esistenti e di definirne di nuovi mediante la sintassi

⟨*nome*⟩`/.style={`⟨*opzioni*⟩`}`

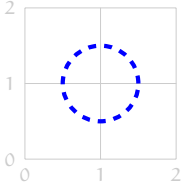
dove

- ⟨*nome*⟩ si spiega da sé;
- le parentesi graffe sono necessarie se si dichiarano più opzioni separate da virgole, oppure quando si dichiarano delle opzioni nella forma ⟨*chiave*⟩=⟨*valore*⟩;
- si faccia attenzione a scrivere correttamente l'operatore `/.` (con il punto finale) che precede `style`.

Applicare lo stile ⟨*nome*⟩ equivale ad applicare in una volta sola tutte le ⟨*opzioni*⟩ dichiarate nella sua definizione. Per esempio, l'opzione `help lines` data all'elemento `grid` è in realtà uno stile che imposta colore e spessore delle linee della griglia.

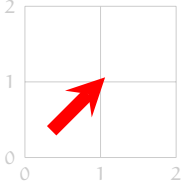
Se lo stile è definito come opzione di un ambiente, è disponibile *localmente*:

```
\begin{tikzpicture} [%
   linea/.style={ultra thick, dashed, blue}]
\draw [help lines] (0,0) grid (2,2);
\draw [linea] (1,1) circle (0.5);
\end{tikzpicture}
```



In alternativa, si può definire uno stile a livello globale con il comando `\tikzset`, che si può dare nel preambolo, preferibilmente, oppure nel corpo del documento: si noti che in quest'ultimo caso lo stile sarà disponibile solo per i disegni *successivi* al punto in cui il comando compare:

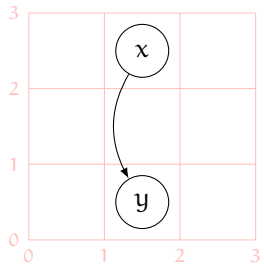
```
\tikzset{freccia/.style={%
   -stealth, red, line width=5pt}}
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,2);
\draw [freccia] (45:0.5) -- (45:1.5);
\end{tikzpicture}
```



Si badi che uno stile definito mediante il comando `\tikzset` dato in un ambiente è disponibile soltanto al suo interno.

Se infine si volesse modificare uno stile esistente aggiungendogli opzioni, basta usare `append style` al posto di `style`. Nell'esempio seguente lo si è fatto per ottenere la griglia colorata di rosa:

```
\tikzset{every node/.style={%
           draw, circle, minimum size=7mm},
         help lines/.append style=pink}
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,3);
\node (x) at (1.5,2.5) {$x$};
\node (y) at (1.5,0.5) {$y$};
\draw [-latex, bend right] (x) to (y);
\end{tikzpicture}
```



Nell'esempio appena mostrato si è personalizzato lo stile predefinito `every node` (inizialmente vuoto), molto utile per applicare in una sola volta tutte le opzioni in esso dichiarate a *tutti* i nodi del disegno.

5 RIEMPIMENTI E TRASPARENZE

Ti*k*Z permette di riempire i percorsi con un colore uniforme, una sfumatura o un motivo e di applicare trasparenze uniformi o graduali. Si possono usare sfumature e motivi predefiniti oppure crearne di personali con le indicazioni contenute nella documentazione del pacchetto.

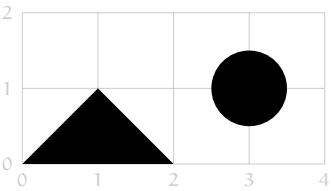
5.1 Campiture

Per riempire un percorso con un colore uniforme si usa il comando `\fill`, scorciatoia di `\path [fill]`, dal quale eredita la sintassi:

```
\fill [⟨opzioni⟩] ⟨istruzioni⟩;
```

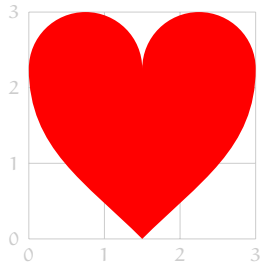
Le ⟨*istruzioni*⟩ permesse sono le stesse accettate dal comando `\draw`, ma questa volta le figure sono riempite senza tracciarne i bordi. Tutte le linee aperte vengono prima chiuse collegandone i due estremi con un segmento e poi riempite. Per le figure intrecciate, un meccanismo interno stabilisce quale parte debba essere riempita. Ecco il comando all'opera:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\fill (0,0) -- (1,1) -- (2,0)
      (3,1) circle (0.5);
\end{tikzpicture}
```



Il colore predefinito è il nero, ma può essere modificato come indicato nel paragrafo 4.5 a pagina 33. Ecco un esempio:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,3);
\fill [red] (1.5,0)
      to [out=45,  in=270]  (3,2.25)
      to [out=90,  in=0]    (2.25,3)
      to [out=180, in=90] (1.5,2.25)
      to [out=90,  in=0]    (0.75,3)
      to [out=180, in=90]  (0,2.25)
      to [out=270, in=135] (1.5,0);
\end{tikzpicture}
```



Si può dichiarare un riempimento anche dentro un percorso tracciato con l'opzione

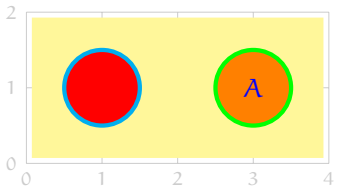
```
fill=⟨colore⟩
```

che applica l'eventuale colore specificato solo al momento di riempire il percorso. Analogamente, si può dichiarare una tracciatura anche dentro un percorso riempito con l'opzione

```
draw=⟨colore⟩
```

che applica l'eventuale colore dichiarato solo al tratto. In entrambi i casi, se il ⟨*colore*⟩ è omesso, il percorso verrà riempito (o tracciato) con lo stesso colore applicato dal comando `\draw` (o `\fill`). Ecco all'opera quanto si è appena descritto:

```
\begin{tikzpicture} [ultra thick]
\draw [help lines] (0,0) grid (4,2);
\fill [yellow!50, draw]
      (0.1,0.1) rectangle (3.9,1.9);
\fill [red, draw=cyan]     (1,1) circle (0.5);
\draw [green, fill=orange] (3,1) circle (0.5)
      node [text=blue] {$A$};
\end{tikzpicture}
```



Si noti che la tracciatura viene eseguita sempre *dopo* il riempimento. Quando si sovrappongono più percorsi opachi colorati, come nell'esempio precedente, il colore delle aree comuni sarà quello del percorso disegnato per ultimo. Talvolta, però, si potrebbe volere al suo posto il colore risultante dalla fusione dei colori di partenza. La documentazione del pacchetto spiega come ottenere questo effetto mediante i *metodi di fusione*, applicabili anche alle sfumature.

5.2 Sfumature

Per riempire un percorso con una sfumatura di colore si usa il comando `\shade`, scorciatoia di `\path [shade]`, dal quale eredita la sintassi:

```
\shade [⟨opzioni⟩] ⟨istruzioni⟩;
```

Il comando si comporta come `\fill`, ma questa volta il riempimento consiste in una sfumatura di colore definita attraverso le ⟨*opzioni*⟩, le più importanti delle quali si descrivono qui di seguito.

Le opzioni

```
top color=⟨colore1⟩, bottom color=⟨colore2⟩
```

definiscono una sfumatura *verticale* tra il ⟨$colore_1$⟩ (in alto) e il ⟨$colore_2$⟩ (in basso).

Le opzioni

```
left color=⟨colore1⟩, right color=⟨colore2⟩
```

definiscono una sfumatura *orizzontale* tra il ⟨$colore_1$⟩ (a sinistra) e il ⟨$colore_2$⟩ (a destra). In entrambi i casi è disponibile anche l'opzione `middle color=`⟨$colore_3$⟩ per impostare il colore intermedio.

