METAPOST: un outil de dessin PostScript pour *TEX

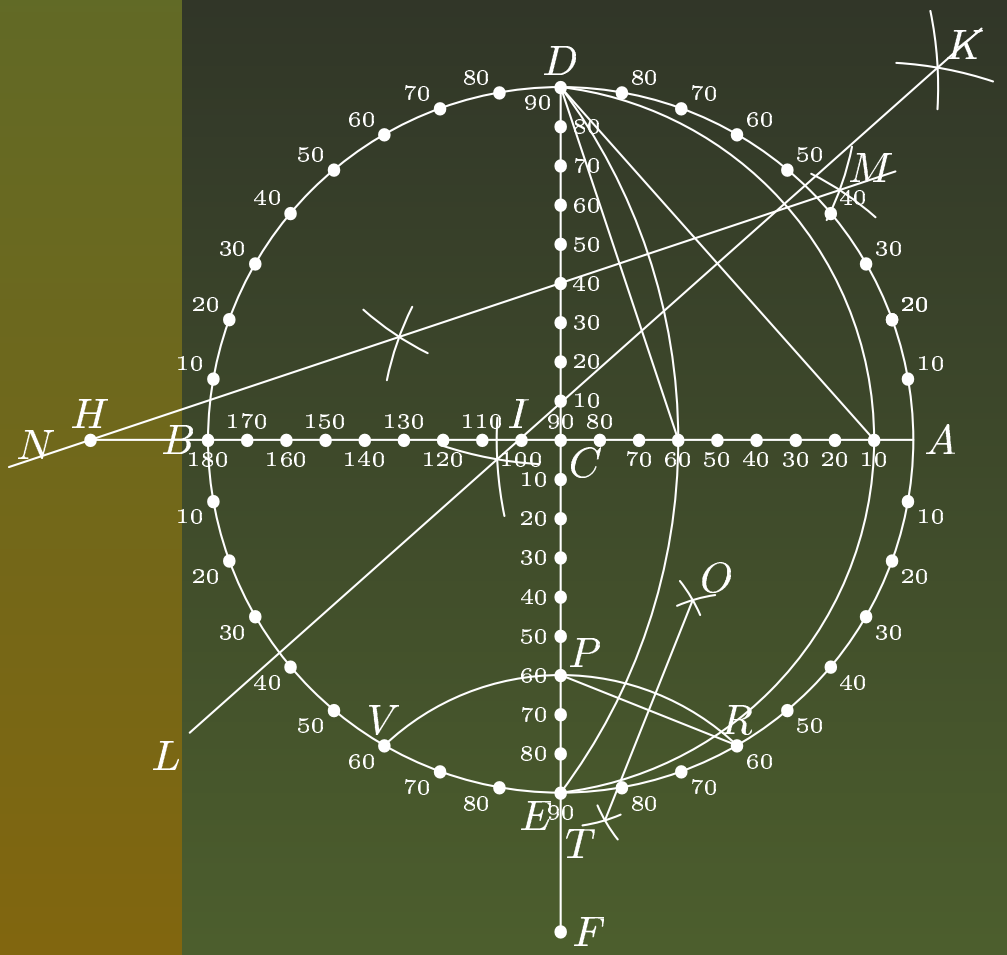
Un langage de programmation

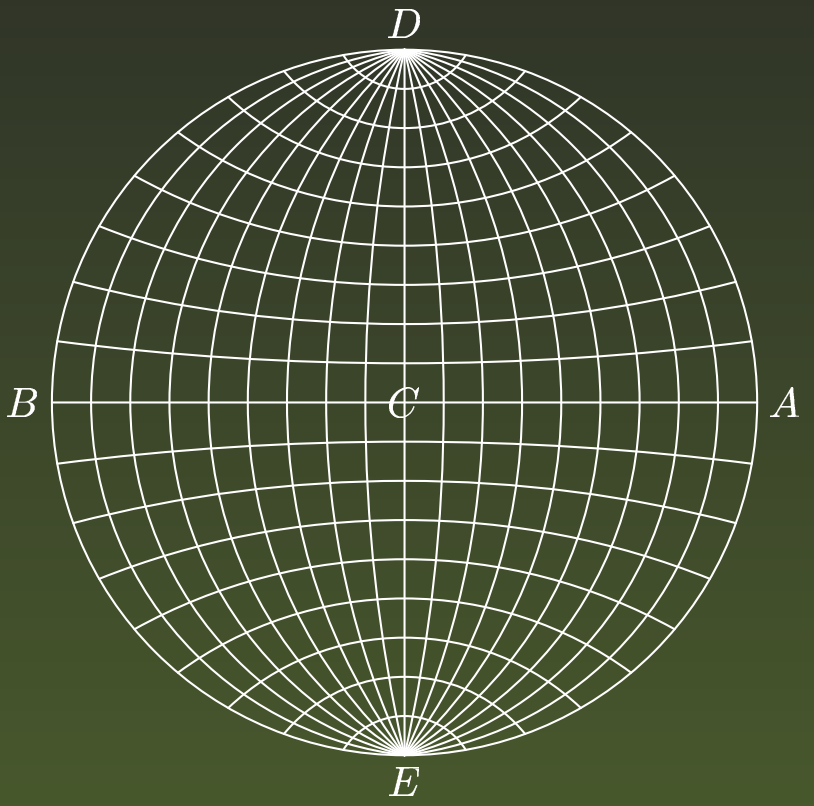
Pierre FOURNIER

pierre.fournier@unilim.fr

IUT du Limousin, Limoges

Un exemple pour débuter





Un fichier METAPOST(toto.mp)

Structure de toto.mp :

- Un préambule ;
- beginfig(1) ;

  ... instructions, terminées par ;

  endfig ;
- ...
- beginfig(n) ;

  ...instructions, terminées par ;

  endfig ;
- end

Un fichier METAPOST(toto.mp)

Structure de toto.mp :

- Un préambule ;
- beginfig(1) ;
  ... instructions, terminées par ;
  endfig ;
- ...
- beginfig(n) ;
  ...instructions, terminées par ;
  endfig ;
- end

```
mp toto.mp
```

⇓

Un fichier METAPOST(toto.mp)

Structure de toto.mp :

- Un préambule ;
- beginfig(1) ;
  ... instructions, terminées par ;
  endfig ;
- ...
- beginfig(n) ;
  ...instructions, terminées par ;
  endfig ;
- end

`mp toto.mp`

⇓

... les fichiers images :

- toto.1
- ...
- toto.n

Un fichier METAPOST(toto.mp)

Structure de toto.mp :

- Un préambule ;
- beginfig(1) ;
  ... *instructions, terminées par ;*
  endfig ;
- ...
- beginfig(n) ;
  ...*instructions, terminées par ;*
  endfig ;
- end

