# Produrre grafica vettoriale di alta qualità programmando *Asymptote*

**Agostino De Marco**

Università degli Studi di Napoli "Federico II"
Dipartimento di Ingegneria Aerospaziale (DIAS)
Via Claudio, 21 – 80125 Napoli, Italy

`agostino.demarco@unina.it`

GUIT *2009 meeting*

**17 ottobre 2009, Pisa, Italy.**

# Indice

# Indice

# Produrre grafici e disegni in LaTeX

- Gli utenti di LaTeX hanno spesso bisogno d'inserire nei loro documenti disegni, schemi e figure di varia complessità.
- LaTeX dispone di pacchetti di estensione per la produzione di materiale grafico (picture, pstricks, pgf). I disegni possono essere creati direttamente nel sorgente LaTeX di origine (senza l'uso di programmi esterni o limitandosi, al più, a qualche conversione di formato).
- Esistono sistemi di disegno esterni detti 'LaTeX-*aware*': capaci di comporre le annotazioni testuali all'interno delle figure utilizzando una distribuzione pre-installata di LaTeX.

## Sistemi grafici esterni LaTeX-aware

Alcuni sistemi di disegno LaTeX-*aware*:

- *Sketch*, programmabile attraverso un linguaggio grafico descrittivo; produce in uscita dei frammenti di codice pstricks o pgf — sistema orientato alla grafica 3D;
- *Ipe*, applicazione interattiva (GUI), produce output direttamente in PDF o EPS;
- *Inkscape*, con il plug-in *textext* (Python) gli oggetti testuali sono composti con LaTeX e trasformati in tracciati;
- *Asymptote*, applicazione *command-driven*, programmabile attraverso un linguaggio simile al C++, produce output direttamente in PDF o EPS.

# Il programma di grafica vettoriale *Asymptote*

- Disponibile per Linux, Windows e Mac (licenza GNU). Sviluppatori: John Bowman, Andy Hammerlindl e Tom Prince.
- È un interprete di comandi che dispone di un linguaggio di programmazione di alto livello orientato alla matematica.
- Potente strumento di lavoro, adatto alla produzione di disegni tecnici e di immagini a carattere matematico.
- Ampia documentazione e numerosi esempi d'uso disponibili in rete.

# Indice

# Come funziona *Asymptote*?

- Lo strumento di lavoro si presenta all'utilizzatore come un'applicazione *command-driven*: all'utente è richiesto di raccogliere una sequenza di comandi in un file sorgente di estensione `.asy`, da *compilare* con il programma `asy` per ottenere un'immagine vettoriale in formato PostScript o PDF.
- Esempio: (sorgente) `test.asy` $\longrightarrow$ `test.pdf` (disegno)

  `asy -V -tex pdflatex test` *(batch mode)*
- Esempio di file ⟨*home*⟩`/.asy/config.asy`

```
import settings;
tex="pdflatex";
batchView=false;
```

# Come funziona *Asymptote*? (II)

- Con l'opzione `-outformat` ⟨*formato*⟩ si può ottenere un'uscita grafica in diversi formati (tramite la libreria *ImageMagick*).

  Esempio: `test.asy` ⟶ `test.png`

  ```
  asy -outformat png test
  ```

- 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.

- `xasy`, un'applicazione dotata di interfaccia grafica ed utile ad apportare semplici modifiche a disegni precedentemente generati tramite uno *script*. Destinata a diventare una vera e propria applicazione visuale interattiva.

- Estensione `asymptote.sty` e ambiente `asy`, per inglobare codice *Asymptote* nei sorgenti LaTeX.

# Indice

## Caratteristiche di *Asymptote*

- Gli autori hanno concepito *Asymptote* come candidato a strumento standard per la creazione di figure matematiche e disegni tecnici (così come LaTeX è lo standard per i documenti matematici).
- Attualmente *Asymptote* è considerato il moderno successore di METAPOST.
- *Asymptote* è un vero e proprio linguaggio di programmazione di alto livello, ispirato a METAPOST, ma con una sintassi molto più pulita e potente, quasi del tutto simile a quella del C++.
- *Asymptote* è anche un ambiente matematico di disegno, basato sull'uso naturale di sistemi di coordinate.
- Estendibile con *moduli*, raccolte di funzioni definite dall'utente. Esempio: il modulo `geometry.asy` (P. Ivaldi).

# Tipi di dati predefiniti

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

## Supporto alla programmazione procedurale e strutturata

- Il linguaggio possiede tutti gli elementi che supportano la programmazione strutturata: costrutti di controllo e di iterazione, supporto alle operazioni di I/O, eccetera.
- Semplice esempio di funzione:

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

# Supporto alla programmazione a oggetti (astrazione dati)

```
// 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 */
}
```

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

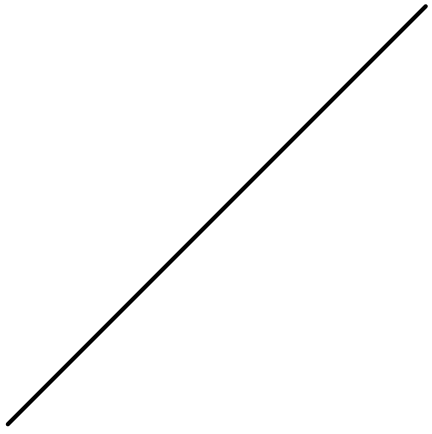
```
// CS, rif. definito altrove
point pt=point(CS,(2,5));
```

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

# Indice

# Hello world!

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



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

Hello world!