Le opzioni

```
inner color=⟨colore1⟩, outer color=⟨colore2⟩
```

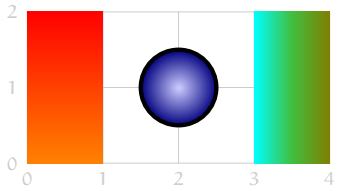
definiscono una sfumatura *radiale* tra il ⟨$colore_1$⟩ (all'interno) e il ⟨$colore_2$⟩ (all'esterno).

Si noti che `\shade` disegna solo la sfumatura. Se si volesse tracciare anche il bordo, basta dare al suo posto `\draw` con almeno una delle opzioni appena viste. Ecco all'opera quanto si è appena descritto:

Tabella 6: Alcuni motivi di riempimento forniti dalla libreria patterns.

horizontal lines	grid	fivepointed stars
vertical lines	crosshatch	sixpointed stars
north east lines	dots	bricks
north west lines	crosshatch dots	checkerboard
horizontal lines light gray	crosshatch dots gray	checkerboard light gray

```
\begin{tikzpicture} [ultra thick]
\draw [help lines] (0,0) grid (4,2);
\shade [top color=red, bottom color=orange]
       (0,0) rectangle (1,2);
\draw [inner color=blue!20, outer color=%
       blue!50!black] (2,1) circle (0.5);
\shade [left color=cyan, right color=olive]
       (3,0) rectangle (4,2);
\end{tikzpicture}
```



5.3 Motivi di riempimento

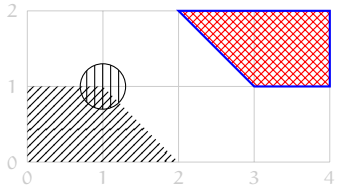
La libreria patterns di Ti*k*Z mette a disposizione numerosi motivi di riempimento predefiniti. Per riempire un percorso con un motivo si usa il comando `\pattern`, scorciatoia di `\path [pattern]`, dal quale eredita la sintassi:

```
\pattern [pattern=⟨motivo⟩, pattern color=⟨colore⟩] ⟨istruzioni⟩;
```

Il comando si comporta come `\fill`, ma questa volta il riempimento è il ⟨*motivo*⟩ indicato da scegliere tra quelli disponibili, alcuni dei quali sono raccolti nella tabella 6.

Anche in questo caso, `\pattern` disegna solo il motivo. Se si volesse tracciarne anche il bordo basta dare al suo posto `\draw` con le stesse opzioni. Eccone un esempio:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\pattern [pattern=north east lines]
         (0,0) -- (0,1) -- (1,1) -- (2,0);
\draw [pattern=vertical lines] (1,1) circle (3mm);
\draw [draw=blue, thick, pattern=crosshatch,
       pattern color=red]
      (4,2) -- (2,2) -- (3,1) -- (4,1) -- cycle;
\end{tikzpicture}
```



5.4 Trasparenze

L'opzione

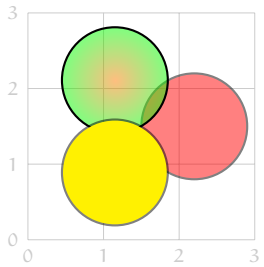
```
opacity=⟨valore⟩
```

permette di regolare il grado di opacità di un percorso mediante un ⟨*valore*⟩ compreso tra 0 (completamente trasparente) e 1 (totalmente opaco). Le opzioni analoghe

```
draw opacity=⟨valore⟩, fill opacity=⟨valore⟩, text opacity=⟨valore⟩
```

regolano il livello di opacità dei tracciati, dei riempimenti e del testo rispettivamente (si noti che i motivi al tratto sono considerati tracciati). Eccole all'opera:

```
\begin{tikzpicture} [thick]
\draw [help lines] (0,0) grid (3,3);
\draw [fill=red, opacity=0.5]
      (2.2,1.5) circle (0.7);
\draw [inner color=orange, outer color=green,
       fill opacity=0.5] (1.15,2.11) circle (0.7);
\draw [fill=yellow, draw opacity=0.5]
      (1.15,0.89) circle (0.7);
\end{tikzpicture}
```



La libreria fadings permette di definire una trasparenza graduale appoggiandosi alla sintassi delle sfumature con il comando

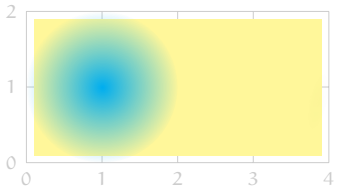
```
\tikzfading [name=⟨nome⟩, ⟨opzioni⟩]
```

grazie al quale si può richiamare una trasparenza in un percorso citandone semplicemente il ⟨*nome*⟩, scelto dall'utente. Le opzioni principali sono le stesse già descritte per il comando `\shade`, ma si noti che la luminosità dei colori viene convertita in livello di opacità: bianco è completamente opaco e nero è completamente trasparente. Per evitare confusione, è disponibile il colore `transparent` che corrisponde al nero, perciò `transparent!60` significa trasparente al 60 %. Poi si può applicare la trasparenza appena definita direttamente al percorso riempito mediante l'opzione

```
path fading=⟨nome⟩
```

come mostra l'esempio seguente:

```
\tikzfading [name=sfuma, inner color=white,
             outer color=transparent]
\begin{tikzpicture} [thick]
\draw [help lines] (0,0) grid (4,2);
\fill [yellow!50] (0.1,0.1) rectangle (3.9,1.9);
\fill [cyan, path fading=sfuma] (1,1) circle (1);
\shade [top color=red, bottom color=green,
        path fading=sfuma] (3,1) circle (1);
\end{tikzpicture}
```



Si noti che di per sé la trasparenza non ha colore, perciò le opzioni di riempimento vanno specificate a parte come al solito.

L'argomento "sfumature e trasparenze" è piuttosto articolato e va ben oltre la semplice introduzione appena esposta, perciò si rimanda il lettore alla documentazione del pacchetto.

6 TRASFORMAZIONI

Le *trasformazioni* più comuni effettuabili su un percorso o su un intero ambiente sono tre: *scalatura*, *traslazione* e *rotazione*. Esse agiscono solo sulle coordinate dei punti del percorso e non sulle altre sue caratteristiche come lo spessore di linea o la distanza del tratteggio.

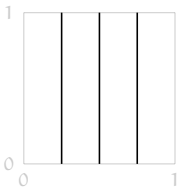
6.1 Scalatura

La *scalatura* è l'operazione che moltiplica le coordinate per un fattore di proporzionalità e si realizza con l'opzione

```
scale=⟨fattore⟩
```

Eccola all'opera:

```
\begin{tikzpicture} [scale=2, semithick]
\draw [help lines] (0,0) grid (1,1);
\draw (0.25,0) -- (0.25,1)
      (0.5cm,0cm) -- (0.5cm,1cm)
      (0.75,0) -- +(90:1cm);
\end{tikzpicture}
```



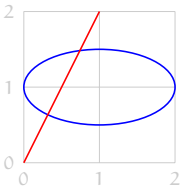
Si osservi che la scalatura agisce anche sulle coordinate espresse come misure di lunghezza.

In alternativa, si può scalare nella direzione di uno solo degli assi coordinati oppure applicare contemporaneamente due differenti fattori di scala con le varianti

```
xscale=⟨fattore⟩, yscale=⟨fattore⟩
```

il cui valore predefinito è 1. Eccone un esempio:

```
\begin{tikzpicture} [semithick]
\draw [help lines] (0,0) grid (2,2);
\draw [xscale=2, blue] (0.5,1) circle (0.5);
\draw [xscale=1, yscale=2, red]
      (0,0) -- (1,1);
\end{tikzpicture}
```



Si noti che la forma delle linee può risultare alterata quando si applica una scalatura dimetrica.

L'opzione `xscale=-1` (o `yscale=-1`) produce un ribaltamento rispetto all'asse delle ordinate (o asse delle ascisse).

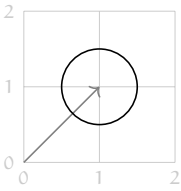
6.2 Traslazione

La *traslazione* è l'operazione che sposta l'origine del sistema di riferimento in un altro punto e si realizza con l'opzione

```
shift={(⟨punto⟩)}
```

il cui effetto è di sommare le coordinate del ⟨*punto*⟩ a tutte le coordinate del percorso, come si vede nell'esempio seguente:

```
\begin{tikzpicture} [semithick]
\draw [help lines] (0,0) grid (2,2);
\draw [->, gray]      (0,0) -- (1,1);
\draw [shift={(1,1)}] (0,0) circle (0.5);
\end{tikzpicture}
```

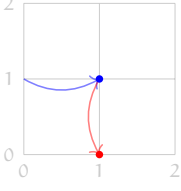


Le due varianti

```
xshift=⟨lunghezza⟩, yshift=⟨lunghezza⟩
```

permettono di traslare il sistema di riferimento singolarmente nelle due direzioni di un valore pari alla ⟨*lunghezza*⟩ assegnata:

```
\begin{tikzpicture} [semithick, bend right]
\draw [help lines] (0,0) grid (2,2);
\draw [->, blue, opacity=0.5] (0,1) to (1,1);
\draw [->, red, opacity=0.5]  (1,1) to (1,0);
\fill [xshift=1cm, blue]  (0,1) circle (1.4pt);
\fill [yshift=-1cm, red]  (1,1) circle (1.4pt);
\end{tikzpicture}
```



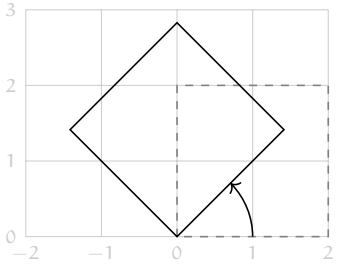
6.3 Rotazione

La *rotazione* è l'operazione che permette di ruotare il sistema di riferimento intorno all'origine e si realizza con l'opzione

```
rotate=⟨angolo⟩
```

dove l'⟨*angolo*⟩ è una coordinata angolare espressa in gradi sessadecimali:

```
\begin{tikzpicture} [semithick]
\draw [help lines] (-2,0) grid (2,3);
\draw [dashed, gray] (0,0) rectangle (2,2);
\draw [rotate=45]    (0,0) rectangle (2,2);
\draw [->] (1,0) arc (0:45:1);
\end{tikzpicture}
```

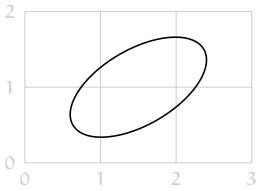


L'opzione

```
rotate around={⟨angolo⟩:(⟨punto⟩)}
```

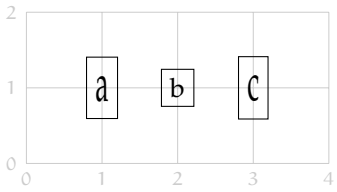
invece, permette di ruotare il sistema di riferimento di un certo ⟨*angolo*⟩ intorno al ⟨*punto*⟩ specificato:

```
\begin{tikzpicture} [semithick]
\draw [help lines] (0,0) grid (3,2);
\draw [rotate around={30:(1.5,1)}]
      (1.5,1) ellipse (1 and 0.5);
\end{tikzpicture}
```



Si noti che le trasformazioni appena descritte vengono applicate solo agli *elementi* di un nodo, se dichiarate tra le sue opzioni. In tutti gli altri casi agiscono solo sulle *coordinate* della posizione corrente del nodo, a meno di non specificare anche l'opzione `transform shape`, che estende le trasformazioni anche agli elementi del nodo, come mostra l'esempio seguente:

```
\begin{tikzpicture} [every node/.style=draw]
\draw [help lines] (0,0) grid (4,2);
\node [yscale=2] at (1,1) {a};
\draw [yscale=2] (2,0.5) node {b};
\draw [yscale=2,transform shape] (3,0.5) node {c};
\end{tikzpicture}
```



Si osservi che la scalatura è applicata solo agli elementi del nodo `a` e non alle coordinate del suo ⟨*punto*⟩.

L'opzione `sloped` può produrre un risultato inatteso se allo stesso nodo si applica *anche* una rotazione, pertanto si sconsiglia di usare contemporaneamente le due funzionalità.

7 APPROFONDIMENTI

Le poche nozioni introdotte in queste pagine non pretendono certo di esaurire le funzionalità di TikZ, che oltre alle possibilità offerte dalle varie librerie di cui è corredato mette a disposizione numerosissimi altri comandi e istruzioni, qui omessi per dovere di brevità, per soddisfare praticamente *qualunque* esigenza grafica. Questa sezione getta un rapido sguardo su poche caratteristiche avanzate, rimandando ancora una volta il lettore alla documentazione del pacchetto per eventuali approfondimenti.

7.1 Pic

Un *pic* è un "disegnino" che può essere aggiunto al disegno semplicemente richiamandolo tramite il suo nome, detto anche *tipo*. Lo si può immaginare come un trasferibile da sovrapporre alla tela e riutilizzabile più volte. Un pic si definisce con il comando `\tikzset` nel modo seguente:

```
\tikzset{⟨tipo⟩/.pic={⟨comandi⟩}}
```

dove:

- ⟨*tipo*⟩ è il nome assegnato al pic, che può contenere lettere, numeri e spazi ma *non* segni d'interpunzione;
- i ⟨*comandi*⟩ sono quelli necessari a generare il disegno del pic nei modi consueti.

Per esempio, il codice seguente genera un pic costituito da una faccina sorridente:

```
\tikzset{smile/.pic={
  \draw [fill=yellow] (0,0) circle (0.5); % volto
  \fill (-0.2,0.1) ellipse (1pt and 2pt)  % occhi
        (+0.2,0.1) ellipse (1pt and 2pt);
  \draw [line cap=round] (-0.2,-0.15) to [bend right=50] (0.2,-0.15); % bocca
  }
}
```

Per inserire un pic in un disegno si utilizza la stessa sintassi valida per i nodi, che nella sua forma più semplice è

```
\pic [⟨opzioni⟩] at (⟨punto⟩) {⟨tipo⟩};
```

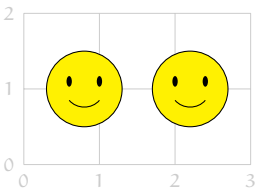
dove:

- `\pic` è l'abbreviazione di `\path [pic];`
- ⟨*punto*⟩ è il punto del disegno che si fa corrispondere all'origine del pic;
- ⟨*tipo*⟩ si spiega da sé.

Si noti l'assenza dell'etichetta di riferimento ⟨*nome*⟩: a differenza di quanto accade per i nodi, infatti, non ci si può riferire a un pic precedentemente inserito in un disegno.

L'esempio seguente mostra all'opera la sintassi appena esaminata:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\pic at (0.8,1) {smile};
\pic at (2.2,1) {smile};
\end{tikzpicture}
```

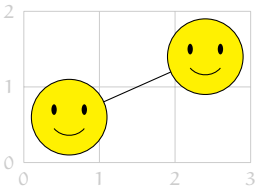


Si può aggiungere un pic direttamente nella posizione corrente di un percorso con l'istruzione

```
pic [⟨opzioni⟩] {⟨tipo⟩}
```

