

Produrre grafica vettoriale di alta qualità programmando *Asymptote*

Agostino De Marco

Sommario

Asymptote è un potente sistema grafico programmabile, distribuito con licenza GNU e particolarmente adatto a produrre disegni tecnici. Il suo linguaggio di programmazione è di alto livello ed è basato su sofisticate funzionalità matematiche. *Asymptote* permette di comporre con \LaTeX le annotazioni delle figure (semplici etichette ed oggetti testuali più complessi) garantendo una resa tipografica di alta qualità.

Questo articolo non offre una panoramica completa su *Asymptote* ma, piuttosto, si propone di introdurre gradualmente, con degli esempi mirati, gli elementi salienti del suo linguaggio di programmazione e di porre in risalto gli aspetti importanti per l'utilizzatore \LaTeX . Data la vastità degli argomenti legati alla programmazione di un linguaggio di alto livello, il lettore interessato agli approfondimenti sulle potenzialità di questo sistema grafico è rimandato ai riferimenti citati ed allo studio del codice sorgente delle numerose funzioni predefinite.

Abstract

Asymptote is a powerful programmable graphics system, distributed with a GNU license. It provides a high level descriptive programming language, which is based on advanced mathematical functions. *Asymptote* is particularly suited to produce technical drawings. It allows users to compose labels and more complicated textual objects with \LaTeX and this feature guarantees a high-quality typographical performance.

This article does not give a complete overview of *Asymptote*, rather it has the aim at introducing gradually, with appropriate examples, the main elements of its programming language emphasizing the aspects which are of interest for the \LaTeX user. Given the breadth of topics related to a high level programming language, the reader interested in the details and in the potential of this graphics system is advised to read carefully the references cited and to study the source code of the many pre-defined functions.

1 Introduzione

Gli utenti di \LaTeX hanno spesso bisogno d'inserire nei loro documenti disegni, schemi e figure di varia complessità. Sui precedenti numeri di \LaTeX sono

comparsi diversi articoli che trattano questo argomento. Si vedano: CASCHILI (2006) per un'introduzione all'uso del pacchetto `picture`, CASCHILI (2007) per un'interessante panoramica sul pacchetto `pstricks`, ed inoltre DE MARCO (2008) per una discussione sulla gestione avanzata delle figure in \LaTeX (con esempi d'uso dei pacchetti `psfrag`, `tikz` e `pgfplots`).

Un'ottima descrizione delle numerose possibilità offerte da \LaTeX tramite i suoi pacchetti di estensione per la produzione di materiale grafico è data nell'articolo di BECCARI (2007) apparso sulla rivista *PracTeX Journal*. Questo lavoro si concentra sulle tecniche adatte a creare disegni direttamente nel sorgente \LaTeX di origine (senza l'uso di programmi esterni o limitandosi, al più, a qualche conversione di formato).

Un'introduzione all'uso del programma *Sketch* è presentata in DE MARCO (2007). È questo un esempio in cui un sistema esterno, programmabile attraverso un linguaggio grafico descrittivo, produce in uscita dei frammenti di codice `pstricks` o `tikz`, da includere in un sorgente \LaTeX . Questo articolo è orientato, in particolare, alla produzione di illustrazioni di grafica tridimensionale.

Gli utenti \LaTeX hanno anche la possibilità di lavorare con strumenti di disegno esterni, comunemente detti ' \LaTeX -aware'. Essi sono cioè capaci di comporre le annotazioni testuali all'interno delle figure utilizzando una distribuzione pre-installata di \LaTeX . Questa caratteristica assicura la coerenza tipografica delle immagini create con questi strumenti con i documenti \LaTeX destinati ad includerle. Ne risulta un alto livello di qualità complessiva della composizione.

Un esempio molto interessante di strumento di disegno \LaTeX -aware è costituito dal programma *Ipe*, disponibile sia per Linux che per Windows¹. *Ipe* è un'applicazione dotata di interfaccia grafica (*graphical user interface*, GUI). Essa si presenta come un *editor* visuale di disegni ed è stata sviluppata per la creazione di immagini direttamente in formato PDF (*Portable Document Format*) o EPS (*Encapsulated PostScript*). L'utente può inserire nei disegni delle annotazioni testuali attraverso un'apposita GUI, nella quale viene richiesto di digitare il corrispondente codice \LaTeX . Ciascuna annotazione viene compilata in un processo separa-

1. <http://tclab.kaist.ac.kr/ipe/> (sito principale), <http://luaforge.net/projects/ipe/> (*repository*), <http://melusine.eu.org/lab/ipe> (galleria di esempi)

to dall'applicazione principale ed il risultato viene importato come oggetto grafico all'interno della finestra di lavoro. Ciò risulta praticamente trasparente all'utente, che ha l'impressione di lavorare con una comune applicazione grafica con il vantaggio di ottenere delle annotazioni create con L^AT_EX. *Ipe* permette di aggiungere delle funzionalità personalizzate attraverso il meccanismo dei *plug-in* detti *ipelet* (cioè delle librerie a collegamento dinamico che l'utente deve sviluppare in C++).

Va osservato che anche *Inkscape*, il popolare programma di grafica vettoriale², è espandibile con dei *plug-in* sviluppati in linguaggio Python. Tra questi va annoverata l'estensione *texttext*, che permette di elaborare oggetti testuali mediante pdfL^AT_EX e di renderli nel *viewport* dell'applicazione come tracciati³.

Nel seguito del presente articolo illustrerò le caratteristiche principali del programma di grafica vettoriale *Asymptote* (BOWMAN, 2009) disponibile per Linux, Windows e Mac. Questo strumento di disegno, essendo anch'esso L^AT_EX-aware, è concettualmente simile ad *Ipe*. A differenza di quest'ultimo, *Asymptote* è anche un interprete di comandi e dispone di un linguaggio di programmazione di alto livello orientato alla matematica. Ciò lo rende un potente strumento, adatto alla produzione di disegni tecnici e di immagini a carattere matematico.

Esistono molte informazioni su internet su *Asymptote*. Una breve introduzione all'uso è data in HAMMERLINDL e BOWMAN (2007). Il forum *Art of Problem Solving*⁴ ha una sezione dedicata a questo programma e ne fornisce un *tutorial* di livello base. Di notevole interesse sono i due articoli scritti da John Bowman, lo sviluppatore di *Asymptote* insieme a Andy Hammerlindl e Tom Prince, pubblicati sulla rivista *TUGboat*: il primo (BOWMAN e HAMMERLINDL, 2008) presenta le caratteristiche principali, il secondo (BOWMAN e SHARDT, 2009) discute gli algoritmi su cui si basano le funzionalità orientate alla rappresentazione di scene tridimensionali.

Inoltre, un'ampia galleria di esempi d'uso (corredati di sorgenti) è disponibile sul sito ufficiale di *Asymptote*⁵. Il sito contiene anche dei collegamenti a molte risorse esterne⁶, tra cui due ulteriori gallerie di esempi⁷ ed una eccellente guida passo passo scritta da Philippe Ivaldi, un utilizzatore esperto che si è distinto per la sua produttività⁸.

2. <http://www.inkscape.org>

3. <http://www.elisabetta.fi/ptvirtan/software/texttext/>

4. [http://www.artofproblemsolving.com/wiki/index.php/Asymptote_\(Vector_Graphics_Language\)](http://www.artofproblemsolving.com/wiki/index.php/Asymptote_(Vector_Graphics_Language))

5. <http://asymptote.sourceforge.net/gallery/>

6. <http://asymptote.sourceforge.net/links.html>

7. <http://piprim.tuxfamily.org/asymptote/index.html> e <http://marris.org/asymptote>

8. <http://piprim.tuxfamily.org/asymptote/generale/index.html>

2 Come funziona *Asymptote*?

Lo strumento di lavoro si presenta all'utilizzatore come un'applicazione *command-driven*. Questo significa che all'utente è richiesto di raccogliere la sequenza di comandi necessari a realizzare un dato disegno in un file sorgente (*script* di estensione *.asy*), da *compilare* con *Asymptote* per ottenere un'immagine vettoriale in formato PostScript o PDF. Questa modalità di funzionamento è chiamata *batch mode*. Detto *test.asy* un file sorgente, l'utente digiterà il comando `asy -V test` al *prompt* di una finestra dei comandi ottenendo il file *test.eps* (l'opzione `-V` sta per *view* e permette di aprire automaticamente il file di output con un'applicazione predefinita).

Di default *Asymptote* produce un output in formato PostScript, ma può anche generare un'uscita in qualsiasi formato che il pacchetto *ImageMagick*⁹ sia in grado di produrre (opzione `-outformat <formato>`). È possibile configurare opportunamente l'applicazione impostando per default il formato di uscita PDF e pdfL^AT_EX come compositore delle annotazioni testuali (opzione `-tex pdflatex`).

Per gli utenti Windows con un ambiente di lavoro opportunamente configurato è sufficiente fare doppio click su un file di estensione *.asy* per lanciare un processo di compilazione con *Asymptote* (tramite l'eseguibile *asy.exe*) ed ottenere automaticamente un'immagine nel formato desiderato.

L'utilizzo di *Asymptote* in modalità *batch* è consigliabile. Esso si contrappone alla modalità *interattiva* per la quale si rimanda alla consultazione del manuale d'uso (HAMMERLINDL *et al.*, 2009).