`mp toto.mp`

⇓

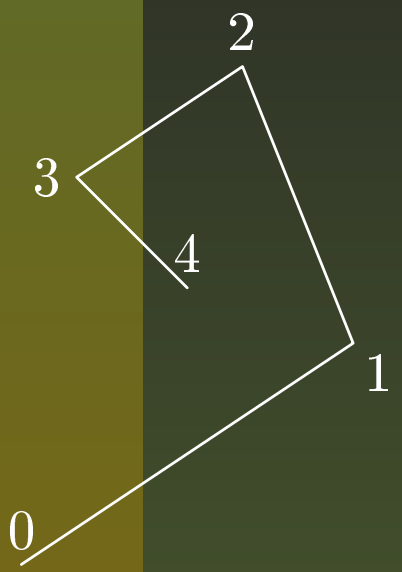
... les fichiers images :

- toto.1
- ...
- toto.n

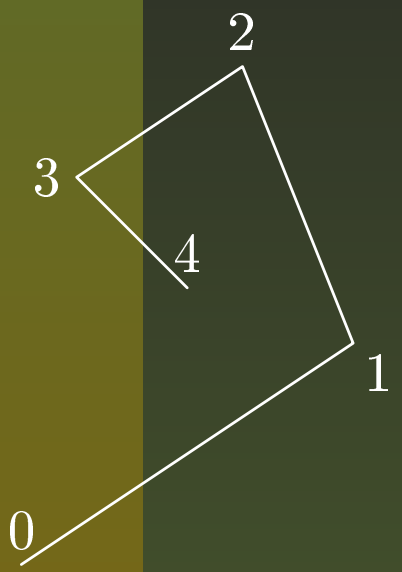
à insérer dans un fichier LaTeX avec \includegraphics{toto.i}, i variant de 1 à n. Les renommer avec l'extension « mps » pour les utiliser avec PDFLaTeX.

Des lignes droites

Des lignes droites

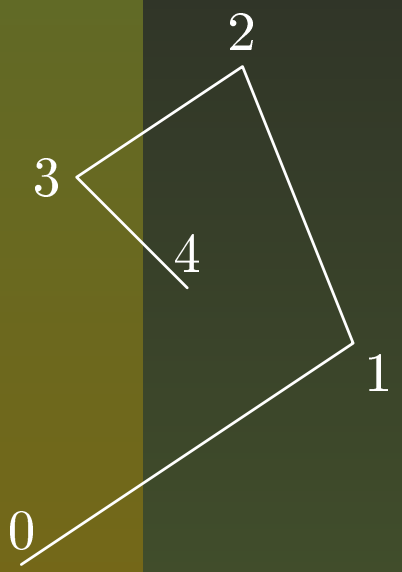


Des lignes droites




```
beginfig(0) ;
draw (0, 0)--(60, 40)--(40, 90)--(10, 70)--(30, 50)
withcolor white ;
endfig ;
```

Des lignes droites

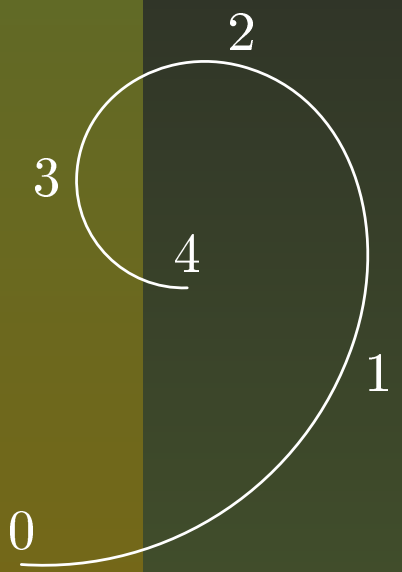



```
beginfig(0) ;
draw (0, 0)--(60, 40)--(40, 90)--(10, 70)--(30, 50)
withcolor white ;
endfig ;
```

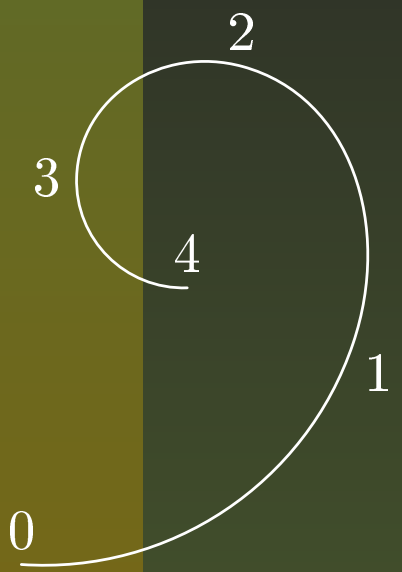
Les coordonnées des points sont exprimées, par défaut, en points PostScript (ou *bp*, 1 inch = 72 bp). Il est possible de préciser l'unité de mesure avec *in* pour inch, *cm*, *mm*, *pt* pour le point d'impression ($\neq$ bp) ou toute autre définie par l'utilisateur.

Des lignes courbes

Des lignes courbes

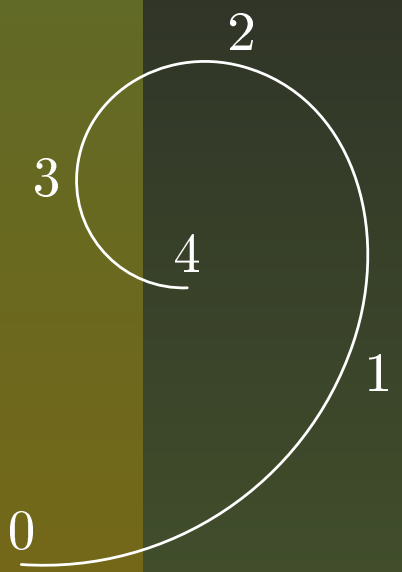


Des lignes courbes




```
beginfig(1) ;
draw (0, 0) .. (60, 40) .. (40, 90) .. (10, 70).. (30, 50)
withcolor white ;
endfig ;
```

Des lignes courbes

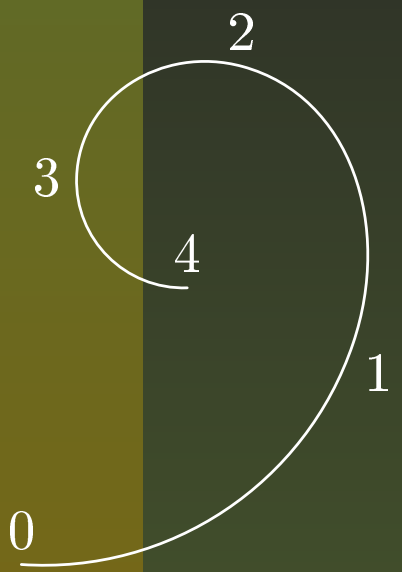



```
beginfig(1) ;
draw (0, 0) .. (60, 40) .. (40, 90) .. (10, 70).. (30, 50)
withcolor white ;
endfig ;
```

`--` a été remplacé par `..` pour obtenir des lignes courbes.