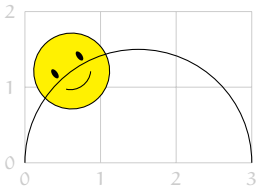
In tal caso, l'origine del pic coinciderà con la posizione corrente, come mostra l'esempio seguente:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (0.6,0.6) pic {smile} --
      (2.4,1.4) pic {smile};
\end{tikzpicture}
```



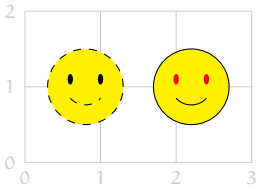
Per impostazione predefinita i pic vengono aggiunti *sopra* il percorso, ma è possibile fare il contrario con l'opzione `behind path`, mostrata all'opera nell'esempio seguente insieme alla chiave `pos` e all'opzione `sloped` già descritte per i nodi:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (0,0) arc (180:0:1.5)
      pic [behind path, pos=0.3, sloped] {smile};
\end{tikzpicture}
```



Le ⟨*opzioni*⟩ vengono assegnate per prime a tutti i ⟨*comandi*⟩ e quindi sono scavalcate dalle eventuali opzioni locali incompatibili. Si osservi l'esempio seguente:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\pic [dashed] at (0.8,1) {smile};
\draw (2.2,1) pic [fill=red] {smile};
\end{tikzpicture}
```

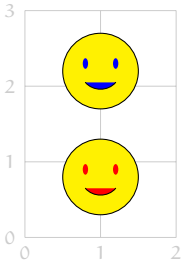


Nella faccina a sinistra, l'opzione `dashed` è passata a tutti i ⟨*comandi*⟩, ma gli occhi, ottenuti con `\fill`, rimangono ovviamente immuni dal suo effetto. In quella a destra, l'opzione `fill=red` è scavalcata dall'analoga `fill=yellow` presente nel comando per disegnare il volto e quindi viene ignorata, mentre è applicata al disegno degli occhi. Tuttavia, contrariamente a quanto ci si possa aspettare, viene ignorata anche nel disegno della bocca, perché tutte le azioni (tracciatura, riempimento, eccetera) dichiarate tra le ⟨*opzioni*⟩ vengono ignorate dai ⟨*comandi*⟩. Nel caso specifico, il riempimento rosso viene ignorato poiché per disegnare la bocca si è dato il comando `\draw`. L'opzione

```
pic actions
```

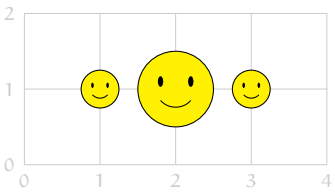
passata a uno dei ⟨*comandi*⟩ esegue su di esso anche le azioni dichiarate tra le ⟨*opzioni*⟩, purché compatibili, come mostra l'esempio seguente:

```
\tikzset{smile/.pic={
  \draw [fill=yellow] (0,0) circle (0.5);
  \fill (-0.2,0.1) ellipse (1pt and 2pt)
        (+0.2,0.1) ellipse (1pt and 2pt);
  \draw [pic actions, line cap=round] (-0.2,-0.15)
        to [bend right=50] (0.2,-0.15);}}
\begin{tikzpicture}
\draw [help lines] (0,0) grid (2,3);
\pic [fill=red] at (1,0.8) {smile};
\draw (1,2.2) pic [fill=blue] {smile};
\end{tikzpicture}
```



Analogamente a quanto accade ai nodi, una trasformazione dichiarata tra le opzioni del pic o del percorso viene applicata solo ai suoi elementi; in caso contrario, è applicata solo alle coordinate del ⟨*punto*⟩ (o posizione corrente), a meno di non specificare anche l'opzione `transform shape`, come nell'esempio seguente:

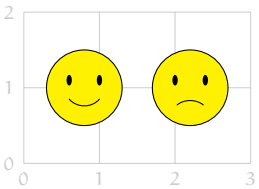
```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\pic  [scale=0.5] at (1,1) {smile};
\path [scale=0.5] (4,2) pic {smile};
\path [scale=0.5, transform shape]
      (6,2) pic {smile};
\end{tikzpicture}
```



Una delle potenzialità maggiori dei pic è quella di essere personalizzabili mediante parametri da passare tra le ⟨*opzioni*⟩ nella forma ⟨*chiave*⟩=⟨*valore*⟩. Per esempio, si può variare il

livello del liquido in una cisterna semplicemente impostando tra le ⟨*opzioni*⟩ la percentuale di riempimento. Per maggiori dettagli sull'argomento, piuttosto avanzato, si rimanda il lettore alla documentazione del pacchetto. A titolo d'esempio, si mostra il funzionamento di un pic di questo tipo:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\pic at (0.8,1) {smile};
\pic [viso=triste] at (2.2,1) {smile};
\end{tikzpicture}
```



7.2 Ripetere le azioni

Non è raro aver bisogno di ripetere più volte la stessa azione variandone solo uno o pochi parametri, effettuando un *ciclo*, nel gergo di TikZ. A questo scopo si usa il comando

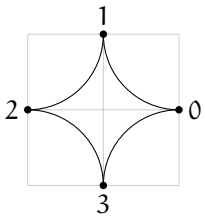
```
\foreach ⟨parametro⟩ [⟨opzioni⟩] in {⟨elenco⟩} {⟨comandi⟩}
```

dove:

- `\foreach` è il comando che ripete le azioni descritte dai ⟨*comandi*⟩;
- ⟨*parametro*⟩, in pratica un comando in stile LaTeX (`\i`, per esempio), è il parametro che varia tra i valori dichiarati nell'⟨*elenco*⟩;
- le ⟨*opzioni*⟩ si spiegano da sé;
- ⟨*elenco*⟩ è un insieme di valori separati dalla virgola e racchiusi tra parentesi graffe o, in alternativa, il nome di una macro che contiene tali valori;
- i ⟨*comandi*⟩, da racchiudere tra parentesi graffe se più d'uno, vanno espressi in funzione del ⟨*parametro*⟩ dichiarato in modo che a ogni passo tale ⟨*parametro*⟩ sia sostituito da un valore preso ordinatamente nell'⟨*elenco*⟩.

In altre parole, `\foreach` ripete le azioni descritte dai ⟨*comandi*⟩ variando il ⟨*parametro*⟩ tra i valori dichiarati nell'⟨*elenco*⟩. Eccolo all'opera:

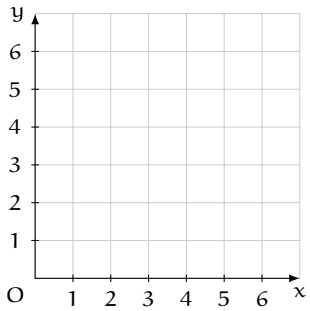
```
\begin{tikzpicture}
\draw [help lines] (-1,-1) grid (1,1);
\foreach \i in {0,1,2,3} {%
   \draw (90*\i:1) coordinate [label=90*\i:$\i$]
         arc (270+90*\i:180+90*\i:1);
   \fill (90*\i:1) circle (1.4pt);}
\end{tikzpicture}
```



Si osservi che il ⟨*parametro*⟩ può comparire anche nelle opzioni delle istruzioni e nel testo dei nodi.