Nella distribuzione ufficiale di *Asymptote* è compresa una rudimentale applicazione, denominata *xasy*, dotata di interfaccia grafica ed utile ad apportare semplici modifiche a disegni precedentemente generati tramite uno *script*. Questo programma, attualmente in fase di sviluppo, è destinato a diventare una vera e propria applicazione interattiva, capace di generare direttamente dei nuovi oggetti grafici. Ciò renderà *Asymptote* accessibile all'utente medio, permettendo a questo strumento di disegno di raggiungere il livello di semplicità d'uso di *Ipe* (almeno per gli utenti interessati al metodo di lavoro esclusivamente visuale).

Una caratteristica ulteriore, di non trascurabile importanza, è che il codice *Asymptote* può essere incluso all'interno di un sorgente L^AT_EX mediante il pacchetto di estensione *asymptote* (presente nella distribuzione ufficiale) che mette a disposizione l'ambiente *asy*. Si veda più avanti in questo articolo per maggiori dettagli.

9. <http://www.imagemagick.org/>

3 Caratteristiche del linguaggio di programmazione *Asymptote*

Asymptote offre un potente linguaggio descrittivo di grafica vettoriale pensato per la creazione di disegni tecnici. Esso dispone di numerose funzioni matematiche che permettono di ottenere risultati di notevole qualità e precisione.

Gli autori del linguaggio dichiarano di averlo concepito allo scopo di creare uno strumento standard per la creazione di figure matematiche, così come LATEX rappresenta lo standard *de facto* per la creazione di documenti matematici. Attualmente *Asymptote* è considerato il moderno successore di METAPOST (BOWMAN e HAMMERLINDL, 2008; HOBBY, 2009).

Uno dei vantaggi principali di *Asymptote* rispetto ad altri pacchetti grafici è che esso mette a disposizione un vero e proprio linguaggio di programmazione di alto livello. Ciò è vero solo per poche altre applicazioni di grafica, quasi tutte commerciali.

Asymptote ha la caratteristica di essere un ambiente matematico di disegno, basato sull'uso naturale di sistemi di coordinate. Il suo linguaggio di programmazione è ispirato a METAPOST, ma con una sintassi molto più pulita e potente, quasi del tutto simile a quella del C++.

Asymptote è stato progettato soprattutto per la creazione di figure matematiche che hanno bisogno di composizione tipografica. Gli utenti hanno comunque la possibilità di scrivere *moduli di estensione* (raccolte di funzioni) che aggiungono nuove funzionalità all'ambiente grafico e lo adattano alle specifiche esigenze. Molte delle nuove caratteristiche di questo sistema grafico, che si vanno migliorando o aggiungendo ad ogni nuova *release*, sono realizzate nel suo stesso linguaggio, sotto forma di funzioni raccolte in moduli.

L'estensione del linguaggio tramite funzioni definite dall'utente rientra fra le tecniche di programmazione tradizionali, messe a disposizione da tutti i linguaggi procedurali. Più avanti vedremo con un semplice esempio come *Asymptote*, al pari del C++, supporta il paradigma di programmazione a oggetti (STROUSTRUP, 2000). Esso non solo permette al programmatore di definire strutture dati personalizzate che inglobano anche delle *funzioni membro* (tipi *concreti*), ma supporta anche il meccanismo di ereditarietà che consente di costruire gerarchie di strutture dati (funzioni membro *virtuali* e tipi *derivati*).

Nelle distribuzioni recenti di *Asymptote* sono già disponibili dei moduli di estensione sviluppati da alcuni utenti esperti, che l'utilizzatore generico può importare all'occorrenza nel suo codice sorgente. Va segnalato il modulo `geometry.asy` di Philippe Ivaldi¹⁰.

10. http://svnweb.tuxfamily.org/filedetails.php?repname=piprim%2Fgeometrydoc&path=%2Fbranches%2Fen%2Fgeometry_en.pdf

4 Disegnare programmando

In *Asymptote*, ogni comando deve essere seguito da un punto e virgola (;). Questo permette all'interprete di separare un comando dal successivo. Pertanto, due o più comandi non hanno bisogno di comparire su righe separate. Gli spazi bianchi prima e dopo i comandi sono ignorati da *Asymptote*, quindi si può fare uso dell'indentazione dei comandi per migliorare la leggibilità del codice.

Il testo di commento, che non va ad influenzare l'output, può essere inserito nel codice secondo le regole del C++, facendolo iniziare con la doppia barra // o circondandolo con la coppia /* ... */.

Un esempio di codice minimale che disegna un segmento è il seguente:

```
/* Disegniamo un segmento dal punto
(0,0) al punto (50,50) */
draw((0,0)--(50,50)); // fatto!
```

La funzione di disegno `draw` è discussa più avanti.

Come avviene spesso nei testi introduttivi dei linguaggi di programmazione, è bene anche qui mostrare la semplicità del codice *Asymptote* necessario a stampare la frase tormentone "Hello world!". Esso consiste nella sola istruzione:

```
label("Hello world!", (0,0), N);
```

e fa uso della funzione `label`. La funzione è stata invocata richiedendo di posizionare la stringa di testo desiderata a Nord (N) del punto di coordinate (0,0). L'aspetto importante da sottolineare qui è che la stringa di testo viene processata di default dal programma `latex` oppure, a richiesta, da `pdflatex`. L'esempio seguente mostra la possibilità di personalizzare il meccanismo di composizione del testo con dei comandi LATEX che si inserirebbero nel preambolo di un documento:

```
usepackage("guit");
usepackage(
    "siunitx", // nome pacchetto
    "decimalsymbol=comma"); // opzione
texpreamble("\newcommand{\anno}{2009}"
);
label("\Large\GuITmeeting
[style=display, year=\anno]",
(0,0), N);
label(
"$\ell=\SI{4.5}{\centi\meter}$",
```

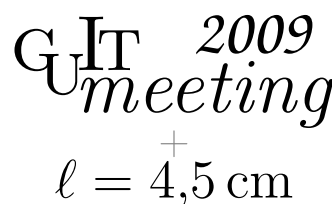


FIGURA 1: Comporre etichette in *Asymptote* con l'ausilio di pacchetti di estensione LATEX, attraverso le funzioni `usepackage` e `texpreamble`.

(0,0),S);

Questo codice realizza l'immagine della figura 1. La funzione predefinita `usepackage` permette di caricare un pacchetto di estensione di L^AT_EX nel contesto del processo di composizione delle annotazioni (ed eventualmente di specificare delle opzioni). La funzione `texpreamble` permette di inserire una determinata stringa nel preambolo. Nel caso precedente si è definita la macro `\anno`. Nel caso in cui si ha l'esigenza di utilizzare un preambolo più articolato e di maggiore complessità sarà conveniente preparare un file, ad esempio `my_preamble.tex`, salvarlo nella stessa cartella di lavoro in cui risiede il sorgente *Asymptote* ed invocare la funzione come segue:

```
texpreamble("\input{my_preamble}");
```

Una delle caratteristiche più utili del linguaggio *Asymptote* è la possibilità di definire nuove funzioni. Il seguente esempio risulterà senz'altro comprensibile anche ai programmatori meno esperti:

```
void drawcross(real x1, real y1,
               real x2, real y2)
{
    draw((x1,y1)--(x2,y2));
    draw((x1,y2)--(x2,y1));
}
```

Nelle intenzioni del programmatore la funzione `drawcross` potrebbe voler disegnare due segmenti che si incrociano. La parola chiave `void` significa che la funzione non ritorna nessun valore (è una procedura). La parola chiave `real` determina il tipo degli argomenti di scambio. I tipi di dati predefiniti in *Asymptote* sono discussi più avanti.

4.1 Definire il sistema di coordinate

I comandi di disegno in *Asymptote* si basano sulla definizione di un opportuno sistema di coordinate. Ogni punto della "tela" (*canvas*) del disegno è inteso come una coppia (x, y) di coordinate. Ci sono diversi modi per scegliere un sistema di coordinate cartesiane nel piano. Bisogna scegliere il posizionamento dell'origine e la scala di ciascuno degli assi. Per default, l'unità di lunghezza in entrambe le direzioni x ed y è il cosiddetto *PostScript bigpoint* (`bp`), che corrisponde ad $1/72$ di pollice. Quindi, se non si modifica la scala del disegno, i punti $(0,0)$ e $(72,0)$ distano esattamente un pollice quando vengono disegnati in *Asymptote*.

Il disegno in unità *bigpoint* può risultare scomodo, ad esempio se si vogliono disegnare figure ed oggetti di dimensioni definite naturalmente in centimetri. La funzione `unitsize` può essere utilizzata per specificare l'unità di lunghezza lungo ciascun lato dell'immagine. Questa funzione richiede fino a 3 argomenti: l'immagine che si vuole scalare, (se questa non è specificata, significa che si vuole scalare la figura corrente), l'unità di lunghezza nella direzione x , l'unità di lunghezza in direzione y .

Se la funzione è invocata con un solo argomento numerico si avrà che per entrambe le direzioni si avrà la stessa unità di lunghezza. In tal modo il comando `unitsize(72)` dirà ad *Asymptote* che, da quel punto in poi, l'unità di lunghezza è di 1 pollice in entrambe le direzioni coordinate.