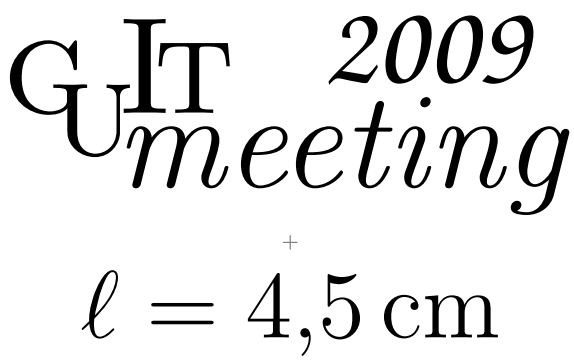
```
label("Hello world!",(0,0),N);
label(scale(0.5)*"Hello world!",(0,0),S);
```

Hello world!

Hello world!

# Personalizzare la composizione testuale

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



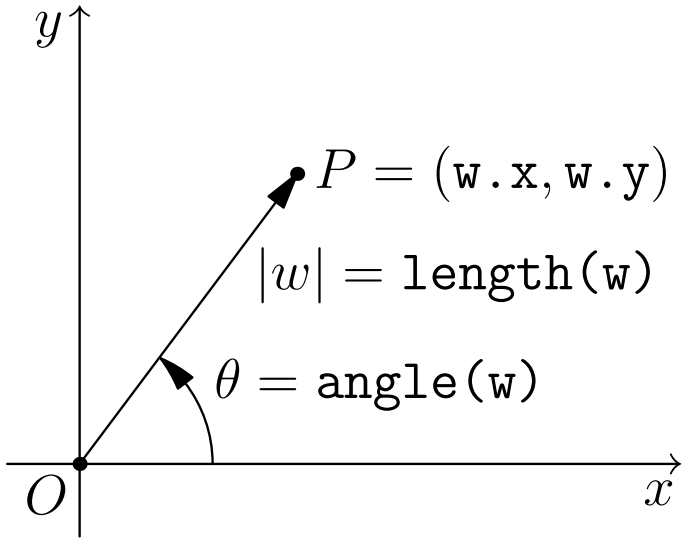

```
texpreamble("\input{my_preamble}");
```

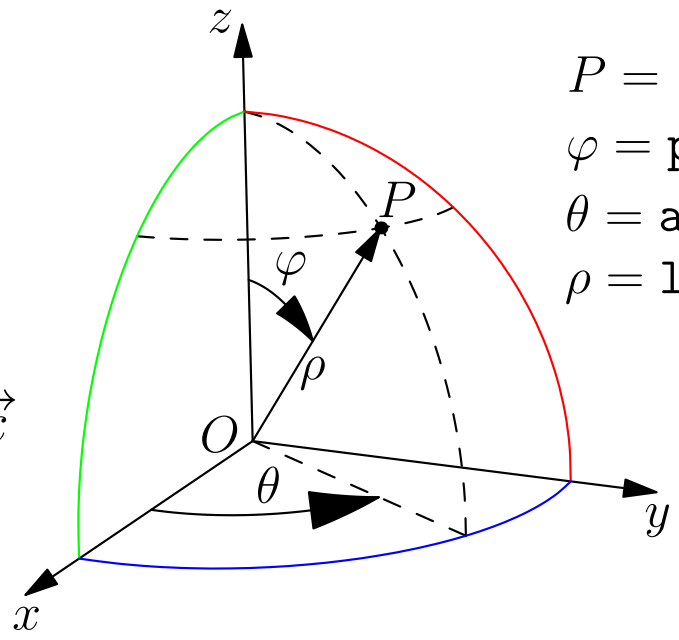
## Sistemi di coordinate

- I comandi di disegno si basano sulla definizione di un opportuno sistema di coordinate. Ogni punto della "tela" è inteso come una coppia $(x, y)$ di coordinate.
- L'unità di lunghezza di default è il *PostScript bigpoint* (`bp`, pari a 1/72 di pollice). I punti `(0,0)` e `(72,0)` distano esattamente un pollice.
- Funzione `unitsize`: cambia l'unità di misura corrente.
- Funzione `size`: scala il disegno.
- Esempio:

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

# Rappresentazione interna di `pair` e `triple`



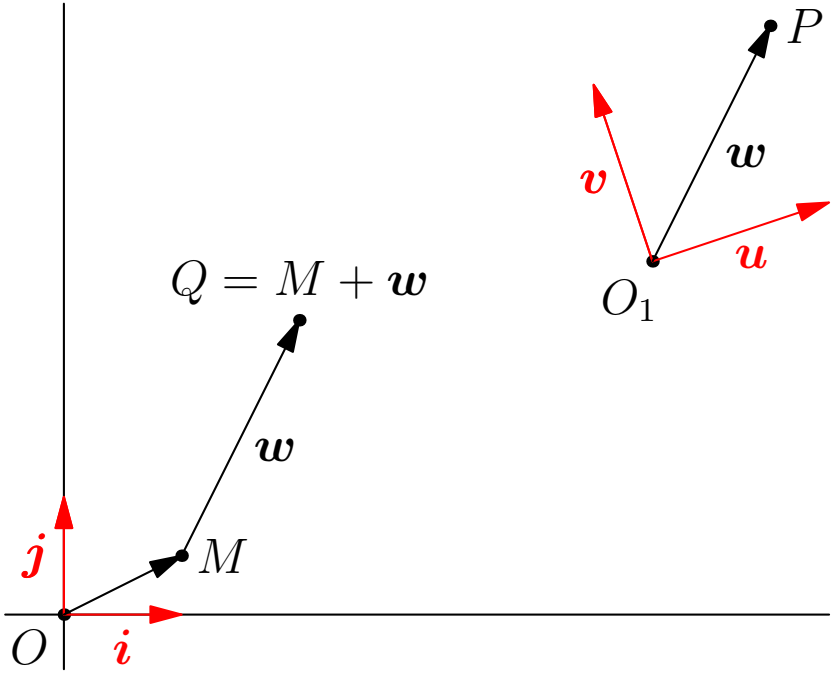

$P = (\texttt{w.x}, \texttt{w.y}, \texttt{w.z})$
$\varphi = \texttt{polar(w)}$
$\theta = \texttt{azimuth(w)}$
$\rho = \texttt{length(w)}$

```
pair w;
```

```
triple w;
```

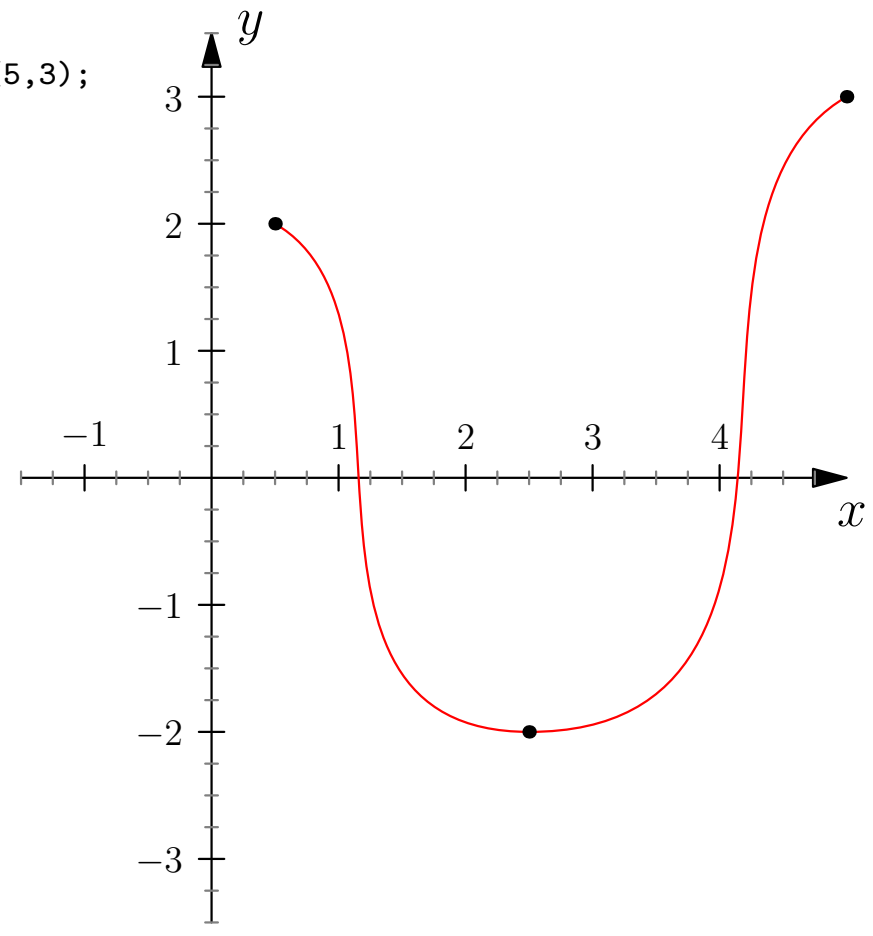
# Sistemi di coordinate (II)

```
import geometry;
size(7cm,0);
usepackage("amsmath");
show("$O$",
     "$\boldsymbol{i}$",
     "$\boldsymbol{j}$",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("$O_1$",
     "$\boldsymbol{u}$",
     "$\boldsymbol{v}$", 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);
```