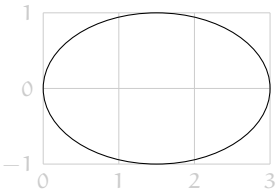
Si noti che per comprimere il contenuto dell'⟨*elenco*⟩ si possono usare i puntini di sospensione: per esempio, `{0,...,3}` equivale a `{0,1,2,3}` e `{9,7,...,2}` a `{9,7,5,3}`. La documentazione del pacchetto descrive tutte le altre alternative disponibili. Nell'esempio seguente, un sistema di assi coordinati, si vede all'opera quanto si è appena descritto:

```
\begin{tikzpicture} [%
   scale=0.5, >=latex, font=\footnotesize]
\draw [help lines] (0,0) grid (7,7);
\draw [<->] (0,7) node [left] {$y$} -- (0,0)
            -- (7,0) node [below] {$x$};
\foreach \i in {1,...,6}
   \draw (\i,1mm) -- (\i,-1mm) node [below] {$\i$}
         (1mm,\i) -- (-1mm,\i) node [left] {$\i$};
\node [below left] at (0,0) {$O$};
\end{tikzpicture}
```



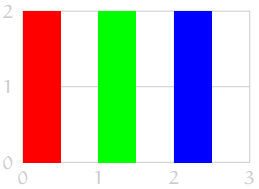
Il prossimo esempio mostra come costruire per conto proprio una griglia numerata:

```
\begin{tikzpicture}
\foreach \x in {0,1,...,3}
   \draw [help lines] (\x,-1) node [below,%
          font=\footnotesize] {$\x$} -- (\x,1);
\foreach \y in {-1,0,1}
   \draw [help lines] (0,\y) node [left,%
          font=\footnotesize] {$\y$} -- (3,\y);
\draw (1.5,0) ellipse (1.5 and 1);
\end{tikzpicture}
```



Si possono anche far variare più parametri contemporaneamente, separandone i rispettivi comandi nel ⟨*parametro*⟩ mediante l'operatore `/` (senza punto finale). La stessa sintassi va usata per gli elementi dell'⟨*elenco*⟩:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\foreach \P/\colore in {{(0,0)}/red,%
   {(1,0)}/green, {(2,0)}/blue}
   \fill [\colore] \P rectangle +(0.5,2);
\end{tikzpicture}
```



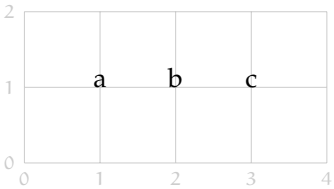
In pratica, a ogni passo i due comandi vengono sostituiti da una coppia di valori presi ordinatamente dall'⟨*elenco*⟩. Si noti che gli elementi dell'⟨*elenco*⟩ possono essere anche coordinate racchiuse tra parentesi graffe.

La documentazione di TikZ descrive tutte le ⟨*opzioni*⟩ del comando. Qui basta segnalare

```
count=⟨macro⟩ from ⟨valore⟩
```

che fornisce, tramite la ⟨*macro*⟩, un contatore incrementato automaticamente di un'unità ad ogni passo a partire dal ⟨*valore*⟩ facoltativo (pari a 1 per impostazione predefinita):

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,2);
\foreach \x [count=\i] in {a,...,c}
   \node [anchor=base] at (\i,1) {\x};
\end{tikzpicture}
```



Si possono annidare uno o più cicli anche dentro un'istruzione `node` o `pic`, come spiega la documentazione del pacchetto.

7.3 Annotare le immagini

In genere un nodo contiene del testo, ma può contenere anche dell'altro. Con il comando `\includegraphics`, per esempio, al disegno si possono aggiungere immagini trattandole come semplici nodi rettangolari, oppure annotarle con scritte e linee, come si mostra nell'esempio seguente per il file `orologio.jpg`:

```
\tikzset{immagine/.style={%
    above right, inner sep=0pt, outer sep=0pt},
        testo/.style={fill=white, align=center,
    fill opacity=0.6, text opacity=1, below,
    font=\sffamily\bfseries\footnotesize}}
\begin{tikzpicture} [>=latex, red, ultra thick]
\node [immagine] at (0,0)
  {\includegraphics[width=4cm]{orologio}};
\draw [->] (2,1) node [testo]
      {Posizione del sole\\
       rispetto all'eclittica} -- (1.57,2);
\end{tikzpicture}
```

L'immagine è stata inserita in modo tale che il suo vertice inferiore sinistro coincida con l'origine (stile `immagine`). Si noti che, avendo specificato una trasparenza per il riempimento dell'etichetta, è stato necessario dichiarare anche il livello di opacità del testo, altrimenti uguale a quello dell'etichetta.

Per ricavare le coordinate dei particolari dell'immagine, durante la realizzazione del disegno può essere molto utile sovrapporle una griglia con passo fine (qui omessa per maggiore chiarezza).

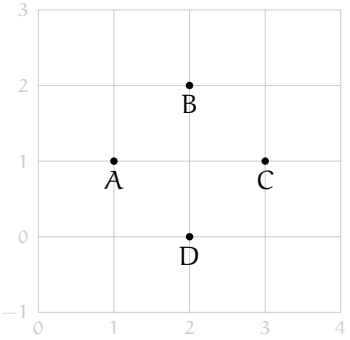
7.4 Calcolare le coordinate

La libreria calc permette di dichiarare le coordinate di un punto mediante espressioni matematiche nelle quali compaiono a propria volta delle coordinate. La sintassi generale delle coordinate calcolate è

```
($⟨espressione⟩$)
```

dove i dollari delimitano l'⟨*espressione*⟩ da valutare, il cui contenuto è una successione di addendi, ciascuno dei quali è costituito da una coordinata moltiplicata per un fattore numerico che a propria volta è il risultato di un'espressione matematica del tipo illustrato nel paragrafo 2.3 a pagina 7, eventualmente delimitato da parentesi graffe per maggiore chiarezza:

```
\begin{tikzpicture}
\draw [help lines] (0,-1) grid (4,3);
\coordinate [label=below:$A$] (A) at (1,1);
% Punti determinati a partire da A
\coordinate [label=below:$B$]
            (B) at (${cos(60)+1.5}*(A)$);
\coordinate [label=below:$C$]
            (C) at ($(A)+2*(0:1)$);
\coordinate [label=below:$D$]
            (D) at ($(C)-(A)$);
\fill (A) circle (1.4pt) (B) circle (1.4pt)
      (C) circle (1.4pt) (D) circle (1.4pt);
\end{tikzpicture}
```



La sintassi

```
($⟨punto1⟩!⟨valore⟩!⟨angolo⟩:⟨punto2⟩$)
```

nella quale ⟨*angolo*⟩: (compresi i due punti) è facoltativo, svolge due funzioni differenti a seconda del formato del ⟨*valore*⟩. In particolare:

- se ⟨*valore*⟩ è un numero, individua un punto che dista dal ⟨$punto_1$⟩ una lunghezza pari a ⟨*valore*⟩ volte la distanza che intercorre tra il ⟨$punto_2$⟩ e il ⟨$punto_1$⟩;
- se ⟨*valore*⟩ è una lunghezza, individua un punto che dista ⟨*valore*⟩ dal ⟨$punto_1$⟩.

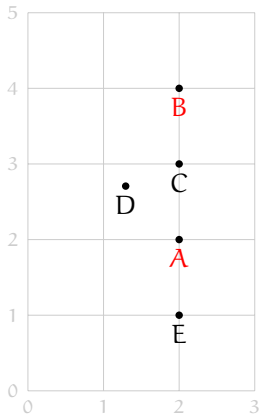
L'⟨*angolo*⟩ è la coordinata angolare del punto individuato rispetto all'asse polare diretto dal ⟨*punto*$_1$⟩ al ⟨*punto*$_2$⟩. Un angolo di 0° (valore predefinito) individua il punto sulla retta che congiunge il ⟨*punto*$_1$⟩ al ⟨*punto*$_2$⟩, come mostra il primo esempio in questa pagina.

Infine, la sintassi

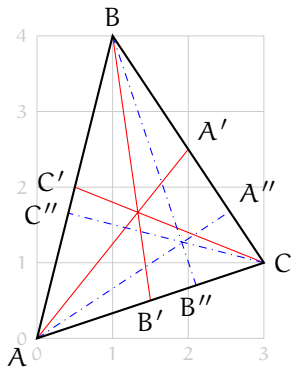
```
($⟨punto1⟩!⟨punto2⟩!⟨punto3⟩$)
```

proietta ortogonalmente il ⟨*punto*$_2$⟩ sulla retta su cui giacciono gli altri due punti. Il secondo esempio in questa pagina mostra come disegnare il baricentro e l'ortocentro di un un triangolo come intersezione delle sue mediane e delle sue altezze rispettivamente.