Les `--` et `..` peuvent cohabiter dans une commande `draw`.

Des lignes courbes




```
beginfig(1) ;
draw (0, 0) .. (60, 40) .. (40, 90) .. (10, 70).. (30, 50)
withcolor white ;
endfig ;
```

`--` a été remplacé par `..` pour obtenir des lignes courbes.

Les `--` et `..` peuvent cohabiter dans une commande `draw`.

On fermera un polygone ou une courbe en terminant l'instruction avec `cycle`.

Les courbes de BÉZIER

Les courbes de BÉZIER

L’instruction z0 .. z1 produit courbe de BÉZIER passant les points de coordonnées (x0, y0) et (x1, y1) et définie par

Les courbes de BÉZIER

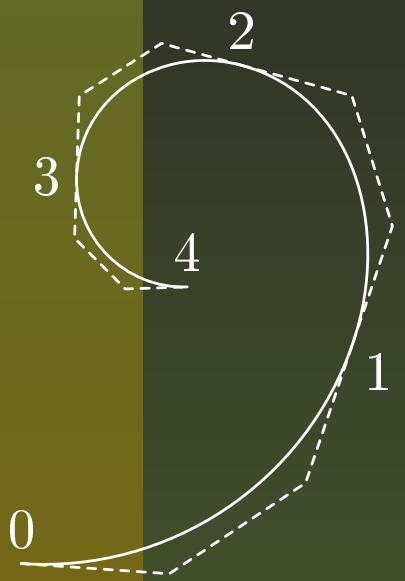
L'instruction z0 .. z1 produit courbe de BÉZIER passant les points de coordonnées (x0, y0) et (x1, y1) et définie par

$$X(t) = (1-t)^3 x_0 + 3t(1-t)^2 x_0^+ + 3t^2(1-t)x_1^- + t^3 x_1$$
$$Y(t) = (1-t)^3 y_0 + 3t(1-t)^2 y_0^+ + 3t^2(1-t)y_1^- + t^3 y_1$$

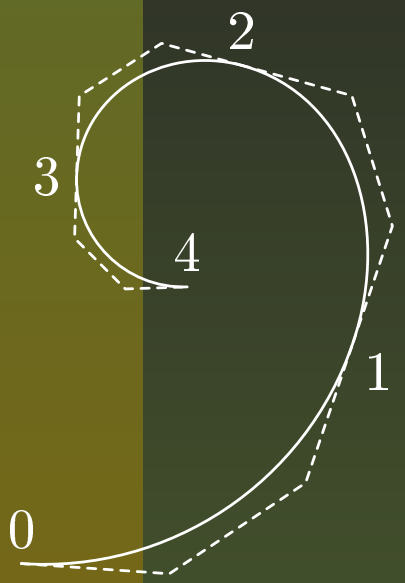
pour $t \in [0,\, 1]$

où $(x_0^+,\, y_0^+)$ et $(x_1^-,\, y_1^-)$ sont deux points de contrôle sélectionnés par METAPOST.

Courbes de Bézier (suite)

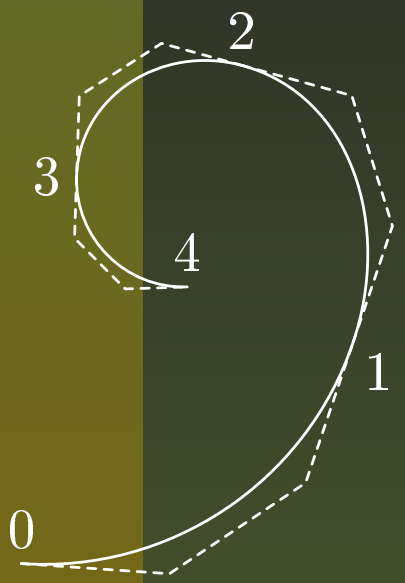


Courbes de Bézier (suite)



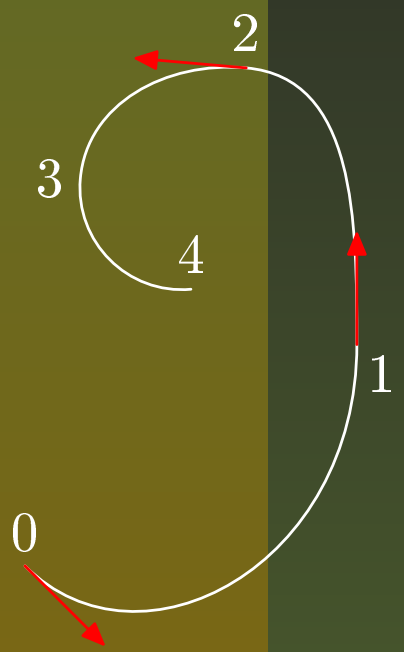
draw z0 .. z1 .. z2 .. z3 .. z4 ;

Courbes de BÉZIER (suite)

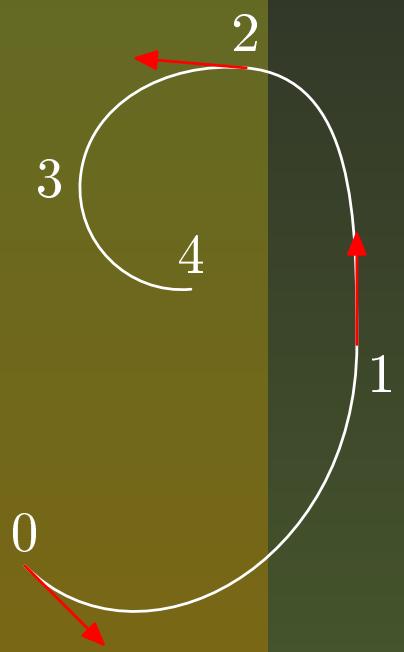



```
draw z0 .. z1 .. z2 .. z3 .. z4 ;
draw z0 .. controls(26.8, -1.8) and (51.4,14.6)
.. z1 .. controls(67.1, 61.0) and (59.8, 84.6)
.. z2 .. controls(25.4, 94.0) and (10.5, 84.5)
.. z3 .. controls(9.6, 58.8) and (18.8, 49.6)
.. z4 withcolor white dashed evenly ;
```