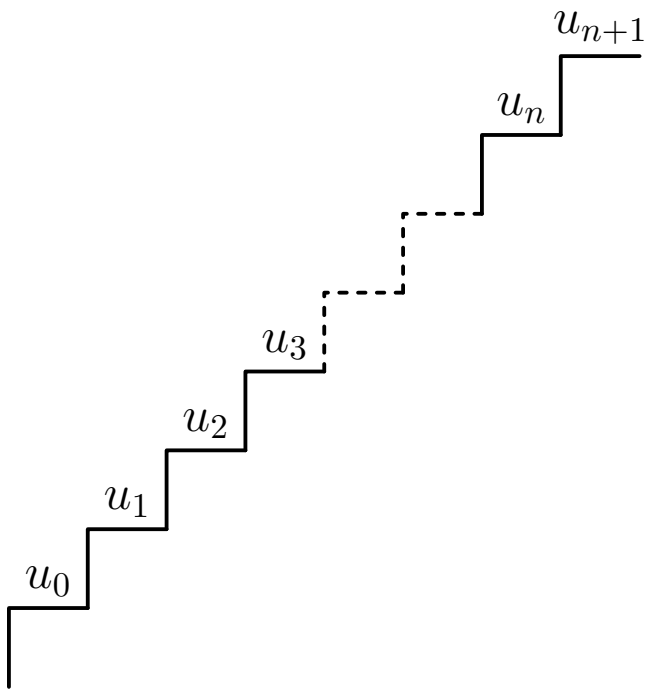
# Grafici

```
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);
```



# Costrutti

```
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);
}
```




- *Asymptote* mette a disposizione tutti i costrutti più comuni (cicli, scelte, ecc.) proprio come in C++.
- Come in Java, il linguaggio permette di scrivere elegantemente dei cicli su tutti gli elementi di un *array*.

```
int[] myarray={1,1,2,3,5};
for(int k : myarray) {
    write(k);
}
```

# Funzioni di disegno

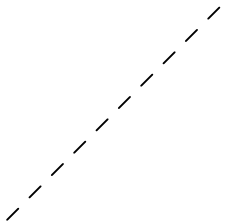
- Tutte le capacità grafiche *Asymptote* si basano su quattro funzioni primitive (comandi). I tre comandi PostScript `draw`, `fill`, `clip`, ed il comando `label`.
- I comandi di disegno accettano molte opzioni (argomenti). Esse hanno tutte delle assegnazioni di default ragionevoli e richiedono un controllo esplicito solo in caso di particolari esigenze.

Esempi:

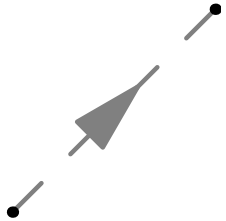
```
pair A=(0,0), B=(50,50);
path p=A--B;
draw(p,dashed);
```

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

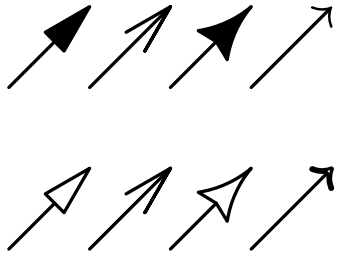
# Esempi: tracciati e pennini



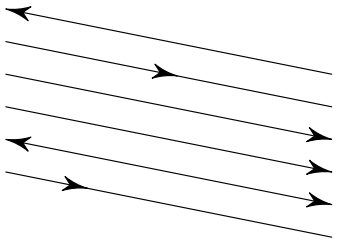
(a) Segmento semplicemente tratteggiato.



(b) Segmento grigio, orientato e con punti estremi evidenziati.

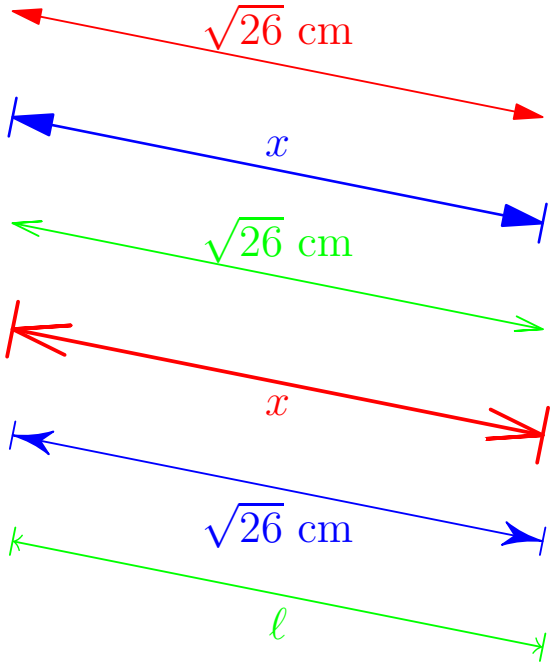


(a) Tipi di freccia.



(b) Posizionamenti di una freccia lungo un segmento.