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,5);
\fill (2,2) circle (1.4pt)
      coordinate [label={[red]below:$A$}] (A)
      (2,4) circle (1.4pt)
      coordinate [label={[red]below:$B$}] (B)
      ($(A)!1/2!(B)$) circle (1.4pt)
      coordinate [label=below:$C$] (C)
      ($(A)!1/2!45:(B)$) circle (1.4pt)
      coordinate [label=below:$D$] (D)
      ($(A)!-1cm!(C)$) circle (1.4pt)
      coordinate [label=below:$E$] (E);
\end{tikzpicture}
```




```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,4);
% Vertici del triangolo
\coordinate ["225:$A$"]    (A) at (0,0);
\coordinate ["$B$"]        (B) at (1,4);
\coordinate ["0:$C$"]      (C) at (3,1);
% Punti medi
\coordinate ["45:$A'$"]   (Ap) at ($(B)!0.5!(C)$);
\coordinate ["-90:$B'$"]  (Bp) at ($(A)!0.5!(C)$);
\coordinate ["180:$C'$" yshift=0.5ex]
                          (Cp) at ($(A)!0.5!(B)$);
% Piedi delle altezze
\coordinate ["45:$A''$"]  (As) at ($(B)!(A)!(C)$);
\coordinate ["-90:$B''$"] (Bs) at ($(A)!(B)!(C)$);
\coordinate ["180:$C''$"] (Cs) at ($(A)!(C)!(B)$);
% Disegno del triangolo, mediane e altezze
\draw [red] (A) -- (Ap) (B) -- (Bp) (C) -- (Cp);
\draw [blue, dash dot]
      (A) -- (As) (B) -- (Bs) (C) -- (Cs);
\draw [thick] (A) -- (B) -- (C) -- cycle ;
\end{tikzpicture}
```



La libreria intersections permette di individuare i punti del disegno mediante l'intersezione di due percorsi. Si consulti la documentazione del pacchetto per gli eventuali approfondimenti.

7.5 Ridimensionare i disegni

Come si è detto nel paragrafo 2.2 a pagina 4, TikZ calcola automaticamente le dimensioni della tela per contenere tutto il disegno nel minor spazio possibile.

Quando si vuole creare un disegno di dimensioni stabilite, bisognerebbe determinare analiticamente le coordinate dei suoi elementi a monte, in modo che quelle dei punti più lontani soddisfino i requisiti richiesti già *durante* la realizzazione. Tuttavia, la cosa non è

sempre facile né immediata (si pensi alle difficoltà poste da una curva di Bézier), perciò in generale si preferisce scalare il disegno solo *dopo* averlo terminato.

Il modo più semplice per ridimensionare un intero disegno è dichiarare l'opzione opportuna tra le ⟨*opzioni globali*⟩ dell'ambiente `tikzpicture`. In tal caso va ricordato che alcune forme, tipicamente i nodi e i pic, non vengono influenzate dalle trasformazioni globali a meno di non specificare l'opzione `transform shape`, globale o locale a seconda che le si vogliano scalare tutte oppure solo alcune. La scelta dipende esclusivamente dal risultato prefisso per un particolare disegno e non può essere suggerita in via generale.

Una volta decisi gli elementi da ridimensionare, bisogna determinare il *fattore di scala*, inteso come rapporto tra le dimensioni desiderate e quelle originali del disegno. Nel caso in cui queste ultime non siano note, si possono determinare a mano mediante la già citata classe `standalone` e qualche semplice passaggio: si copi il codice del disegno nel corpo di un nuovo documento di prova; lo si componga; si apra il PDF risultante con un visualizzatore che permetta di leggerne le proprietà e si prenda nota delle dimensioni della pagina, che corrispondono alle dimensioni originali del disegno.

A questo punto, conoscendo le dimensioni finali del disegno si può calcolare il fattore di scala, che si consiglia di testare in via preliminare nel documento di prova. Se tutto va a buon fine, basta impostare lo stesso fattore nel documento originale. Però, quando nel disegno di partenza o in quello finale qualche nodo o pic da lasciare inalterato si trova sul bordo della tela, qualcosa potrebbe non funzionare come ci si aspetta. Non rimane che procedere per tentativi fino a raggiungere il risultato voluto.

A parte la scelta degli elementi da ridimensionare, che va fatta *sempre* a mano, si possono eseguire automaticamente tutte le altre operazioni tramite il pacchetto tikzscale. I passaggi da eseguire sono due: si memorizzi ciascun disegno in un file a sé, costituito *esclusivamente* dall'ambiente `tikzpicture` con le relative istruzioni; si includano i file appena creati nel documento con il comando `\includegraphics` come se fossero normali immagini.

Le opzioni alternative disponibili sono

```
width=⟨larghezza⟩, height=⟨altezza⟩
```

con le quali si può ridimensionare il disegno a una larghezza *oppure* a un'altezza assegnata rispettivamente (non contemporaneamente).

Si supponga di aver generato a parte il file `disegno.tikz` costituito dal seguente codice (si può usare un'estensione qualsiasi: qui si è scelto `.tikz` per chiarezza):

```
\begin{tikzpicture}
\draw (0,0) rectangle (4,2);
\end{tikzpicture}
```

che corrisponde chiaramente a un disegno di $4 \times 2\,\mathrm{cm}^2$. Se si vuole il disegno alto 3 cm, bisogna scrivere nel punto del documento in cui s'intende inserirlo:

```
\includegraphics[height=3cm]{disegno.tikz}
```

e si otterrà una versione $6 \times 3\,\mathrm{cm}^2$ del disegno originale, cioè con la larghezza scalata di conseguenza.

Si possono utilizzare anche espressioni che si riferiscono a registri di lunghezza come in