Directions

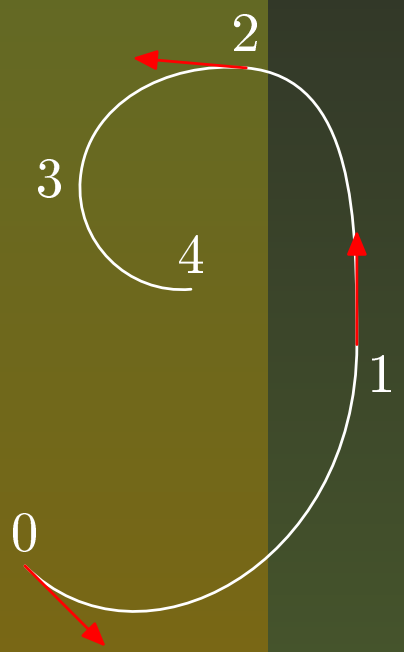


Directions




```
draw z0{1,-1} .. z1{up} .. z2{dir 175} .. z3 .. z4;
```

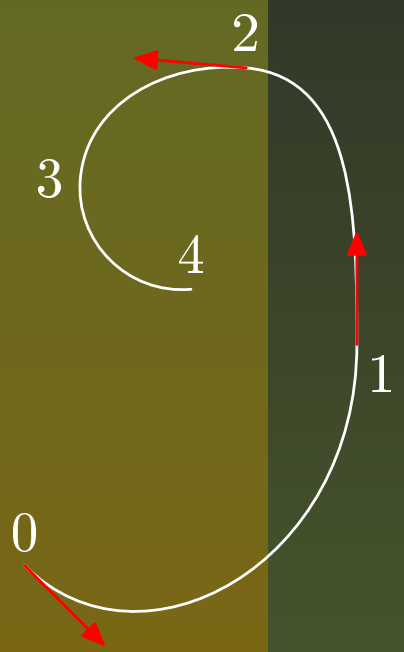
Directions



draw z0{1,-1} .. z1{up} .. z2{dir 175} .. z3 .. z4 ;

Les directions définies par :

Directions

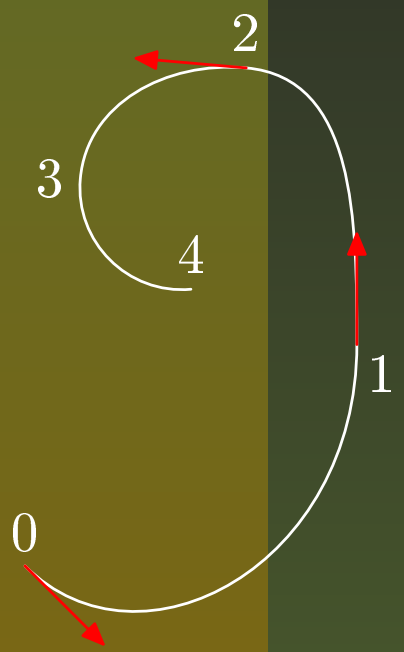


draw z0{1,-1} .. z1{up} .. z2{dir 175} .. z3 .. z4 ;

Les directions définies par :

- le vecteur tangente $\{a, b\}$;

Directions

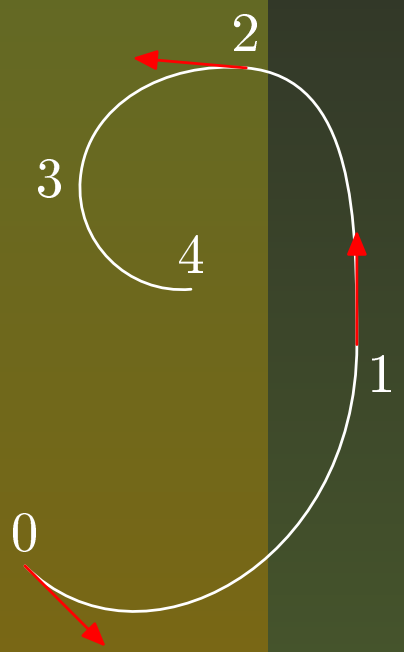


draw z0{1,-1} .. z1{up} .. z2{dir 175} .. z3 .. z4 ;

Les directions définies par :

- le vecteur tangente $\{a, b\}$;
- une direction par {*right*} (0 degré), {*up*} (90 degrés), {*left*} (180 degrés), {*down*} (270 degrés) ;

Directions



draw z0{1,-1} .. z1{up} .. z2{dir 175} .. z3 .. z4 ;

Les directions définies par :

- le vecteur tangente $\{a, b\}$;
- une direction par {*right*} (0 degré), {*up*} (90 degrés), {*left*} (180 degrés), {*down*} (270 degrés) ;
- {*dir* x} (x étant un angle en *degrés*).

Équations linéaires...

METAPOST est capable de lire

`a + b = 3;` et `2 * a = b + 3;`

Équations linéaires...

METAPOST est capable de lire

`a + b = 3;` et `2 * a = b + 3;`

À la commande `show a, b;`, METAPOST affichera

```
» 2
» 1
```

Équations linéaires...

METAPOST est capable de lire

```
a + b = 3;
```
et
```
2 * a = b + 3;
```

À la commande `show a, b;`, METAPOST affichera

```
» 2
» 1
```

Même résultat avec

```
a + b = 2 * a - b = 3;
```

Équations linéaires...

METAPOST est capable de lire

```
a + b = 3;
```
et
```
2 * a = b + 3;
```

À la commande `show a, b;`, METAPOST affichera

```
» 2
» 1
```

Même résultat avec

```
a + b = 2 * a - b = 3;
```

« $=$ » $\Rightarrow$ équations linéaires

« $:=$ » $\Rightarrow$ assignation

Équations linéaires... pour des points

Équations linéaires... pour des points

Les déclarations

```
z1=-z2=(.2in,0);
x3=-x6=.3in;
x3+y3=x6+y6=1.1in;
```

Équations linéaires... pour des points

Les déclarations

```
z1=-z2=(.2in,0);
x3=-x6=.3in;
x3+y3=x6+y6=1.1in;
```

permettent à METAPOST de déterminer les points `z1`, `z2`, `z3` et `z6`

Équations linéaires, *médiation*

Équations linéaires, *médiation*

Opérateur de médiation :

```
z4 = a[z3,z6];
```

Équations linéaires, *médiation*

Opérateur de médiation :

$$z4 = a[z3,z6];$$

METAPOST déterminera que `z4 = z3 + a(z6-z3)`

Équations linéaires, *médiation*

Opérateur de médiation :

```
z4 = a[z3,z6];
```

METAPOST déterminera que `z4 = z3 + a(z6-z3)`

Point d'intersection entre deux segments à l'aide de deux équations de médiation et la variable particulière `whatever` :