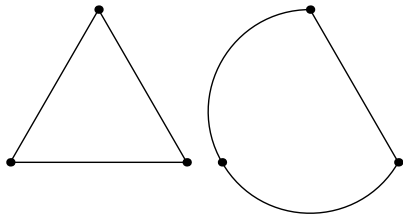
```
pair A=(0,0), B=(50,50);
path p=A--B;
draw(p,dashed);

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


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



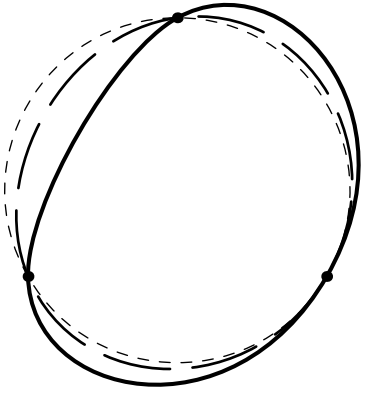

(c) Dimensioni di un segmento.

# Esempi di tracciati

```
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);
```



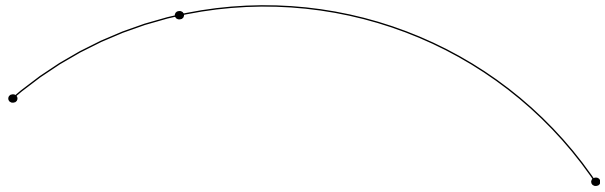
(a) Due tracciati chiusi definiti con gli stessi nodi.



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

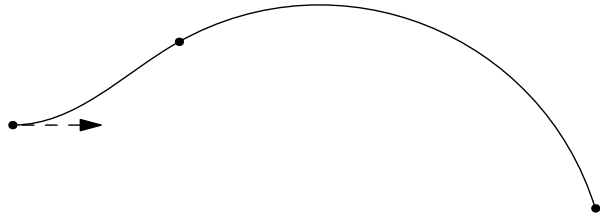
```
t1=p1..p2..p3..cycle;
t2=p1..tension 1.1 ..p2..p3..cycle;
t3=p1..tension 1.5 ..p2..p3..cycle;
draw(t1,dashed); dot(t1,black+4bp);
draw(t2,linetype("24 8")+1bp);
draw(t3,black+1.5bp);
```

```
path curva=(-3,1)..(-1,2)..(4,0);
```



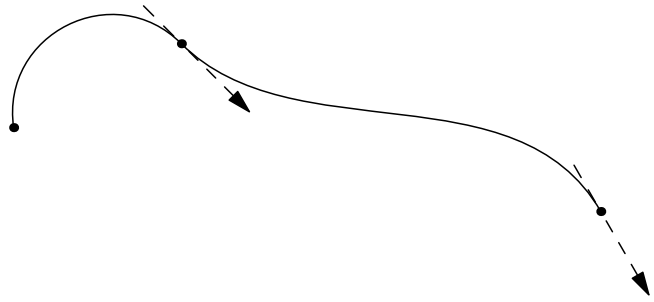
(a) Curva passante per i tre punti assegnati con tensione di default.

```
path curva=(-3,1){right}..(-1,2)..(4,0);
```



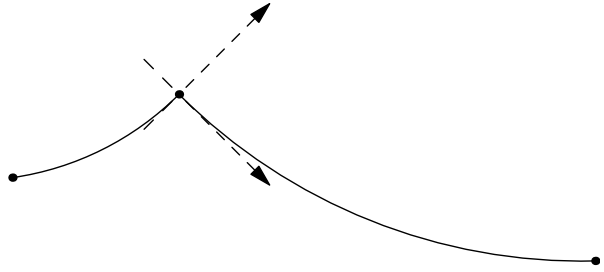
(b) Alterazione della direzione della tangente nel primo nodo.

```
path curva=(-3,1)..{dir(-45)}(-1,2)..{dir(-60)}(4,0);
```



(c) Alterazione della direzione della tangente locale nel secondo e terzo nodo.

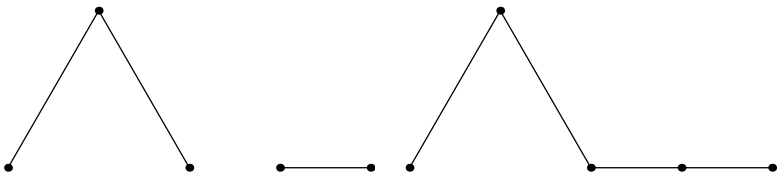
```
path curva=(-3,1)..{dir(-45)}(-1,2){dir(45)}..(4,0);
```



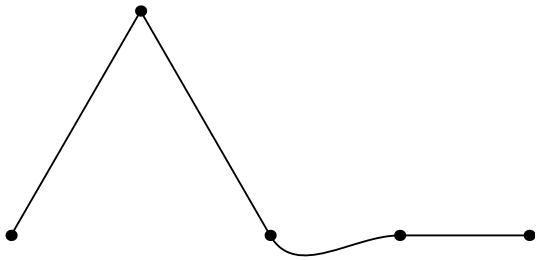
(d) Alterazione della direzione della tangente destra e sinistra nel secondo nodo.

# Concatenazione di tracciati