Asymptote offre la possibilità di operare con diverse unità predefinite: il punto (`pt`, pari a $1/72,27$ pollici), il pollice (`inch`), il centimetro (`cm`), il millimetro (`mm`). Un comando equivalente a quello precedente è dunque: `unitsize(1inch)`.

Un'altra funzione utile è `size` che specifica la larghezza e l'altezza finali del disegno. La stessa funzione può essere invocata con un solo argomento, ottenendo che sia la larghezza che l'altezza dell'immagine corrisponderanno a questa dimensione. Per esempio, il comando `size(3cm,3cm)` o semplicemente `size(3cm)` permette di scalare il disegno in modo che rientri di una casella di dimensioni $3\text{ cm} \times 3\text{ cm}$, a prescindere da quanto specificato da eventuali chiamate alla funzione `unitsize`.

Se si scrive:

```
unitsize(1cm)
// size(15mm,10mm)
draw((0,0)--(5,5));
```

si otterrà un'immagine di dimensioni $5\text{ cm} \times 5\text{ cm}$. Basta commentare la riga con il comando che invoca la funzione `size` per ottenere la scalatura del disegno ed un'immagine finale di dimensioni $15\text{ mm} \times 10\text{ mm}$.

Se uno degli argomenti di `size` è 0 l'algoritmo di scalatura renderà uno due lati dell'immagine di lunghezza pari a quella specificata; ad esempio `size(0,3cm)` permette di scalare il disegno in modo che l'altezza sia pari a 3 cm, dimensionando proporzionalmente la dimensione corrispondente all'argomento nullo.

Nella figura 2 è riportato un esempio di codice sorgente che genera il disegno di un sistema di assi cartesiani. Le funzioni utilizzate a configurare le dimensioni degli assi, le etichette, le tacche principali e quelle secondarie, eccetera, sono `xaxis` ed `yaxis`. Un esempio delle funzionalità introdotte dal modulo `geometry.asy` è mostrato nella figura 4. Si esamini l'uso dei nuovi tipi di dato, `point` e `vector`, introdotti da questo modulo. Il codice usato per generare il disegno mostra anche esempi di conversione implicita di tipi di variabili (*casting*), da `point` a `vector` oppure da `point` a `pair`.

4.2 Variabili e tipi di dati

L'interprete di *Asymptote* attua un meccanismo di *parsing* del codice sorgente ispirato a quello dei compilatori C++. Ciò consente agli utenti di utilizzare diversi tipi di dati predefiniti e di crearne di nuovi (secondo il paradigma della programmazione a oggetti).

```

import graph;
unitsize(1cm);
// Una curva
path curva=(.5,2){dir(-30)}..{0}(2.5,-2)..{dir(30)}(5,3);
draw(curva,red);
dot(curva,black);
// Configurazione degli assi
xaxis(Label("$x$",position=EndPoint, align=SE),
xmin=-1.5,
Ticks(scale(.7)*Label(align=W),
NoZero,
endlabel=false,
Step=1,step=.25,
end=false,
Size=1mm, size=.5mm,
pTick=black,ptick=gray),
Arrow);
yaxis(Label("$y$",position=EndPoint, align=NE),
ymin=-3.5,ymax=3.5,
Ticks(scale(.7)*Label(),
NoZero,
Step=1,step=.25,
Size=1mm, size=.5mm,
pTick=black,ptick=gray),
Arrow);

```

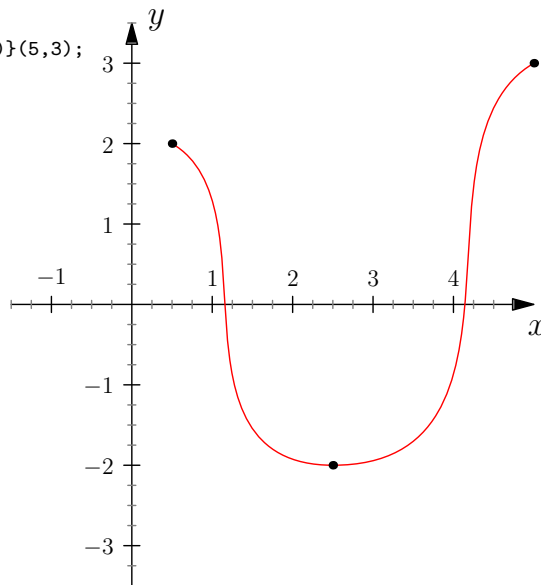


FIGURA 2: Disegnare un riferimento cartesiano e configurare lo stile degli assi.

4.2.1 Variabili numeriche e coordinate

Introduciamo con un esempio la dichiarazione e l'assegnamento di variabili:

```

unitsize(1mm);
pair A, B, C, D;
A = (0, 0); B = (50,50);
int yc = 50;
real xd = 50.0;
C = (0,yc); D = (xd, 0);
draw (A--B);
draw (C--D);

```

In *Asymptote* l'assegnamento di valore ad una variabile può anche avvenire contestualmente alla sua dichiarazione come nel frammento di codice seguente:

```

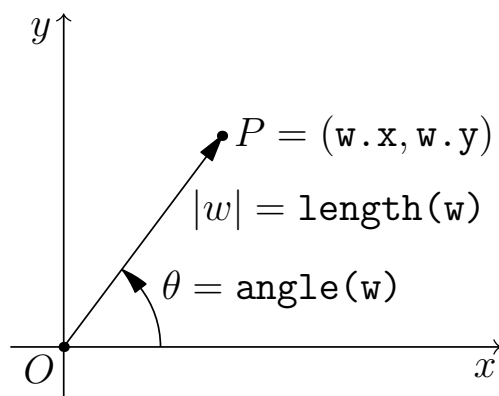
real d=50;
pair A=(0,0), B=(d,d);

```

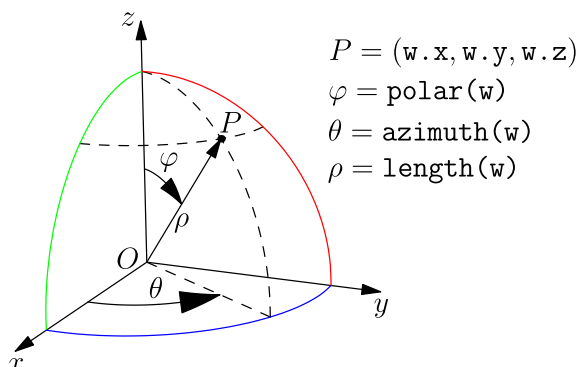
```
draw(A--B);
```

cioè il programmatore può inizializzare esplicitamente una variabile.

Oltre ai tipi numerici `int` e `real`, comuni ad altri linguaggi di alto livello, nei frammenti di codice precedenti si riconosce la dichiarazione di variabili di tipo `pair`. Una variabile `pair` non è altro che una coppia di coordinate nel sistema di riferimento specificato (nel caso dell'esempio, il sistema di coordinate di default). In *Asymptote* una `pair` è trattata come numero complesso, cioè una coppia ordinata (x, y) di elementi reali. Definita una variabile `w` appartenente a questo tipo, corrispondente al numero complesso w , la sua parte reale e la sua parte immaginaria sono date da `w.x` e `w.y`. Le istruzioni `length(w)` e `degree(w)` ritornano valori reali corrispondenti, rispettivamente, al modulo $|w|$ ed all'angolo di fase (in gradi) nell'intervallo



(a) Un punto P del piano complesso.



(b) Un punto P dello spazio tridimensionale.

FIGURA 3: Interpretazione di una variabile di tipo `pair` e di tipo `triple`.

```

import geometry;
size(7cm,0);
usepackage("amsmath");
show("$0$",
    "\boldsymbol{i}",
    "\boldsymbol{j}",currentcoordsys);
vector u=(1.5,0.5), v=rotate(90)*u;
point O1=(5,3);
coordsys R=cartesiansystem(O=O1,i=u,j=v);
show("$0_1$",
    "\boldsymbol{u}",
    "\boldsymbol{v}", R, xpen=invisible);
point M=(1,0.5);
dot("$M$", M);
point P=point(R, (1,1));
dot("$P$", P);
vector w=P;
show("\boldsymbol{w}", w);
point Q=M+w;
dot("$Q=M+\boldsymbol{w}", align=Relative(N),Q);
draw((0,0)--M,Arrow);
draw("\boldsymbol{w}",M--Q,RightSide,EndArrow);

```

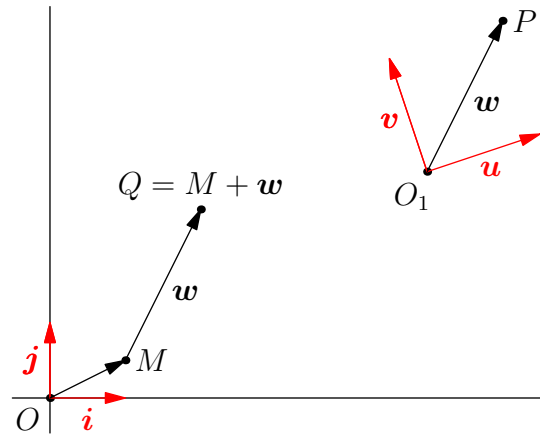


FIGURA 4: Introdurre un nuovo riferimento, definire un vettore e sommare un punto ad un vettore con le funzioni definite dal modulo `geometry.asy`.