Équations linéaires, *médiation*

Opérateur de médiation :

```
z4 = a[z3,z6];
```

METAPOST déterminera que `z4 = z3 + a(z6-z3)`

Point d'intersection entre deux segments à l'aide de deux équations de médiation et la variable particulière `whatever` :

```
z20=whatever[z1,z3]=whatever[z2,z4];
```

Équations linéaires, *médiation*

Opérateur de médiation :

```
z4 = a[z3,z6];
```

METAPOST déterminera que `z4 = z3 + a(z6-z3)`

Point d'intersection entre deux segments à l'aide de deux équations de médiation et la variable particulière `whatever` :

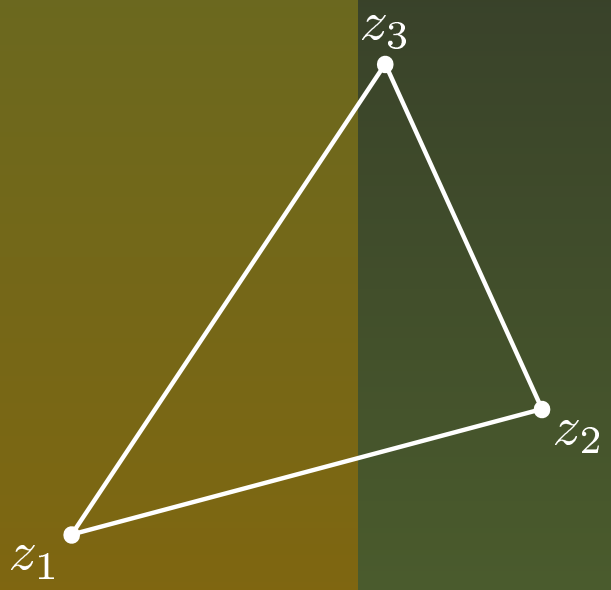
```
z20=whatever[z1,z3]=whatever[z2,z4];
```

Si `z20` $\in$ `[z1, z3]` et $\in$ `[z2, z4]` alors `z20` est déterminé, sinon message d'erreur.

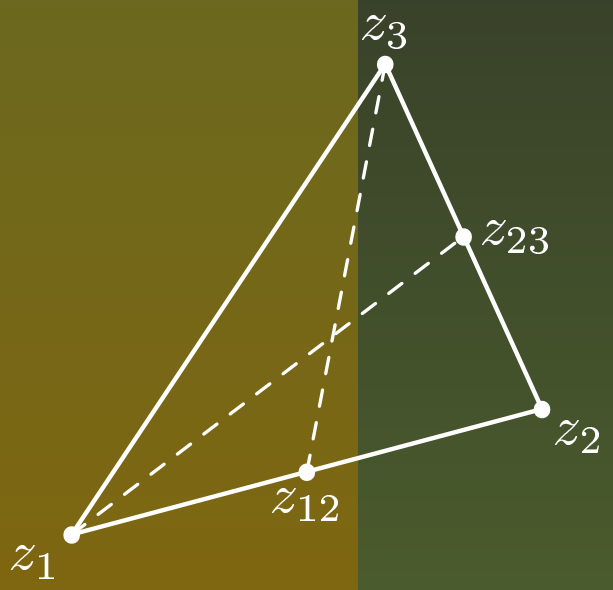
Application : barycentre du triangle

Application : barycentre du triangle

```
z1 = (0, 0) ; z2 = (4cm, 1cm) ; z3 = (2.5cm, 4cm) ;
draw z1--z2--z3--cycle withpen pencircle scaled .8bp ;
dotlabel.llft(btex $z_1$ etex, z1) ;
dotlabel.lrt(btex $z_2$ etex, z2) ;
dotlabel.top(btex $z_3$ etex, z3) ;
```

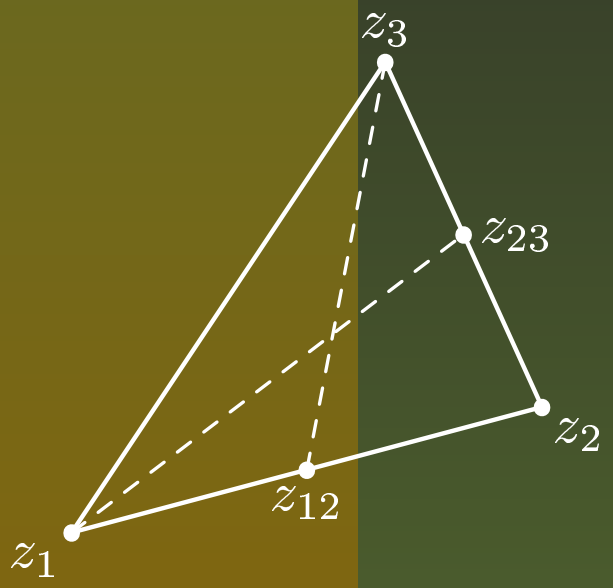


Application : barycentre du triangle



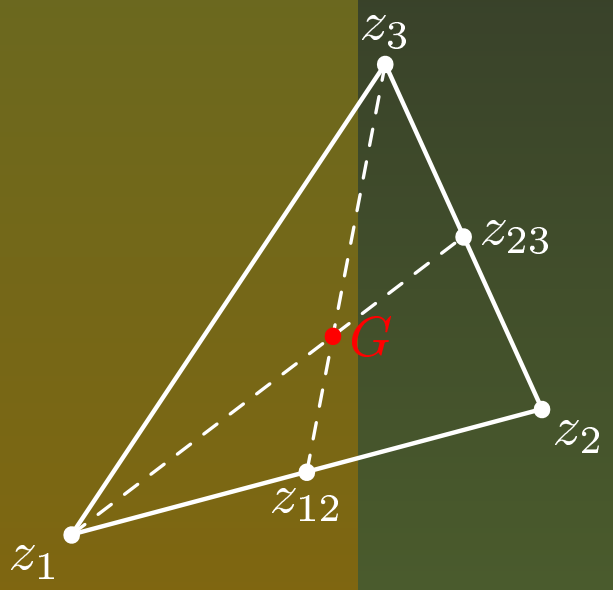

```
z1 = (0, 0) ; z2 = (4cm, 1cm) ; z3 = (2.5cm, 4cm) ;
draw z1--z2--z3--cycle withpen pencircle scaled .8bp ;
dotlabel.llft(btex $z_1$ etex, z1) ;
dotlabel.lrt(btex $z_2$ etex, z2) ;
dotlabel.top(btex $z_3$ etex, z3) ;
z12 = 1/2[z1, z2] ; dotlabel.bot(btex $z_{12}$ etex, z12) ;
z23 = 0.5[z2, z3] ; dotlabel.rt(btex $z_{23}$ etex, z23) ;
draw z1 -- z23 withpen pencircle scaled .6bp dashed evenly ;
draw z3 -- z12 withpen pencircle scaled .6bp dashed evenly ;
```