```
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);
```



(a) Due spezzate separate. (b) Unione dei due tracciati con una linea retta.

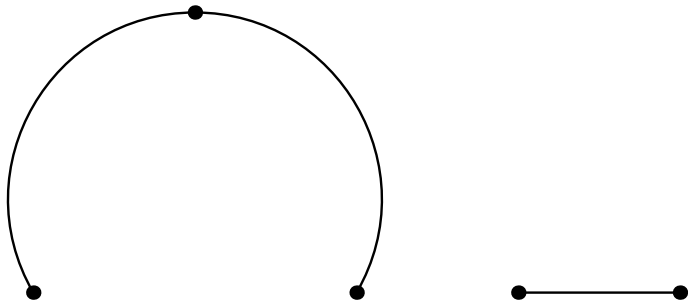


(c) Unione dei due tracciati con una curva di Bezier che li raccorda dolcemente.

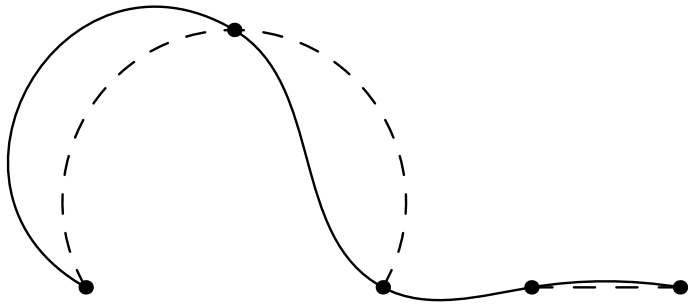
```
path t3=t1--t2;
draw(t3); dot(t3);
```

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

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



(a) Due curve di Bezier separate.

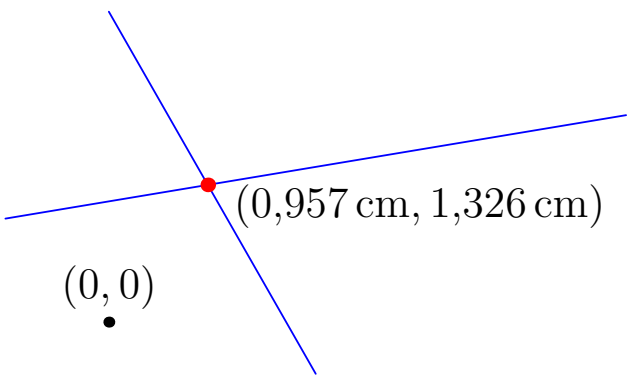


(b) Unione dei due tracciati con aggiustamento e riparametrizzazione della rappresentazione interna (raccordo morbido).

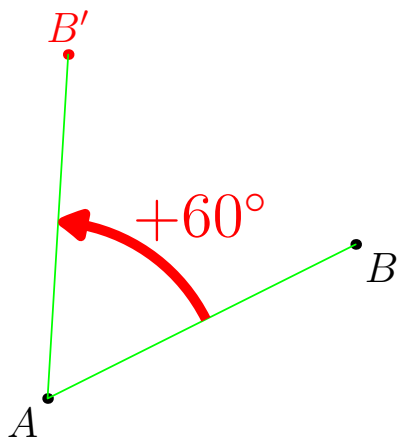
```
path t3=t1..t2; // cast da guide a path
draw(t3); dot(t3);
```

# Trasformazioni

```
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);
```

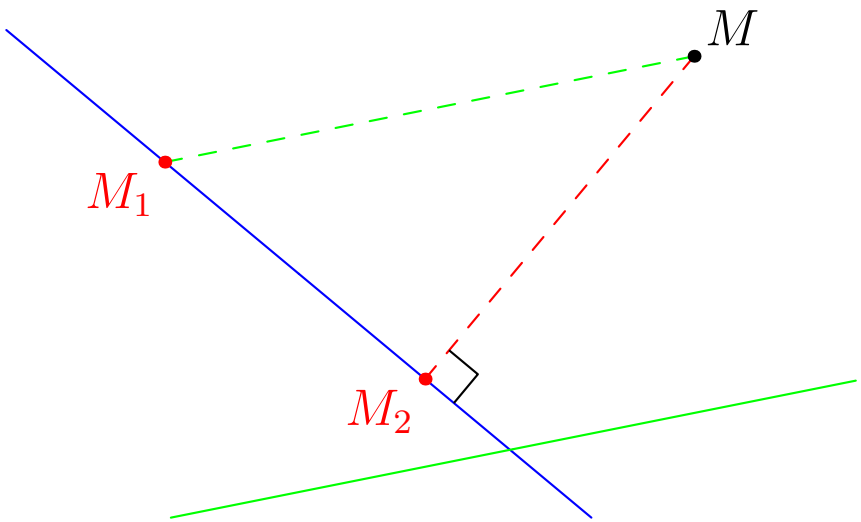



```
size(4cm,0);
import markers;
// Due punti, A e B
pair pA=(0,0),pB=(4,2);
dot("$A$",pA,SW); dot("$B$",pB,SE);
// B1 per rotazione di B intorno ad A
// di un angolo di 60 gradi
pair pB1=rotate(60,pA)*pB;
// disegno
dot("$B'$",B1,N,red);
draw(pB--pA--pB1,green);
// marcatura dell'angolo
markangle(scale(1.5)*"$+60^\circ$",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