$[0, 360[$ (la funzione ritorna un *warning* se w coincide con l'origine). Una funzione analoga a `degree` è `angle` che ritorna un angolo in radianti nell'intervallo $[0, 2\pi[$. Questi concetti sono resi in forma grafica nella figura 3a.

Il tipo `triple` è analogo a `pair` e serve per definire terne di coordinate (x, y, z) . La figura 3b rappresenta un'interpretazione grafica delle funzioni `length(triple)`, `polar(triple)` e `azimuth(triple)`. Esse estraggono le coordinate polari di un punto P rappresentato dalla variabile w di tipo `triple`. Funzioni che ritornano valori in gradi analoghe a `polar` e `azimuth` sono, rispettivamente, `colatitude` e `latitude`. Si rimanda al manuale utente per i dettagli sui costruttori e sulle funzioni di manipolazione di variabili `triple`.

I tipi di dati predefiniti più comunemente usati in *Asymptote* sono riportati nella tabella 1. Nel prossimo paragrafo, che introduce le funzioni di disegno più importanti, verranno presentati alcuni esempi che fanno uso dei tipi `path` e `guide`, necessari per la gestione di tracciati.

4.2.2 Tipi definiti da utente

A titolo di esempio di programmazione a oggetti in *Asymptote*, si riporta un estratto dal file `geometry.asy` che mostra la definizione del tipo `point`. Si tratta di un tipo che non compare tra i tipi predefiniti del linguaggio base. Esso viene definito nell'ambito del modulo `geometry` che attualmente fa parte della distribuzione ufficiale di *Asymptote*. Il tipo `point` è usabile dopo aver dato l'istruzione `import geometry`. La definizione si serve della parola chiave `struct` (analogo alla parola chiave `class` del C++) e fornisce un tipo di dato che contiene informazioni su una `pair` e sul riferimento cartesiano in cui vengono assegnate le

coordinate. Il codice che lo definisce assomiglia al seguente:

```

// Estratto da geometry.asy
struct point
{
    // variabili membro
    coordsys coordsys; // pubblica
    restricted pair coordinates; //
    privata
    restricted real x, y; // private
    // funzione membro di
    // inizializzazione
    void init(coordsys R,
        pair coordinates, real mass)
    {
        this.coordsys=R;
        this.coordinates=coordinates;
        this.x=coordinates.x;
        this.y=coordinates.y;
        this.m=mass;
    }
    /* ...
    coordsys, defaultcoordsys,
    e currentcoordsys
    definite nello stesso modulo */
}

```

Il lettore che ha familiarità con la programmazione a oggetti riconoscerà da questo frammento di codice l'utilizzo dei concetti di *funzioni membro* e di *variabili membro* (pubbliche e private) e della parola chiave `this` (come in C++ o in Java).

Sempre nel file `geometry.asy` è definito uno dei costruttori del tipo `point`:

```

// Un costruttore
point point(coordsys R, pair p,

```

TABELLA 1: Tipi di dati predefiniti del linguaggio *Asymptote*.

<i>Tipo</i>	<i>Descrizione</i>	<i>Esempi</i>
<code>bool</code>	tipo logico, vero o falso	<code>true</code> , <code>false</code> , <code>1>2</code> (<code>false</code>)
<code>string</code>	stringa di caratteri, delimitati da doppi apici	<code>"Ciao"</code> , <code>"Questa e' una stringa"</code>
<code>int</code>	un numero intero	<code>-2</code> , <code>-1</code> , <code>0</code> , <code>1</code> , <code>2</code> , <code>3</code>
<code>real</code>	un numero reale	<code>1.0</code> , <code>5.48</code> , <code>12345.6789</code>
<code>pair</code>	una coppia di numeri reali o interi, delimitati da parentesi tonde	<code>(0,2)</code> , <code>(-30,42.5)</code>
<code>triple</code>	una terna di numeri reali o interi, delimitati da parentesi tonde	<code>(1,2,3)</code> , <code>(-2.5,5,4)</code>
<code>path</code>	una <i>spline</i> cubica non modificabile	<code>(0,0)--(5,0)</code> , <code>(0,1)..(1,0)..(1,1)--cycle</code>
<code>guide</code>	una <i>spline</i> cubica, simile a un <code>path</code> ma aggiustabile se raccordata ad altre curve dello stesso tipo	<code>(0,0)--(5,0)</code> , <code>(0,1)..(1,0)..(1,1)--cycle</code> , <code>(0,1)--(1,0)--(1,1)--cycle</code>
<code>picture</code>	una tela (<i>canvas</i>) su cui disegnare nel sistema di coordinate corrente	<code>currentpicture</code> , qualsiasi insieme di oggetti
<code>frame</code>	una tela (<i>canvas</i>) su cui disegnare in coordinate PostScript	<code>currentpicture</code> , qualsiasi insieme di oggetti
<code>transform</code>	una trasformazione del piano, applicabile a qualsiasi oggetto disegnabile usando l'operatore <code>*</code>	<code>scale(2)</code> , <code>xscale(2)</code> , <code>rotate(30)</code> , <code>rotate(15,(-1,3))</code> , <code>shift(2,4)</code>
<code>void</code>	usato per dichiarare funzioni che non hanno argomenti	

```

        real m=1) // val. di
    default
{
    point op;
    op.init(R, p, m);
    return op;
}

```

In base a queste definizioni una variabile di questo tipo può essere inizializzata come segue:

```

// CS, rif. definito precedentemente
point pt=point(CS, (2,5));

```

ed, eventualmente, si può fare un controllo sul sistema di coordinate al quale si riferiscono i valori delle variabili private `x` e `y`:

```

if (pt.coordsys == defaultcoordsys){
    // riferimento predefinito
    ...
} else {
    // abbiamo un nuovo riferimento
    ...
}

```

Per un ulteriore esempio d'uso del tipo `point` si veda anche la definizione del sistema cartesiano `R` e del punto `P` nel codice che genera il disegno della figura 4. In quell'esempio il punto `M` è un `point` definito nel sistema di riferimento di default (che è quello corrente) come:

```

point M=(1,0.5); // cast da pair a
point

```

mentre `P` è definito assegnando coordinate nel nuovo riferimento `R`:

```

point P=point(R, (1,1)); // costruttore

```

4.2.3 Array di variabili

Aggiungendo `[]` nella dichiarazione di variabili predefinite o in variabili di tipo definito da utente si possono costruire degli array (matrici di dati di una o più dimensioni). L'istruzione:

```

real[] A={-0.5,1.5,2};

```

dichiara ed inizializza l'array `A`. Si può accedere all'elemento `i`-mo della matrice unidimensionale `A` con l'operazione `A[i-1]` (come in C, il primo elemento dell'array `A` è l'elemento `A[0]`).

Per default, i tentativi di accesso o di assegnamento che utilizzano un indice negativo generano un errore. Anche l'accesso ad un elemento di un array con un indice che va oltre la dimensione della matrice genera un errore. Tuttavia, l'assegnazione che fa uso di un indice maggiore del riempimento corrente dell'array provoca il ridimensionamento della matrice in modo da allocare il nuovo elemento. L'istruzione `A[6] = -1.0;` provoca il ridimensionamento di `A`, che avrà 6 elementi di cui il quarto e quinto nulli.

Gli array di *Asymptote* hanno caratteristiche che li rendono simili ai contenitori `vector` della libreria standard del C++. Si veda il manuale utente per le funzioni di manipolazione degli array e per i dettagli sulle matrici a più dimensioni.

```

int b=6; // b>3
for(int a=0; a<b-2; ++a) {
    draw ((a,a)--(a,a+1)--(a+1,a+1));
    label(format("$u_{%i}$",a),(a+.5,a+1),N);
}
for(int a=b-2; a<b; ++a) {
    draw ((a,a)--(a,a+1)--(a+1,a+1),dashed);
}
for(int a=0; a<2; ++a) {
    draw ((b+a,b+a)--(b+a,b+a+1)--(b+a+1,b+a+1));
    if (a != 0)
        label(format("$u_{n+%i}$",a),
            (b+a+.5,b+a+1),N);
    else label("$u_n$", (b+a+.5,b+a+1),N);
}
    
```

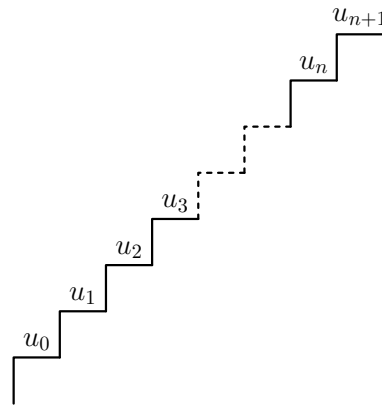


FIGURA 5: Esempio di disegno ottenuto con un ciclo iterativo.