Application : barycentre du triangle




```
z1 = (0, 0) ; z2 = (4cm, 1cm) ; z3 = (2.5cm, 4cm) ;
draw z1--z2--z3--cycle withpen pencircle scaled .8bp ;
dotlabel.llft(btex $z_1$ etex, z1) ;
dotlabel.lrt(btex $z_2$ etex, z2) ;
dotlabel.top(btex $z_3$ etex, z3) ;
z12 = 1/2[z1, z2] ; dotlabel.bot(btex $z_{12}$ etex, z12) ;
z23 = 0.5[z2, z3] ; dotlabel.rt(btex $z_{23}$ etex, z23) ;
draw z1 -- z23 withpen pencircle scaled .6bp dashed evenly ;
draw z3 -- z12 withpen pencircle scaled .6bp dashed evenly ;
z4 = whatever[z1, z23] = whatever[z12, z3] ;
```

Application : barycentre du triangle




```
z1 = (0, 0) ; z2 = (4cm, 1cm) ; z3 = (2.5cm, 4cm) ;
draw z1--z2--z3--cycle withpen pencircle scaled .8bp ;
dotlabel.llft(btex $z_1$ etex, z1) ;
dotlabel.lrt(btex $z_2$ etex, z2) ;
dotlabel.top(btex $z_3$ etex, z3) ;
z12 = 1/2[z1, z2] ; dotlabel.bot(btex $z_{12}$ etex, z12) ;
z23 = 0.5[z2, z3] ; dotlabel.rt(btex $z_{23}$ etex, z23) ;
draw z1 -- z23 withpen pencircle scaled .6bp dashed evenly ;
draw z3 -- z12 withpen pencircle scaled .6bp dashed evenly ;
z4 = whatever[z1, z23] = whatever[z12, z3] ;
drawoptions (withcolor red) ;
dotlabel.rt(btex $G$ etex, z4) ;
endfig ;
```

Types de variables

Types de variables

- Numérique (*numeric*) : entier multiple de $\frac{1}{65536}$ et $|\text{numérique}| < 4096$;

Types de variables

- Numérique (*numeric*) : entier multiple de $\frac{1}{65536}$ et $|\text{numérique}| < 4096$;
- Couple ou paire (*pair*) de numérique ;

Types de variables

- Numérique (*numeric*) : entier multiple de $\frac{1}{65536}$ et $|\text{numérique}| < 4096$;
- Couple ou paire (*pair*) de numérique ;
- Chemin (*path*) : courbe de BÉZIER ;

Types de variables

- Numérique (*numeric*) : entier multiple de $\frac{1}{65536}$ et $|\text{numérique}| < 4096$;
- Couple ou paire (*pair*) de numérique ;
- Chemin (*path*) : courbe de BÉZIER ;
- Transformation (*transform*) de points, de chemins... de façon affine ;

Types de variables

- Numérique (*numeric*) : entier multiple de $\frac{1}{65536}$ et $|\text{numérique}| < 4096$;
- Couple ou paire (*pair*) de numérique ;
- Chemin (*path*) : courbe de BÉZIER ;
- Transformation (*transform*) de points, de chemins... de façon affine ;
- Couleur (*color*) à trois composantes (bleu, vert, rouge) ;

Types de variables

- Numérique (*numeric*) : entier multiple de $\frac{1}{65536}$ et $|\text{numérique}| < 4096$;
- Couple ou paire (*pair*) de numérique ;
- Chemin (*path*) : courbe de BÉZIER ;
- Transformation (*transform*) de points, de chemins... de façon affine ;
- Couleur (*color*) à trois composantes (bleu, vert, rouge) ;
- Chaîne de caractères (*string*) entre guillemets `"comme ceci"` ;

Types de variables

- Numérique (*numeric*) : entier multiple de $\frac{1}{65536}$ et $|\text{numérique}| < 4096$;
- Couple ou paire (*pair*) de numérique ;
- Chemin (*path*) : courbe de BÉZIER ;
- Transformation (*transform*) de points, de chemins... de façon affine ;
- Couleur (*color*) à trois composantes (bleu, vert, rouge) ;
- Chaîne de caractères (*string*) entre guillemets `"comme ceci"` ;
- Booléen (*boolean*) vrai (*true*) ou faux (*false*) ;

Types de variables

- Numérique (*numeric*) : entier multiple de $\frac{1}{65536}$ et $|\text{numérique}| < 4096$;
- Couple ou paire (*pair*) de numérique ;
- Chemin (*path*) : courbe de BÉZIER ;
- Transformation (*transform*) de points, de chemins... de façon affine ;
- Couleur (*color*) à trois composantes (bleu, vert, rouge) ;
- Chaîne de caractères (*string*) entre guillemets `"comme ceci"` ;
- Booléen (*boolean*) vrai (*true*) ou faux (*false*) ;
- Figure (*picture*) contient tout ce que METAPOST sait dessiner ;

Types de variables

- Numérique (*numeric*) : entier multiple de $\frac{1}{65536}$ et $|\text{numérique}| < 4096$;
- Couple ou paire (*pair*) de numérique ;
- Chemin (*path*) : courbe de BÉZIER ;
- Transformation (*transform*) de points, de chemins... de façon affine ;
- Couleur (*color*) à trois composantes (bleu, vert, rouge) ;
- Chaîne de caractères (*string*) entre guillemets `"comme ceci"` ;
- Booléen (*boolean*) vrai (*true*) ou faux (*false*) ;
- Figure (*picture*) contient tout ce que METAPOST sait dessiner ;
- Stylo ou plume (*pen*) pour tracer des chemins, obtenir des effets calligraphiques.