```
\includegraphics[width=0.7\textwidth]{disegno}
```

per impostare la larghezza del disegno al 70 % di quella della gabbia del testo. Si noti che si possono omettere le estensioni `.tikz`, `.TikZ`, `.TIKZ`, `.pgf` o `.PGF`, come si è fatto nell'esempio precedente.

Poiché nella ricerca della dimensione assegnata Ti*k*Z procede per approssimazioni successive, la composizione del documento potrebbe subire rallentamenti, tanto più in documenti con numerosi disegni da ridimensionare. In tal caso, la già citata libreria external velocizza le operazioni.

8 UNIVERSO TikZ

Con i comandi e le istruzioni messi a disposizione dal pacchetto TikZ è virtualmente possibile disegnare qualsiasi cosa; tuttavia, per esigenze particolari, sono disponibili librerie interne ed esterne e pacchetti dedicati a settori specifici (biologia, chimica, ingegneria, matematica e altre discipline) che permettono di realizzare disegni anche complessi con una sintassi semplificata. Questo paragrafo mostra una lista abbastanza completa (al momento della pubblicazione) di quelli disponibili su ☞ CTAN suddivisi per discipline, descrive le principali librerie interne e offre una galleria d'esempi (senza il codice sorgente, figure 10 a pagina 53 e 11 a pagina 54) realizzati con alcuni tra i più riusciti pacchetti basati su TikZ.

8.1 Librerie interne

automata Per disegnare macchine a stati finiti e macchine di Turing (figura 10a).

calendar Per realizzare calendari.

circuits Per disegnare circuiti logici ed elettronici abbastanza semplici.

datavisualization Per semplificare la realizzazione di grafici elaborati.

er Per disegnare diagrammi Entità-Relazioni.

graphs Per disegnare grafi.

lindenmayersystems Per disegnare frattali bidimensionali e profili di arbusti (figura 10b).

mindmap Per realizzare mappe mentali o concettuali (figura 10c).

petri Per disegnare reti di Petri.

spy Per ingrandire un particolare del disegno in una zona della figura (figura 10b).

tree Per disegnare strutture ad albero.

8.2 Uno sguardo su CTAN

Chimica e biologia

bohr Per disegnare semplici modelli atomici di Bohr fino al numero atomico 112. Con appositi comandi permette di scrivere il simbolo o il nome di un elemento dato il suo numero atomico, e viceversa.

chemfig Per disegnare la struttura di qualunque tipo di molecola chimica (figura 10d).

chemmacros Suite di tre pacchetti: chemmacros è un insieme di macro predefinite per scrivere la chimica; chemformula (alternativo a mhchem) permette di scrivere qualunque tipo di formula chimica; ghsystem permette di inserire e richiamare le *indicazioni di pericolo* e i *consigli di prudenza* del GHS (figura 10e).

modiagram Crea facilmente diagrammi dei livelli energetici degli orbitali molecolari (figura 10f).

pgfmolbio Suite di tre moduli per disegnare grafici di biologia molecolare: chromatogram disegna cromatogrammi di DNA sequenziato a partire da file `.scf`; domains disegna diagrammi del dominio di una proteina; convert rende pgfmolbio compatibile con i motori di TEX non Lua. Il pacchetto richiede LuaLATEX.

tikzorbital Per disegnare facilmente diagrammi di orbitali molecolari e di orbitali atomici di tipo *s*, *p* e *d*.

xymtex Per disegnare formule di struttura.

Matematica e fisica

braids Per disegnare diagrammi a treccia (diffusi in topologia).

duotenzor Per disegnare circuiti e diagrammi *duotensor*.

endiagram Per disegnare diagrammi della curva di energia potenziale.

fast diagram Per disegnare diagrammi FAST (diffusi in analisi funzionale).

hf-tikz Per evidenziare formule matematiche o loro parti. Funziona anche nelle presentazioni beamer (figura 11e).

luasseq Per disegnare diagrammi di sequenze spettrali. Il pacchetto richiede LuaLaTeX.

neuralnetwork Per realizzare diagrammi di reti neurali.

pgfplots Per disegnare grafici di funzione nel piano e nello spazio e altri tipi di diagrammi per la rappresentazione grafica dei dati (figura 10g).

randomwalk Per disegnare e personalizzare passeggiate aleatorie.

rulercompass Per il disegno geometrico con riga e compasso.

spath3 Per manipolare i "percorsi morbidi" in PGF, disegnare percorsi calligrafici e diagrammi di nodo.

tikz-bayesnet Libreria per disegnare reti bayesiane, modelli grafici e *factor graphs*.

tikz-cd Per disegnare diagrammi commutativi (figura 10h).

tikzpfeile Per uniformare le punte delle frecce standard a quelle disegnate da Ti*k*Z, altrimenti diverse.

tkz-base Per creare sistemi di riferimento cartesiani ortogonali.

tkz-berge Per disegnare alcuni classici grafi combinatori della teoria dei grafi.

tkz-euclide Per il disegno geometrico nel piano cartesiano (figura 11a).

tkz-fct Per creare grafici di funzione nel piano cartesiano.

tkz-graph Per creare grafi (figura 11b).

tkz-kiviat Per creare diagrammi di Kiviat.

tkz-linknodes Per evidenziare attraverso delle linee di collegamento le operazioni eseguite tra righe successive di ambienti matematici `align` o `aligned`.

tkz-tab Per creare tabelle di segni e variazioni.

tqft Fornisce forme di nodi per costruire diagrammi della teoria topologica dei campi quantizzati.