4.3 Costrutti

Ecco un breve esempio che dimostra la somiglianza delle strutture di controllo del linguaggio *Asymptote* con quelle del C, del C++ e di Java:

```

// Una variabile reale
real x=1.0;
// Un test condizionale
if (x == 1.0) {
    // stampa un messaggio
    // sulla console
    write("x uguale a 1.0");
} else {
    write("x diverso da 1.0");
}
// Un loop di 10 iterazioni
for(int i=0; i<10; ++i){
    write(i);
}
    
```

Asymptote mette a disposizione anche i costrutti **while do**, **break** e **continue**, proprio come in C++. Inoltre, come in Java, il linguaggio permette di scrivere elegantemente dei cicli su tutti gli elementi di un *array*:

```

// Itera sugli elementi di
// un array
int[] myarray={1,1,2,3,5};
for(int k : myarray) {
    write(k);
}
    
```

Un esempio più articolato di disegno basato su un ciclo iterativo è mostrato nella figura 5. Esso utilizza la funzione **format** per formattare stringhe destinate alla composizione tipografica.

4.4 Funzioni di disegno

Tutte le capacità grafiche di *Asymptote* si basano su quattro funzioni primitive (comandi). I tre comandi PostScript **draw**, **fill** e **clip** servono ad aggiungere oggetti ad un disegno. Il quarto comando **label** viene utilizzato per aggiungere annotazioni testuali (e immagini EPS esterne). Una

quinta funzione molto utile è **shipout** e si usa tipicamente come ultimo comando, dopo aver creato il disegno, per aggiustare alcune impostazioni di default sul formato finale. Anche se i comandi di disegno accettano molte opzioni, queste ultime hanno tutte delle assegnazioni di default ragionevoli e richiedono un controllo esplicito solo in caso di particolari esigenze di disegno.

I quattro comandi di disegno fondamentali di *Asymptote* sono ampiamente documentati nel manuale d'uso. Nel prossimo paragrafo vedremo qualche dettaglio sull'uso della funzione **draw**.

4.4.1 Punti, segmenti e stili

Una delle funzioni predefinite più importanti del linguaggio è **draw**. Dall'esame dei frammenti di codice riportati fino a questo punto si comprende che il suo utilizzo è molto intuitivo.

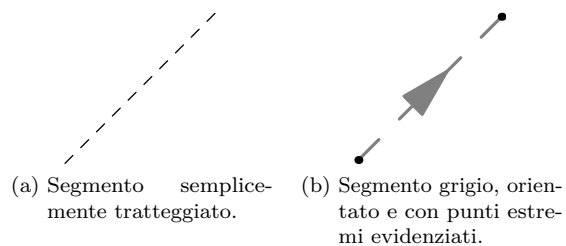


FIGURA 6: Disegnare un segmento in modi diversi.

Consideriamo un semplice esempio: disegnare un segmento che va dal punto *A* al punto *B* del piano. In *Asymptote* si possono definire due variabili di tipo **pair** che memorizzano le coordinate degli estremi del segmento, ad esempio *A* e *B*, utilizzando direttamente nella chiamata alla funzione **draw**. Nel disegno del segmento *AB* l'istruzione fondamentale è **draw(A--B)**. In questo comando viene passato a **draw** un solo argomento, cioè il risultato dell'operazione *A--B*.

L'operatore **--**, così come accade in METAPost oppure in **pgf**, determina l'unione di due punti del piano attraverso una linea. In *Asymptote* questa

operazione produce un tracciato che può essere memorizzato nella sua interezza come tipo di dato. Un possibile tipo predefinito che memorizza il tracciato A--B è il tipo `path`. Disporre di una variabile che memorizza un tracciato consente di riusare quest'oggetto in tutte le funzioni che concettualmente ne fanno uso, che siano esse predefinite o definite dall'utente.

Volendo gestire il segmento come un `path`, si può scrivere:

```
pair A=(0,0), B=(50,50);
path p=A--B;
draw(p,dashed);
```

ed ottenere il disegno della figura 6a. Alla funzione `draw` viene passato un secondo argomento (`dashed`) che serve a specificare lo stile della linea.

Il segmento della figura 6b è ottenuto con le istruzioni:

```
draw(p, // tracciato
      gray+dashed+1.2pt, // pennino
      MidArrow); // freccia
dot(p);
```

Il secondo argomento mostra come sia intuitivo definire lo stile della linea utilizzando l'operatore `+` per sommare logicamente una lista di attributi. In questo caso si è specificato: il colore grigio (`gray`), il tratteggio (`dashed`) e lo spessore (`1.2pt`).

Le diverse chiamate a `draw` mettono in evidenza la possibilità offerta da *Asymptote* di definire funzioni invocabili con un numero variabile di argomenti. Ciò è possibile perché molte funzioni in *Asymptote* sono definite assegnando agli argomenti dei valori di default (ad esempio gli argomenti che controllano lo stile dei tracciati o il posizionamento delle etichette). Gli argomenti hanno spesso dei nomi predefiniti che consentono il loro passaggio *per nome*, oltre che per posizione, semplificando il lavoro del programmatore. Un esempio evidente è dato dalla chiamata:

```
coordsys R=cartesiansystem(
  0=01, i=u, j=v);
```

effettuata nel codice sorgente riportato nella figura 4. Le variabili `01`, `u` e `v` sono state precedentemente definite e vengono passate alla funzione `cartesiansystem` come gli argomenti di nome `0`, `i` e `j`, rispettivamente.

La funzione `dot` serve a disegnare punti. Nell'esempio che genera la figura 6b essa viene invocata come `dot(p)`, passandole come argomento un `path` ed ottenendo il disegno di tutti i punti (nodi) che definiscono il percorso memorizzato nella variabile `p`. Questa funzione può essere anche chiamata, come è naturale che sia, passando una `pair`, ad esempio con l'istruzione `dot((20,0))`, per disegnare un semplice puntino. In realtà, come vedremo più avanti, grazie alla rappresentazione generica di un tracciato in *Asymptote*, un `path` definito da un solo nodo (un percorso degenerare, di lunghezza nulla) si identifica con quel punto stesso. È lecito

cioè dire: `path p=(20,0)`. Pertanto, il disegno di un singolo puntino corrisponde all'applicazione di `dot` ad un `path` degenerare.

L'ultimo esempio mostra il passaggio della specifica `MidArrow` come terzo argomento della funzione `draw`. Lo stesso risultato si ottiene con l'istruzione

```
draw(p,gray+dashed+1.2pt,
      Arrow(Relative(0.5)));
```

che utilizza le funzioni `Arrow` e `Relative`. Quest'ultima serve a controllare la posizione lungo il tracciato in cui si vuole disegnare la freccia (in questo caso nella posizione intermedia).

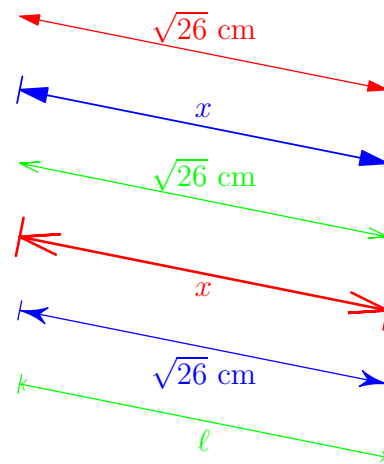
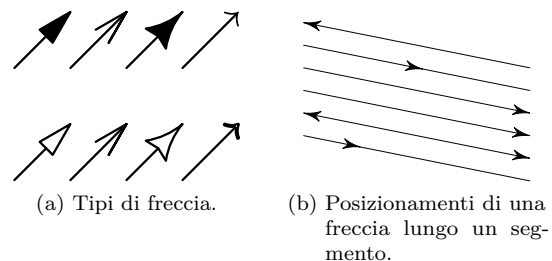


FIGURA 7: Diversi stili di freccia.

Esempi di diversi stili di freccia sono riportati nella figura 7. Il seguente frammento di codice realizza il disegno della figura 7a:

```
// 4 tipi di freccia
draw((0,0)--(1,1),2bp+black,
      Arrow);
draw((1,0)--(2,1),2bp+black,
      Arrow(SimpleHead));
draw((2,0)--(3,1),2bp+black,
      Arrow(HookHead));
draw((3,0)--(4,1),2bp+black,
      Arrow(TeXHead));
// Varianti (option NoFill)
draw((0,-2)--(1,-1),2bp+black,
      Arrow(NoFill));
draw((1,-2)--(2,-1),2bp+black,
      Arrow(SimpleHead,NoFill));
draw((2,-2)--(3,-1),2bp+black,
```

```

Arrow(HookHead,NoFill));
draw((3,-2)--(4,-1),2bp+black,
Arrow(TeXHead,NoFill));

```

Il disegno della figura 7b si ottiene con la seguente sequenza di istruzioni:

```

DefaultHead=HookHead; // N.B.
draw((0,1)--(5,0),BeginArrow());
draw((0,.5)--(5,-.5),MidArrow());
draw((0,0)--(5,-1),EndArrow());
draw((0,-.5)--(5,-1.5),Arrow());
draw((0,-1)--(5,-2),Arrows());
draw((0,-1.5)--(5,-2.5),
Arrow(Relative(.25)));