Quelques opérateurs

Quelques opérateurs

- 4 niveaux de priorité $\Rightarrow$ 4 types de formules : primaires, secondaires, tertiaires et expressions de chacun des 9 types (*cf.* *METAFONTbook*)

Quelques opérateurs

- 4 niveaux de priorité $\Rightarrow$ 4 types de formules : primaires, secondaires, tertiaires et expressions de chacun des 9 types (*cf.* METAFONTbook)
- `*` et `**` : opérateurs binaires primaires

`3*a**2` signifie $(3a)^2$ et non $3a^2$

Quelques opérateurs

- 4 niveaux de priorité $\Rightarrow$ 4 types de formules : primaires, secondaires, tertiaires et expressions de chacun des 9 types (*cf.* METAFONTbook)
- `*` et `**` : opérateurs binaires primaires

  `3*a**2` signifie $(3a)^2$ et non $3a^2$
- de même `-a**2` signifie $(-a)^2$ mais l'opérateur de soustraction `-` a un niveau de priorité inférieure et `a - b**2` signifie bien $a-(b^2)$ et non $(a-b)^2$

Quelques opérateurs

- 4 niveaux de priorité $\Rightarrow$ 4 types de formules : primaires, secondaires, tertiaires et expressions de chacun des 9 types (*cf.* METAFONTbook)
- `*` et `**` : opérateurs binaires primaires

  `3*a**2` signifie $(3a)^2$ et non $3a^2$
- de même `-a**2` signifie $(-a)^2$ mais l'opérateur de soustraction `-` a un niveau de priorité inférieure et `a - b**2` signifie bien $a-(b^2)$ et non $(a-b)^2$
- `++` et `+-+` : addition et soustraction pythagoriciennes

Quelques opérateurs

- 4 niveaux de priorité $\Rightarrow$ 4 types de formules : primaires, secondaires, tertiaires et expressions de chacun des 9 types (*cf.* METAFONTbook)
- `*` et `**` : opérateurs binaires primaires

  `3*a**2` signifie $(3a)^2$ et non $3a^2$
- de même `-a**2` signifie $(-a)^2$ mais l'opérateur de soustraction `-` a un niveau de priorité inférieure et `a - b**2` signifie bien $a-(b^2)$ et non $(a-b)^2$
- `++` et `+-+` : addition et soustraction pythagoriciennes

  `a ++ b` $\Leftrightarrow \sqrt{a^2+b^2}$ et `a +-+ b` $\Leftrightarrow \sqrt{a^2-b^2}$

Quelques opérateurs

- 4 niveaux de priorité $\Rightarrow$ 4 types de formules : primaires, secondaires, tertiaires et expressions de chacun des 9 types (*cf.* METAFONTbook)
- `*` et `**` : opérateurs binaires primaires

  `3*a**2` signifie $(3a)^2$ et non $3a^2$
- de même `-a**2` signifie $(-a)^2$ mais l'opérateur de soustraction `-` a un niveau de priorité inférieure et `a - b**2` signifie bien $a-(b^2)$ et non $(a-b)^2$
- `++` et `+-+` : addition et soustraction pythagoriciennes

  `a ++ b` $\Leftrightarrow \sqrt{a^2+b^2}$ et `a +-+ b` $\Leftrightarrow \sqrt{a^2-b^2}$
- Opérateurs booléens `and` (binaire primaire) et `or` (binaire secondaire)

Quelques opérateurs (suite)

- `substring`<*paire*> `of`<*chaîne primaire*>

Quelques opérateurs (suite)

- `substring`<*paire*> `of`<*chaîne primaire*>

  `substring`(2,4) `of "abcde"` $\leftrightarrow$ `"cd"`

Quelques opérateurs (suite)

- `substring`<*paire*> `of`<*chaîne primaire*>

  `substring`(2,4) `of "abcde"` $\leftrightarrow$ `"cd"`

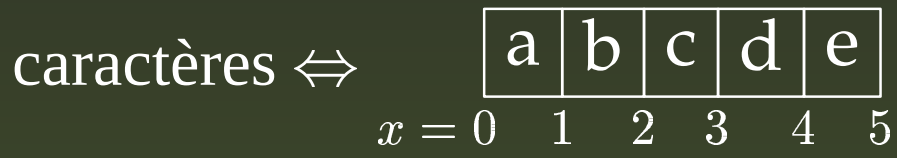
  Une chaîne de caractères $\Leftrightarrow$ 

;

Quelques opérateurs (suite)

- `substring`<*paire*> `of`<*chaîne primaire*>

  `substring`(2,4) `of "abcde"` $\leftrightarrow$ `"cd"`

  Une chaîne de caractères $\Leftrightarrow$ 

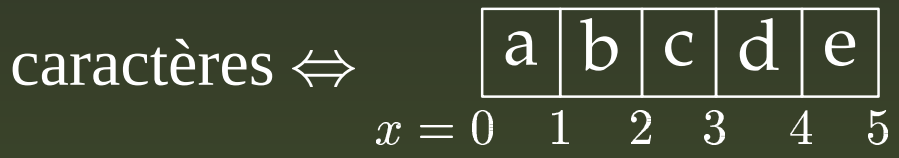
  ;

- concaténation : `"ab" & "cd"` produit `abcd`;

Quelques opérateurs (suite)

- `substring`<*paire*> `of`<*chaîne primaire*>

  `substring`(2,4) `of "abcde"` $\leftrightarrow$ `"cd"`

  Une chaîne de caractères $\Leftrightarrow$ 

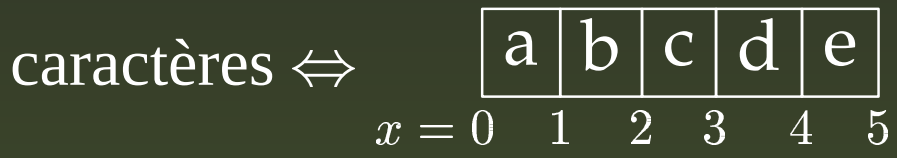
  ;

- concaténation : `"ab" & "cd"` produit `abcd`;
- *attention* à certains opérateurs

Quelques opérateurs (suite)

- `substring`<*paire*> `of`<*chaîne primaire*>

  `substring`(2,4) `of "abcde"` $\leftrightarrow$ `"cd"`

  Une chaîne de caractères $\Leftrightarrow$ 

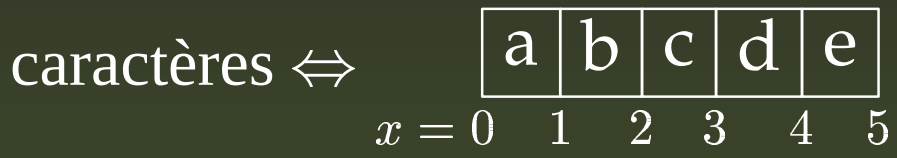
  ;

- concaténation : `"ab" & "cd"` produit `abcd`;
- *attention* à certains opérateurs
  - Médiation :

Quelques opérateurs (suite)

- `substring`<*paire*> `of`<*chaîne primaire*>

  `substring`(2,4) `of "abcde"` $\leftrightarrow$ `"cd"`

  Une chaîne de caractères $\Leftrightarrow$ 

  ;

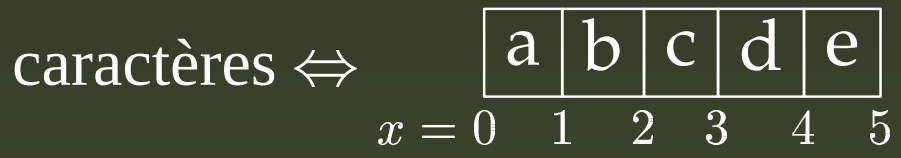
- concaténation : `"ab" & "cd"` produit `abcd`;
- *attention* à certains opérateurs
  - Médiation :