# Proiezioni

```
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);
```



# Accesso ai font PostScript standard

```
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("ABCDEFGHIJK",(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

ABCDEFGHIJK
abcdefghijk
0123456789

*Helvetica Italic*

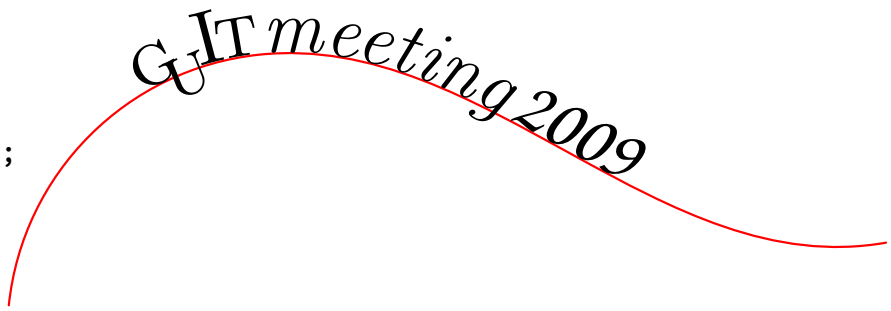
*ABCDEFGHIJK*
*abcdefghijk*
*0123456789*

**Helvetica bold**

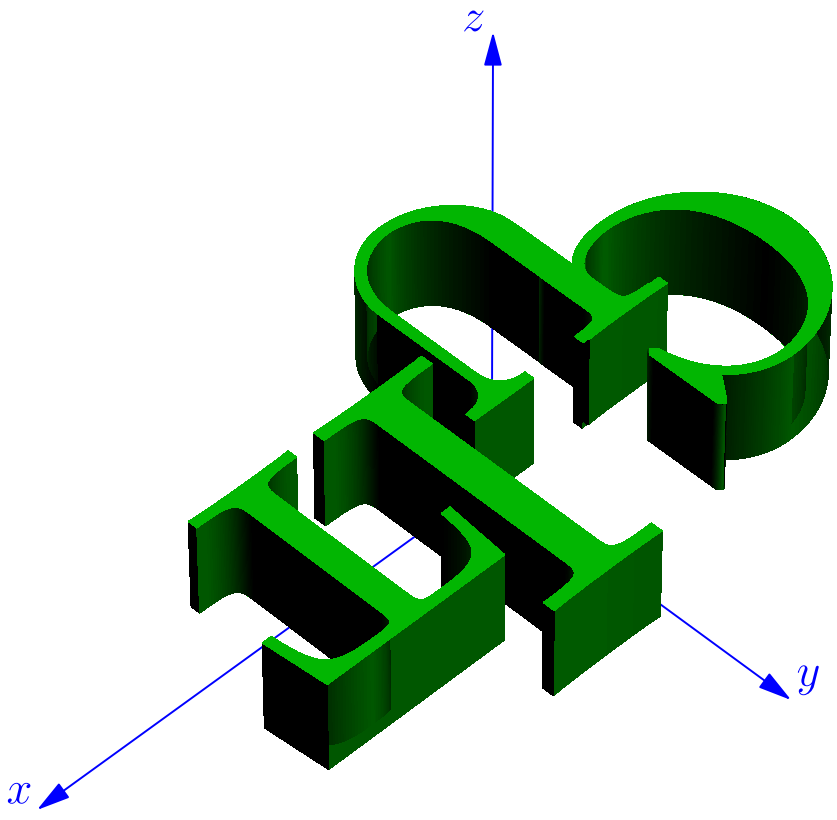
**ABCDEFGHIJK**
**abcdefghijk**
**0123456789**

# Effetti speciali sul testo

```
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);
```



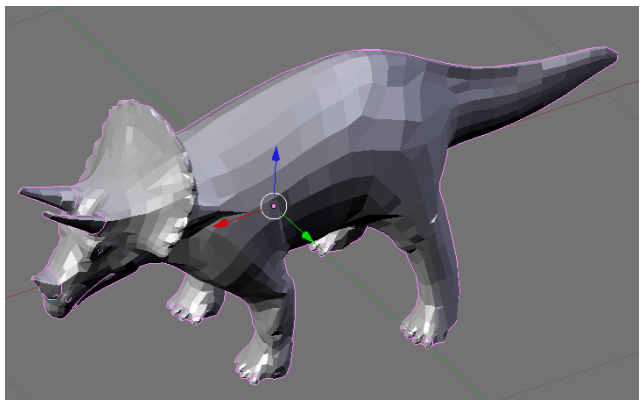