```

Il disegno della figura 7c si ottiene come segue:

```

unitsize(1cm);
draw("$\sqrt{26}$ cm",
(0,5)--(5,4),
N,red,Arrows);
draw("$x$",
(0,4)--(5,3),
N,.7bp+blue,Arrows,Bars);
draw("$\sqrt{26}$ cm",
(0,3)--(5,2),
N,green,Arrows(SimpleHead));
draw("$x$",
(0,2)--(5,1),
S,1bp+red,Arrows(SimpleHead),
Bars);
draw("$\sqrt{26}$ cm",
(0,1)--(5,0),
S,blue,Arrows(HookHead),
Bars);
draw("$\ell$",
(0,0)--(5,-1),
S,green,Arrows(TeXHead), Bars);

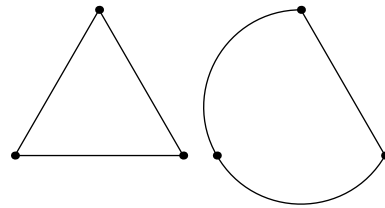
```

4.4.2 Tracciati

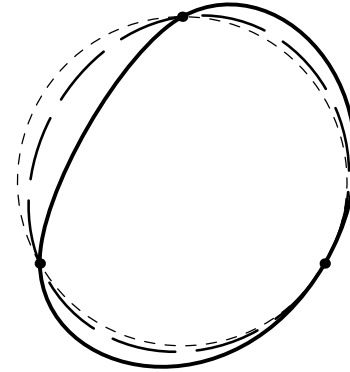
Un percorso di tipo `path` (o `guide`) è rappresentato internamente in *Asymptote* come una curva *spline* cubica a tratti di parametro t , con t che varia tra 0 ed n (numero di *nod*i). Questa è una rappresentazione abbastanza generale da permettere di creare e manipolare un gran numero di tracciati diversi.

Un caso particolarissimo di tracciato è quello di lunghezza nulla, costituito da un solo nodo (punto). Come già detto al paragrafo precedente, l'assegnazione `path p=(0,0)` è perfettamente lecita. Inoltre, due punti (o percorsi di lunghezza nulla) possono essere semplicemente uniti collegandoli con una linea retta. Pertanto, detto anche q un percorso degenero definito come `path q=(1,1)`, il semplice tracciato dato dal segmento che unisce i due punti è realizzato dall'operazione: `p--q`.

L'operatore di doppio trattino `--` unisce, in generale, due tracciati con una linea retta. È ciò che accade nel disegno del triangolo della figura 8a ottenuta con il codice seguente:



(a) Due tracciati chiusi definiti con gli stessi nodi.



(b) Alterazione della tensione della curva nel tratto tra i primi due nodi.

FIGURA 8: Tracciati ed operazioni su di essi.

```

pair p1=(0,0), p2=(1/2,sqrt(3)/2),
p3=(1,0);
path t1, t2;
t1=p1--p2--p3--cycle;
t2=p1..p2..p3..cycle;
t2=shift(1.2*right)*t2;
draw(t1); dot(t1);
draw(t2); dot(t2);

```

Si noti come sia naturale in *Asymptote* utilizzare funzioni matematiche per la definizione di punti particolari come p_2 . La parola chiave `cycle` è usata per chiudere un tracciato unendo l'ultimo nodo con il primo.

Il tracciato `t2` dell'ultimo listato mostra un esempio di collegamento tra punti tramite curve di Bezier ottenuto con l'operatore di doppio punto (`p..q`). Si rimanda al manuale di *Asymptote* ed alla letteratura ivi citata per un approfondimento sugli algoritmi di disegno dei tracciati curvi.

La figura 8b mostra come sia possibile modificare la tensione della curva *spline* che unisce tre punti. Il disegno è ottenuto con il codice seguente:

```

path t1, t2, t3;
t1=p1..p2..p3..cycle;
t2=p1..tension 1.1 ..p2..p3..cycle;
t3=p1..tension 1.5 ..p2..p3..cycle;
draw(t1,dashed); dot(t1,black+4bp);
draw(t2,linetype("24 8")+1bp);
draw(t3,black+1.5bp);

```

Ogni tratto costruito con l'operatore `..` ha un valore di tensione predefinito pari ad 1. Con la parola chiave `tension` si può controllare questo

parametro che accetta valori maggiori o uguali di 0,75. Nell'esempio precedente la curva iniziale viene modificata variando semplicemente la tensione del tratto che unisce i primi due nodi, p1 e p2. Più è alto il valore della tensione e più quel tratto di tracciato si avvicina ad una linea retta.

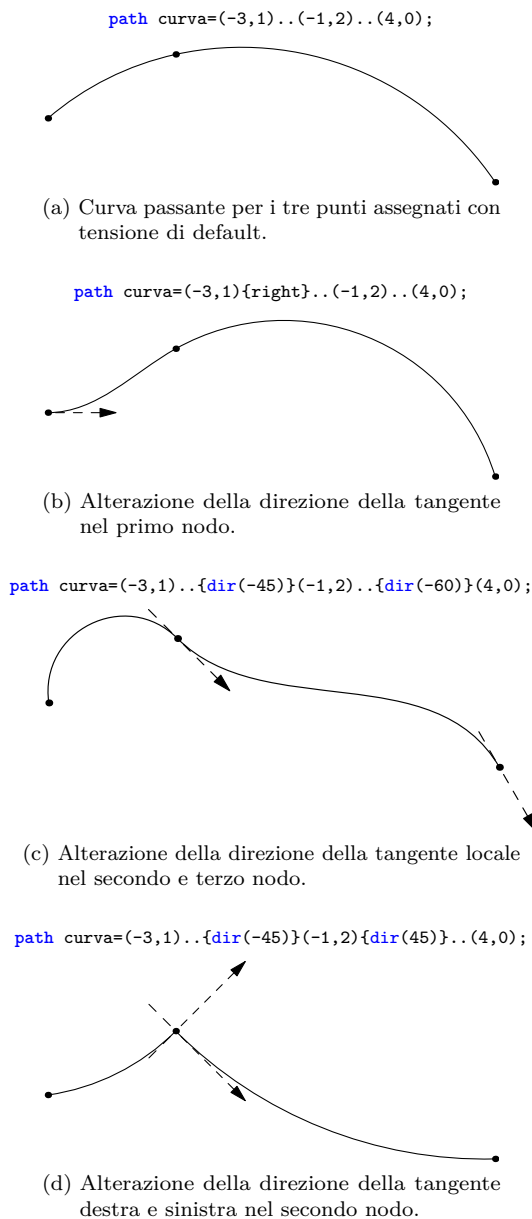


FIGURA 9: Controllo delle tangenti ad una curva nei nodi.

La figura 9 mostra esempi di controllo delle tangenti ad un tracciato in corrispondenza dei nodi. Nei disegni viene riportato il comando che definisce la variabile `curva`. Si osservi l'uso della funzione `dir` per la specifica della pendenza in gradi della tangente.

La differenza tra i tipi di tracciato `path` e `guide` si comprende quando ci si pone il problema di unire due curve definite inizialmente in maniera separata. Si consideri il disegno della figura 10a che rappresenta due linee spezzate. Esse sono generate dal codice seguente:

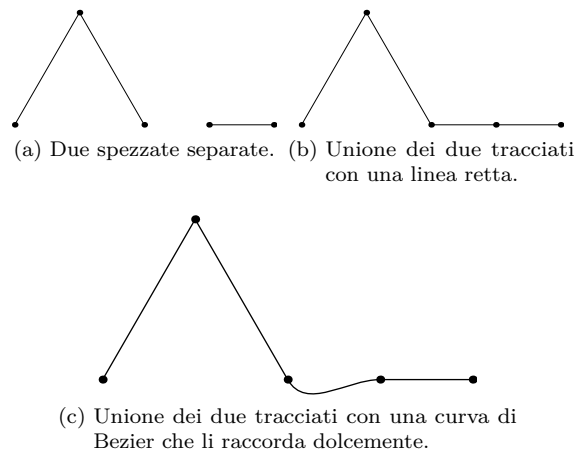


FIGURA 10: Tracciati ed operazioni su di essi.

```
path t1, t2;
t1=(0,0)--(1/2,sqrt(3)/2)--(1,0);
t2=(1.5,0)--(2,0);
draw(t1); dot(t1);
draw(t2); dot(t2);
```

La semplice unione dei due tracciati `t1` e `t2`, entrambi di tipo `path`, con un segmento è ottenuta con l'operazione `t1--t2`. Il codice seguente genera la figura 10b:

```
path t3=t1--t2;
draw(t3); dot(t3);
```

Il raccordo con una curva di Bezier che sia tangente ad entrambi i tracciati si ottiene con l'operazione `t1..t2`. Il codice seguente genera la figura 10c:

```
path t4=t1..t2;
draw(t4); dot(t4);
```

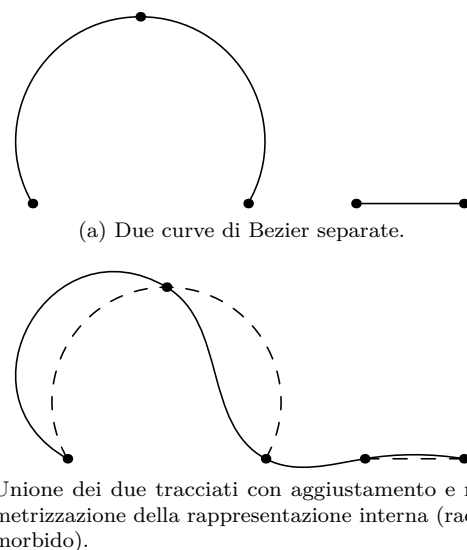


FIGURA 11: Tracciati aggiustabili (guide).