venndiagram Per disegnare diagrammi di Venn a 2 o 3 insiemi.

Informatica ed elettronica

bloques Per disegnare diagrammi a blocchi.

bodegraph Per disegnare diagrammi di Bode, Nichols e Nyquist (figura 11c). Richiede gnuplot.

circuitikz Per disegnare reti elettriche, logiche ed elettroniche (figura 11d).

drawstack Per disegnare pile di esecuzione, generalmente per illustrare i concetti del linguaggio *assembly*.

flowchart Fornisce forme per disegnare diagrammi di flusso.

pgf-umlsd Per disegnare diagrammi di sequenza secondo l'UML.

reotex Per disegnare circuiti *Reo*.

sa-tikz Per disegnare architetture di commutazione.

schemabloc Per disegnare schemi a blocchi (figura 11e).

tikz-opm Per disegnare diagrammi OPD.

tikz-timing Per disegnare diagrammi temporali.

tkz-orm Per disegnare diagrammi ORM.

Grafica e geometria della pagina

harveyballs Per disegnare "Harvey balls".

ltximg Script Perl per esportare automaticamente i disegni realizzati con Ti*k*Z in vari formati grafici.

makeshape Per definire forme personalizzate.

hobby Rende disponibile l'algoritmo di Hobby per tracciare curve di Bézier su una serie di punti.

mdframed Per evidenziare una porzione di testo con una cornice spezzabile su più pagine.

pgf-blur Per realizzare ombre sfocate da aggiungere agli elementi del percorso, nodi compresi (figure 10a, 10c e 11c).

pxpgfmark Per realizzare connessioni inter-immagine con il motore e-pTEX.

reflectgraphics Per disegnare immagini riflesse.

tcolorbox Per realizzare box di testo colorati e incorniciati, con un titolo e personalizzabili in ogni aspetto (figura 11f).

tikz-3dplot Per disegnare sistemi di riferimento nello spazio e semplici grafici tridimensionali.

tikzpagenodes Definisce speciali nodi corrispondenti alle diverse aree della pagina per facilitare la disposizione del materiale.

tikzposter Classe che facilita la creazione di poster usando Ti*k*Z.

tikzscale Per scalare ad una dimensione assegnata i disegni realizzati con Ti*k*Z.

tikzsymbols Definisce simboli vari (emoticon, simboli di cucina e altri).

vgrid Per emulare l'aspetto dei fogli a righe nella gabbia del testo.

Varie

bchart Per disegnare semplici grafici a barre orizzontali (figura 11g).

guitarchordschemes Per realizzare diagrammi degli accordi per chitarra.

chronosys Per disegnare semplici diagrammi temporali.

tikz-dependency Per disegnare grafi delle dipendenze.

forest Per disegnare alberi linguistici e d'altro tipo (figura 11h).

grafcet Per disegnare diagrammi funzionali sequenziali GRAFCET.

logicpuzzle Per disegnare gli schemi di numerosi giochi.

mycv Classe per comporre un curriculum vitae in diversi layout.

pas-cours Per preparare materiale didattico.

pas-crosswords Per realizzare schemi di parole crociate.

pas-cv Per semplificare la stesura di un curriculum vitae.

pas-tableur Per emulare l'aspetto di un foglio di calcolo.

pgf-soroban Per disegnare i soroban, un tipo di abaco giapponese.

pgfgantt Per disegnare diagrammi di Gantt (diffusi nel *project management*).

rubikcube Per disegnare configurazioni e istruzioni di rotazione di un cubo di Rubik.

setdeck Per disegnare le carte del gioco Set.

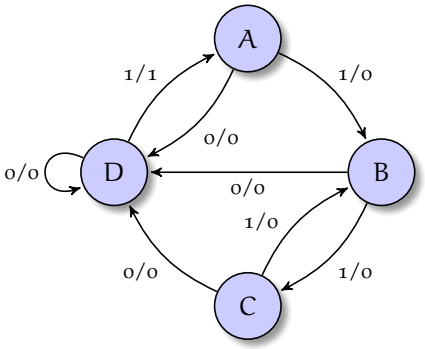
smartdiagram Per disegnare diagrammi a partire da una lista di elementi colorati automaticamente.

tikz-inet Per disegnare reti d'interazione.

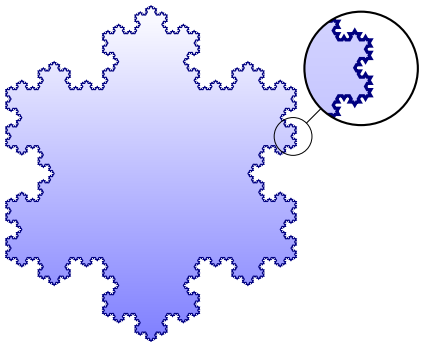
tikz-qtree Per disegnare strutture ad albero.

tikzinclude Per importare un solo disegno realizzato con Ti*k*Z da un file che ne contiene più d'uno.

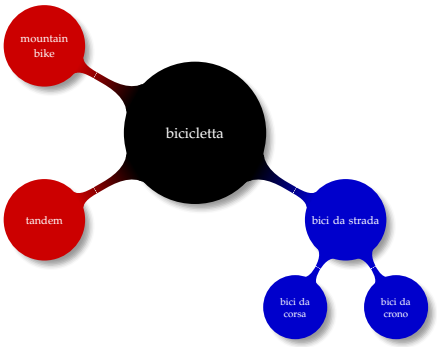
timing-diagrams Per disegnare e annotare vari tipi di diagrammi temporali.



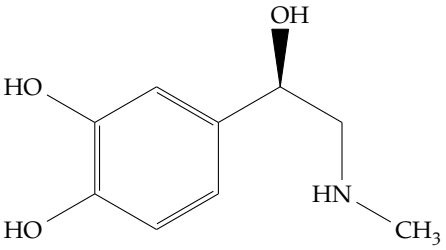
(a) automata + pgf-blur



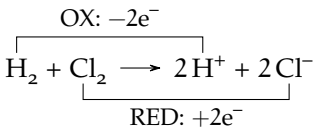
(b) lindenmayersystems + spy



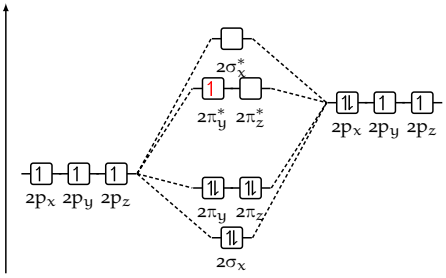
(c) mindmap + pgf-blur



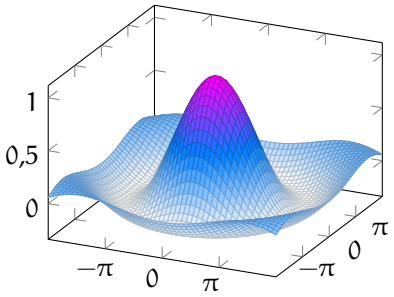
(d) chemfig



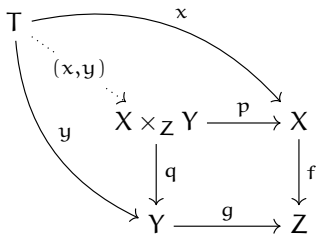
(e) chemmacros



(f) modiagram

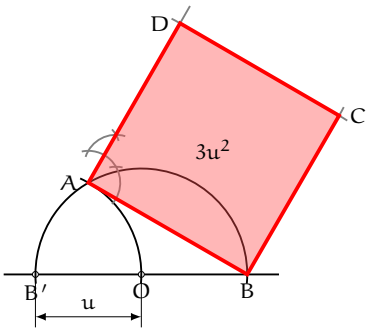


(g) pgfplots

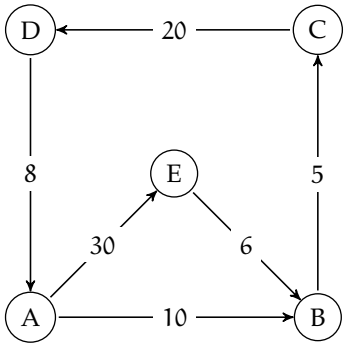


(h) tikz-cd

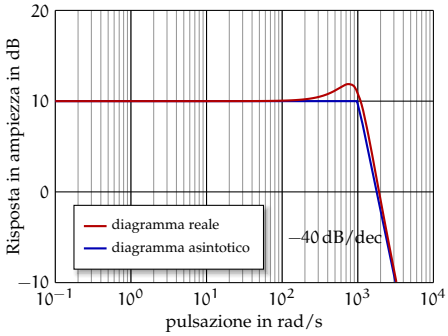
Figura 10: Galleria d'esempi/1.



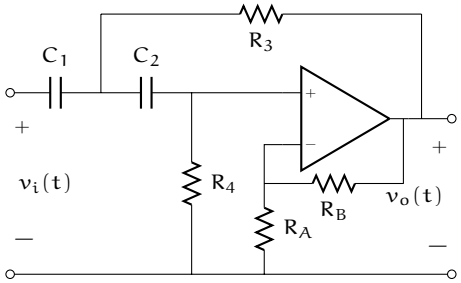
(a) tkz-euclide



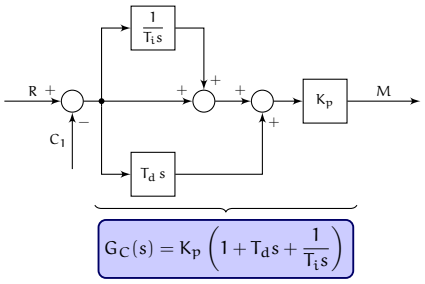
(b) tkz-graph



(c) bodegraph + pgf-blur



(d) circuitikz

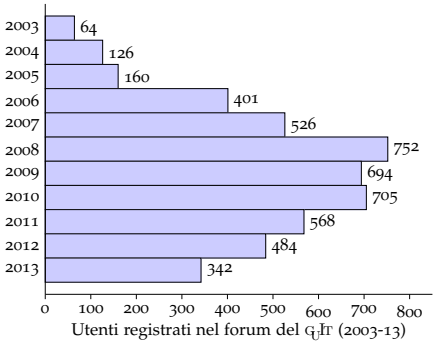


(e) schemabloc + hf-tikz

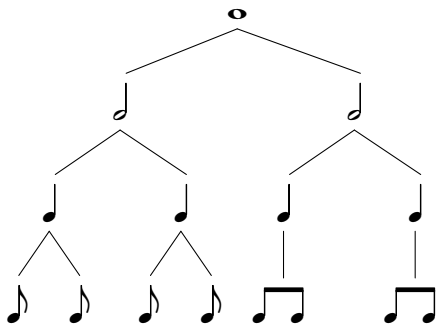
Formula di Erone

$$A = \sqrt{p(p-a)(p-b)(p-c)}$$

(f) tcolorbox



(g) bchart



(h) forest

Figura 11: Galleria d'esempi/2.

RIFERIMENTI BIBLIOGRAFICI

Pantieri, Lorenzo e Tommaso Gordini

2012a «L'arte di disegnare grafici con LaTeX», http://www.lorenzopantieri.net/LaTeX_files/Grafici.pdf.

2012b *L'arte di scrivere con LaTeX*, http://www.lorenzopantieri.net/LaTeX_files/ArteLaTeX.pdf.

Stacey, Andrew

2014 *The* Hobby *package*, http://ftp.uniroma2.it/TeX/graphics/pgf/contrib/hobby/hobby_doc.pdf.

Tantau, Till

2013 *TikZ and* PGF *Manual for Version 3.0.0*, http://ftp.uniroma2.it/TeX/graphics/pgf/base/doc/generic/pgf/pgfmanual.pdf.