```
import settings;
import graph3;
settings.render=0;
size(7.5cm,0);
usepackage("guit");
currentprojection=perspective
    (100,100,150);
draw(extrude("\GuIT{}",3Z),green);
limits(0,15X+10Y+15Z);
xaxis3(Label("$x$",1),blue,arrow=Arrow3);
yaxis3(Label("$y$",1),blue,arrow=Arrow3);
zaxis3(Label("$z$",1),blue,arrow=Arrow3);
```

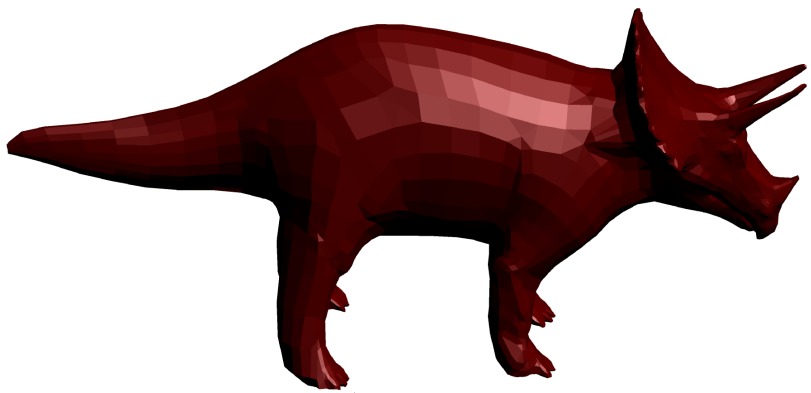


# Indice

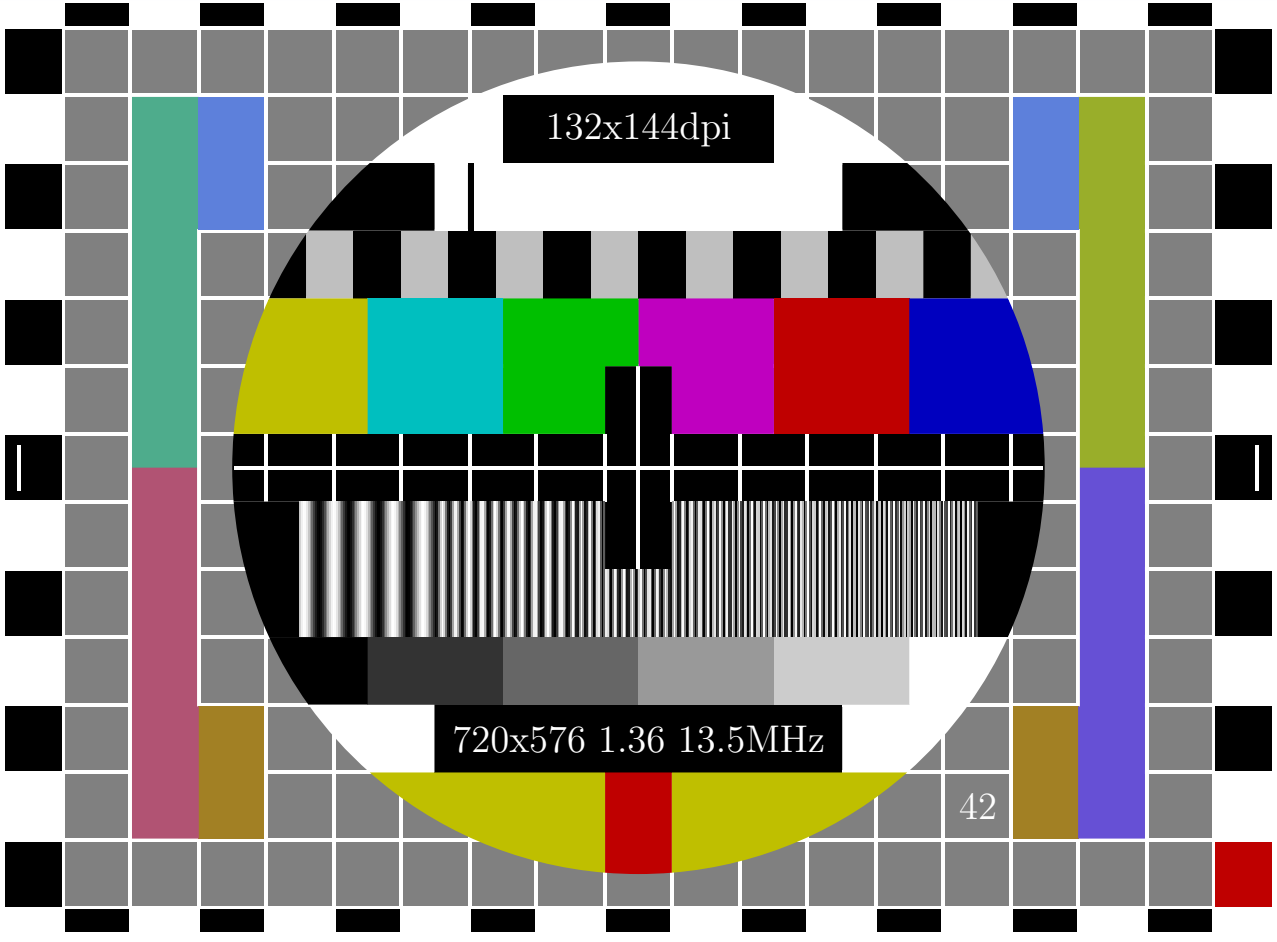
# Contenuti 3D in file PDF

- Le scene 3D create con *Asymptote* sono di default in formato PRC.
- PRC è una specifica di formato introdotta da Adobe per inserire oggetti 3D in un file PDF. Ha diversi livelli di compressione. ($Acrobat \geq 9$)





`asy -render=0 test3d` (*crea un PDF statico*)




`http://asymptote.sourceforge.net/gallery/tvgen.asy`

# Grazie dell'attenzione