Nelle precedenti operazioni di unione di due tracciati le curve iniziali si comportano come se fossero rigide. Se queste fossero definite come `guide`, a

seguito di operazioni di unione esse sarebbero soggette ad una riparametrizzazione con conseguente modifica dell'aspetto.

Un esempio di guide è rappresentato dalla figura 11. I due tracciati della figura 11a sono definiti come segue:

```
guide t1, t2;
t1=(0,0)..(1/2,sqrt(3)/2)..(1,0);
t2=(1.5,0)..(2,0);
```

Il raccordo delle due guide con una curva di Bezier è effettuato ancora una volta dall'operazione di concatenazione `t1..t2`. Il risultato è quello della figura 11b che è generato dal codice

```
path t3=t1..t2; // cast guide<-path
draw(t3); dot(t3);
```

L'esempio che segue dimostra l'uso della funzione predefinita `intersectionpoint`. Essa restituisce in generale un array di `pair` corrispondenti ai punti di intersezione tra due tracciati. L'esempio proposto qui è molto semplice, ed è il seguente:

```
unitsize(1cm); size(6cm,0);
usepackage("siunitx",
"decimalsymbol=comma");
path segm1=(-1,1)--(5,2),
      segm2=(0,3)--(2,-0.5);
draw(segm1^^segm2,blue);
pair pX=intersectionpoint(segm1,segm2)
;
dot(pX,4bp+red);
string sX= // formatta e concatena
format("\SI{%.3f}{\centi\meter}",
pX.x)+
format("\SI{%.3f}{\centi\meter})",
pX.y);
label(sX,pX,0.9S+2E);
dot((0,0));
label("\$(0,0)\$", (0,0),N);
```

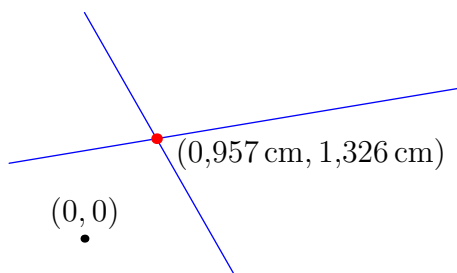


FIGURA 12: Esempio di intersezione tra due tracciati ed estrazione delle coordinate.

Il risultato è il disegno della figura 12 in cui si è ricercata l'intersezione tra due segmenti. L'esempio mostra anche un'interessante applicazione della funzione `format`, utilizzata per la composizione dell'annotazione che mostra le coordinate, non note *a priori*, del punto di intersezione `pX`.

4.4.3 Trasformazioni

Molte delle funzioni predefinite di *Asymptote* fanno ampio uso di trasformazioni affini (mutuate dalle analoghe funzioni di trasformazione del linguaggio PostScript).

Le trasformazioni possono essere applicate a variabili di tipo `pair`, `guide`, `path`, `pen`, `string`, `frame` e `picture`. Una trasformazione è applicata sotto forma di moltiplicazione a sinistra, attraverso l'operatore `*`. Le trasformazioni possono essere composte l'una con l'altra e memorizzate in variabili di tipo `transform`. L'inizializzazione implicita di tali variabili assegna ad esse la trasformazione predefinita `identity` (di ovvio significato). Una variabile `T` di tipo `transform` può trasformare, ad esempio, un tracciato `crv` (`path` o `guide`) attraverso l'operazione `T*crv`.

Il codice che segue è quello che realizza il disegno della figura 13:

```
size(4cm,0);
import markers;
// Due punti, A e B
pair pA=(0,0),pB=(4,2);
dot("\$A\$",pA,SW); dot("\$B\$",pB,SE);
// B1 per rotazione di B intorno ad A
// di un angolo di 60 gradi
pair pB1=rotate(60,pA)*pB;
// disegno
dot("\$B'\$",B1,N,red);
draw(pB--pA--pB1,green);
// marcatura dell'angolo
markangle(scale(1.5)*"+60^\circ",
radius=50,
pB,pA,pB1,ArcArrow(2mm),
1mm+red);
```

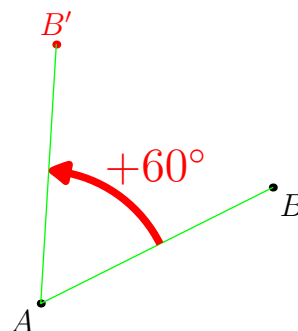


FIGURA 13: Esempio di trasformazione di rotazione applicata ad un punto e di scalatura di una casella di testo.

Per la rotazione di 60° del punto `B` intorno al punto `A` viene usata la funzione `rotate`. L'ingrandimento dell'annotazione `" $+60^\circ$ "` è prodotto applicando la funzione `scale` direttamente alla stringa passata alla funzione `markangle`.

Una utile funzione di trasformazione è quella che permette di proiettare un dato punto su una retta. Il codice seguente genera il disegno della figura 14:

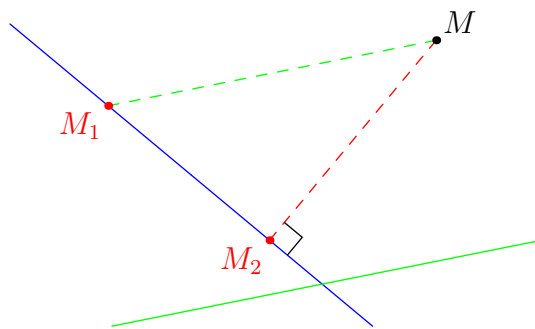


FIGURA 14: Proiettare un punto ortogonalmente ad una retta e secondo una direzione prefissata.

```
import geometry;
size(7cm,0);
// i punti assegnati
point A=(2,1),B=(-1,3.5), // direz. r
      C=(3,1.5),D=(.5,1), // direz. d
      M=(3,4); // punto da proiettare
// linee
line r=line(A,B), d=line(C,D);
// risultati delle proiezioni su r
pair M1=projection(r,d)*M, // lungo d
     M2=projection(r)*M; // ortogonale
// Costruzioni
draw(r,blue); draw(d,green);
draw(segment(M,M1),dashed+green);
draw(segment(M,M2),dashed+red);
perpendicularmark(
    line(M,M2),line(A,B),quarter=2);
dot("$M$",M,NE);
dot("$M_1$",M1,SW,red);
dot("$M_2$",M2,SW,red);
addMargins(1cm,1cm);
```

In questo esempio sono stati usati tipi e funzioni messi a disposizione dal modulo `geometry`: i tipi `line` e `segment` e le funzioni `projection` e `perpendicularmark`.

4.4.4 Pennini e tipi di tratto

In Asymptote, è possibile fornire un contesto per i quattro comandi di disegno di base attraverso un tipo predefinito detto `pen` (pennino).

Una variabile `pen` può essere utilizzata per specificare i seguenti attributi di disegno: il colore, il tipo di linea, lo spessore della linea, l'estremità della linea, il tipo di linea di unione, regole di riempimento di regioni piane. Inoltre è possibile agire con una `pen` su alcuni aspetti legati alla composizione delle etichette testuali come: l'allineamento del testo, il font e la sua dimensione.

Il pennino utilizzato dalla routine di disegno, quando non vi è il controllo esplicito da parte dell'utente, è chiamato `currentpen`. L'inizializzazione implicita di una variabile `pen` assegna il valore `defaultpen`.

Gli attributi di un pennino possono essere assegnati cumulandoli attraverso l'uso dell'operatore non associativo binario `+`. Così, si può ottenere un'impostazione del pennino corrispondente ad un tratteggio rosso di spessore pari a 2pt specificando il valore `dashed+red+2pt`.

L'operatore binario `*` può essere usato per scalare un colore con un numero reale, fino alla saturazione di una o più componenti. Per le funzionalità di manipolazione del colore si vedano, tra le altre, le funzioni `gray`, `rgb`, `cmymk` nel manuale utente di *Asymptote*.

Una funzione utile ad impostare il tipo di tratteggio è `linetype(string s, real offset=0)`, che ritorna un valore di tipo `pen`. Il primo argomento è il più importante e specifica tramite una stringa il tipo di tratteggio. Ad esempio, il codice seguente mostra come creare uno stile personalizzato di "tratto e punto":

```
// Tratteggio:
// 16 continuo, 8 vuoto,
// 1 continuo, 8 vuoto
pen mydashdotted=linestyle(
    "16 8 1 8");
draw((0,0)--(5,5),mydashdotted);
```

Lo spessore di un tracciato è impostabile con una chiamata alla funzione `linewidth(real w)`.

La dimensione del font è specificata in punti con la funzione `fontsize(real size, real lineskip=1.2*size)`, che ritorna un `pen`. La dimensione del font di default, 12pt, può essere modificata con la funzione `defaultpen(pen p)`. Per impostare una dimensione non standard dei caratteri è richiesto il comando `import fontsize`; all'inizio del file sorgente (questo richiede anche l'uso del pacchetto `LATEX fix-cm` che è supportato dalle recenti distribuzioni di `LATEX`). Si rimanda al manuale di *Asymptote* per un riferimento su tutte le funzioni di gestione dei caratteri e della composizione del testo.

Il linguaggio offre una comoda interfaccia con i font PostScript standard attraverso le funzioni `AvantGarde`, `Bookman`, `Courier`, `Helvetica`, `NewCenturySchoolBook`, `Palatino`, `TimesRoman`, `ZapfChancery`, `Symbol`, `ZapfDingbats`. Un esempio di composizione che fa uso del font Helvetica è riportato nella figura 15.

5 Integrazione di codice *Asymptote* in sorgenti `LATEX`

Per poter usare del codice *Asymptote* in un documento `LATEX` esiste l'ambiente `asy` messo a disposizione dal pacchetto di estensione `asymptote`. Quest'ultimo è implementato nel file `asymptote.sty` e viene distribuito ufficialmente insieme al resto del sistema grafico.

Per usufruire di questa funzionalità, tutto quello che si deve fare è includere il coman-

```

unitsize(1cm,1cm);
real i=0,j=0;
pen p1=Helvetica(series="m",shape="n");
pen p2=Helvetica(series="m",shape="it");
pen p3=Helvetica(series="b",shape="n");
// funzione di stampa
void testpolice(string phrase, pen police, real d=0) {
  label(phrase,(i,j),police);
  label("ABCDEFGHIIJK",(i+3,j-d),police);
  label("abcdefghijk",(i+3,j-.5-d),police);
  label("0123456789",(i+3,j-1-d),police);
  j-=2+d;
}
testpolice("Helvetica normal",p1,0.5); // stampa
testpolice("Helvetica Italic",p2,0.5);
testpolice("Helvetica bold",p3,0.5);

```

Helvetica normal

ABCDEFGHIIJK
 abcdefghijk
 0123456789

Helvetica Italic

ABCDEFGHIIJK
abcdefghijk
0123456789

Helvetica bold

ABCDEFGHIIJK
abcdefghijk
0123456789

FIGURA 15: Gestione del font con il tipo `pen`.

```

import labelpath;
usepackage("guit");
unitsize(1cm);
string t="\Large\GuITmeeting[style=inline,year=2009]";
path c=(0,0)..(2,2)..{dir(10)}(7,0.5);
labelpath(t,c);
draw(c, red);

```



FIGURA 16: Disegnare un testo lungo un tracciato.

do `\usepackage{asymptote}` nel preambolo del documento L^AT_EX. Un esempio minimale è il seguente:

```

% preambolo
\usepackage%
    [pdftex] se si usa pdflatex
    {graphicx}
\usepackage{asymptote}
% ...
% dopo \begin{document}
% ...
\begin{figure}
\centering
\begin{asy}
size(3cm);
draw(unitcircle);
\end{asy}
\caption{Sembra facile!}
\label{fig:asymptote:embedded}
\end{figure}

```

Si deve osservare che la dichiarazione di chiusura `\end{asy}` deve figurare su una linea a sé stante, senza spazi iniziali o finali.

Se il frammento di documento precedente appartiene ad un file di nome `doc.tex` la sequenza dei comandi che serve a creare il documento finale è la seguente:

```

latex doc
asy doc
latex doc

```

Se si preferisce usare PDFL^AT_EX basta specificare l'opzione `pdftex` per il pacchetto `graphicx` ed invocare il programma `pdflatex` al posto di `latex`.

Si rimanda alla guida TEIXEIRA (2007) per maggiori dettagli sull'uso di questo metodo di lavoro.

La figura 16 mostra un esempio dell'uso di `labelpath` per disegnare un testo lungo un tracciato.

6 Miscellanea

Il disegno nella figura 17 fa parte della galleria di immagini del sito ufficiale di *Asymptote* ed il sorgente è ottenibile all'indirizzo <http://asymptote.sourceforge.net/gallery/CAD1.asy>.

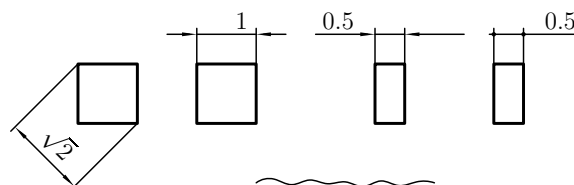


FIGURA 17: Esempio realizzato mediante il modulo di estensione denominato CAD.

Compilando il seguente programma si ottiene l'immagine riportata nella figura 18.

```

import graph3;
size(7.5cm,0);
texpreamble("\usepackage{guit}");
currentprojection=perspective
(100,100,150);

```

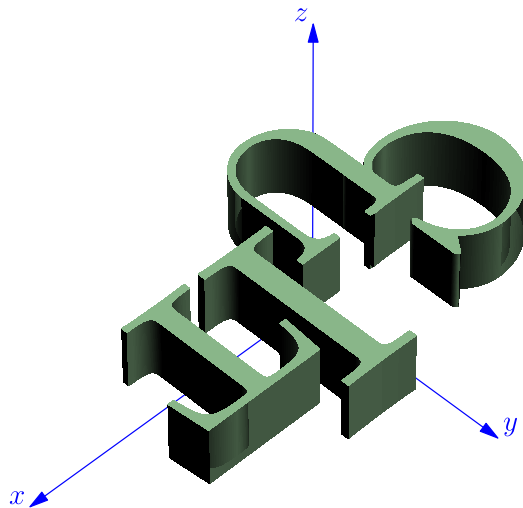


FIGURA 18: Esempio realizzato mediante il modulo di estensione denominato `graph3`.

```
draw(extrude("\GuIT{ }",3Z),palegreen);
limits(0,15X+10Y+15Z);
xaxis3(Label("$x$",1),blue,arrow=
Arrow3);
yaxis3(Label("$y$",1),blue,arrow=
Arrow3);
zaxis3(Label("$z$",1),blue,arrow=
Arrow3);
```

7 Conclusioni

Questo articolo si è proposto di offrire una graduale introduzione ad *Asymptote*. Il sistema grafico è stato presentato con una sequenza di esempi, spesso semplificati, che si sono focalizzati sui diversi aspetti del linguaggio di programmazione. Si sono evidenziate le potenzialità e le possibilità grafiche offerte da questo software libero.

Per motivi di brevità sono diversi gli argomenti che si sono lasciati alla cura del lettore. Si pensi al riempimento di figure chiuse con *pattern* e tratteggi vari, alla gestione dei colori e degli algoritmi di *shading*, alle librerie per il disegno di grafici scientifici, alle potenti funzioni di gestione di scene tridimensionali. Tutto questo, ed altro ancora, può essere approfondito nei manuali citati ed è dimostrato efficacemente nelle gallerie di esempi disponibili in rete.

Riferimenti bibliografici

- BECCARI, C. (2007). «Graphics in L^AT_EX». *PracT_EX Journal*, (1). URL <http://www.tug.org/pracjourn/2007-1/beccari/>.
- BOWMAN, J. (2009). «Webpage of Asymptote: The vector graphics language». URL <http://asymptote.sourceforge.net/>.
- BOWMAN, J. C. e HAMMERLINDL, A. (2008). «Asymptote: A vector graphics language». *TUG-*

BOAT: The Communications of the T_EX Users Group, **29** (2), pp. 288–294. URL <http://www.tug.org/TUGboat/Contents/contents29-2.html>.

BOWMAN, J. C. e SHARDT, O. (2009). «Asymptote: Lifting T_EX to three dimensions». *TUGBOAT: The Communications of the T_EX Users Group*, **30** (1), pp. 58–63. URL <http://www.tug.org/TUGboat/Contents/contents30-1.html>.

CASCHILI, M. (2006). «Semplici figure con l'ambiente *picture*». *ArsT_EXnica*, (1), pp. 20–28. URL <http://www.guit.sssup.it/arstexnica.php>.

— (2007). «Introduzione a PSTricks». *ArsT_EXnica*, (4), pp. 25–44. URL <http://www.guit.sssup.it/arstexnica.php>.

DE MARCO, A. (2007). «Illustrazioni tridimensionali con Sketch/L^AT_EX/PSTricks/TikZ nella didattica della Dinamica del Volo». *ArsT_EXnica*, (4), pp. 51–68. URL <http://www.guit.sssup.it/arstexnica.php>.

— (2008). «Gestione avanzata delle figure in L^AT_EX». *ArsT_EXnica*, (6), pp. 10–27. URL <http://www.guit.sssup.it/arstexnica.php>.

HAMMERLINDL, A. e BOWMAN, J. (2007). «Asymptote: The vector graphics language». URL <http://asymptote.sourceforge.net/intro.pdf>.

HAMMERLINDL, A., BOWMAN, J. e PRINCE, T. (2009). «Asymptote: The vector graphics language». URL <http://asymptote.sourceforge.net/asymptote.pdf>.

HOBBY, J. D. (2009). *METAPOST A User Manual*. URL <http://www.tug.org/tutorials/mp/mpman.pdf>.

STROUSTRUP, B. (2000). *C++ Linguaggio, libreria standard, principi di programmazione*. Addison-Wesley Publishing, Pearson Education Italia (collana Professionale).

TEIXEIRA, D. (2007). «Asymptote and L^AT_EX: An Integration Guide». URL <http://www.dse.nl/~dario/projects/asylatex/asylatex.pdf>.

▷ Agostino De Marco
Università degli Studi di Napoli
“Federico II”
Dipartimento di Ingegneria
Aerospaziale
agostino dot demarco at unina
dot it