    `-1[a,b]`$\Rightarrow -b$ car $\Leftrightarrow$ `-(1[a,b])` $(= -(a + 1(b - a))$

Quelques opérateurs (suite)

- `substring`<*paire*> `of`<*chaîne primaire*>

  `substring`(2,4) `of "abcde"` ↔ `"cd"`

  Une chaîne de caractères ⇔ 

  ;

- concaténation : `"ab" & "cd"` produit `abcd`;
- *attention* à certains opérateurs
  - Médiation :

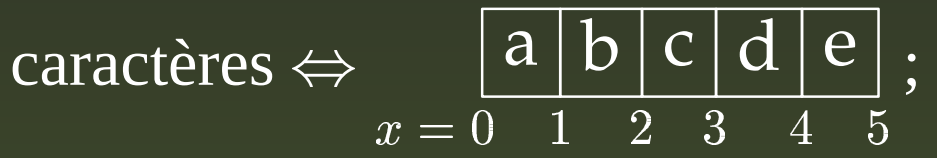
    `-1[a, b]`⇒ $-b$ car ⇔ `-(1[a, b])` $(= -(a + 1(b - a))$

    `(-1)[a, b]` ⇒ $2a - b$ $(= a - (1(b - a)))$

Quelques opérateurs (suite)

- `substring`<*paire*> `of`<*chaîne primaire*>

  `substring`(2,4) `of "abcde"` ↔ `"cd"`

  Une chaîne de caractères ⇔ 

  ;

- concaténation : `"ab" & "cd"` produit `abcd` ;
- *attention* à certains opérateurs
  - Médiation :

    `-1[a, b]` ⇒ $-b$ car ⇔ `-(1[a, b])` $(= -(a + 1(b - a))$

    `(-1)[a, b]` ⇒ $2a - b$ $(= a - (1(b - a)))$
  - `sqrt 2/3` ⇔ $\sqrt{\frac{2}{3}}$

Quelques opérateurs (suite)

- `substring`<*paire*> `of`<*chaîne primaire*>

  `substring(2,4) of "abcde"` ↔ `"cd"`

  Une chaîne de caractères ⇔ 

  ;

- concaténation : `"ab" & "cd"` produit `abcd` ;
- *attention* à certains opérateurs
  - Médiation :

    `-1[a,b]` ⇒ $-b$ car ⇔ `-(1[a,b])` $(= -(a + 1(b - a))$

    `(-1)[a,b]` ⇒ $2a - b$ $(= a - (1(b - a)))$
  - `sqrt 2/3` ⇔ $\sqrt{\frac{2}{3}}$
  - `sqrt (1+1)/3` ⇔ $\sqrt{2}/3$

Quelques opérateurs... suite

- `abs` x ou `abs` (x)

Quelques opérateurs... suite

- `abs` x ou `abs` (x)

$$\Rightarrow \begin{cases} |x| \text{ si } x \text{ est un numérique} \\ \text{la longueur euclidienne du vecteur si } x \text{ est un couple} \end{cases}$$

Quelques opérateurs... suite

- `abs` x ou `abs` (x)

$$\Rightarrow \begin{cases} |x| \text{ si } x \text{ est un numérique} \\ \text{la longueur euclidienne du vecteur si } x \text{ est un couple} \end{cases}$$

- *type* A

Quelques opérateurs... suite

- `abs` x ou `abs` (x)

$$\Rightarrow \begin{cases} |x| \text{ si } x \text{ est un numérique} \\ \text{la longueur euclidienne du vecteur si } x \text{ est un couple} \end{cases}$$

- *type* A

$$= \begin{cases} \textit{true} \text{ si A est du type } \textit{type} \\ \textit{false} \text{ sinon} \end{cases}$$

Quelques opérateurs... suite

- `abs` x ou `abs` (x)

$$\Rightarrow \begin{cases} |x| \text{ si } x \text{ est un numérique} \\ \text{la longueur euclidienne du vecteur si } x \text{ est un couple} \end{cases}$$

- *type* A

$$= \begin{cases} \textit{true} \text{ si A est du type } \textit{type} \\ \textit{false} \text{ sinon} \end{cases}$$

- `3in` $\Leftrightarrow$ `3*in;`

Quelques opérateurs... suite

- `abs` x ou `abs` (x)

$$\Rightarrow \begin{cases} |x| \text{ si } x \text{ est un numérique} \\ \text{la longueur euclidienne du vecteur si } x \text{ est un couple} \end{cases}$$

- *type* A

$$= \begin{cases} \textit{true} \text{ si A est du type } \textit{type} \\ \textit{false} \text{ sinon} \end{cases}$$

- `3in` $\Leftrightarrow$ `3*in;`
- `2/3x` $\Leftrightarrow \frac{2}{3}x$ mais `(2)/3x` $\Leftrightarrow \frac{2}{3x}$ et `3 3` est illégal.

Les variables

À déclarer soit :

Les variables

À déclarer soit :

- dans le préambule ;

Les variables

À déclarer soit :

- dans le préambule ;
- dans chaque bloc `beginfig(`i`) endfig` (locales à la figure).

Les variables

À déclarer soit :

- dans le préambule ;
- dans chaque bloc `beginfig(`i`) endfig` (locales à la figure).

Déclaration sous la forme : *type variable(s)* séparées par des virgules

Les variables

À déclarer soit :

- dans le préambule ;
- dans chaque bloc `beginfig(`i`) endfig` (locales à la figure).

Déclaration sous la forme : *type variable(s)* séparées par des virgules

`pair` ab, a, b`;` : trois couples ab, a, b à initialiser.

Les variables

À déclarer soit :

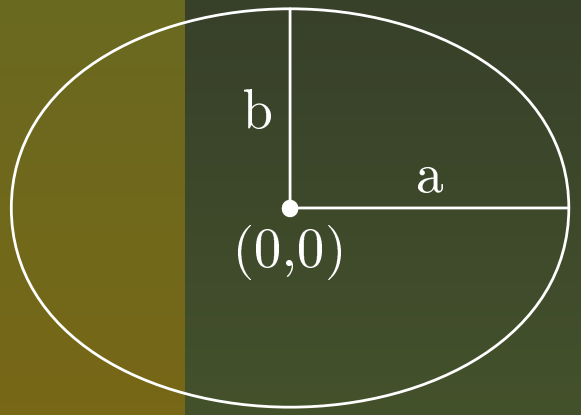
- dans le préambule ;
- dans chaque bloc `beginfig(`i`) endfig` (locales à la figure).

Déclaration sous la forme : *type variable(s)* séparées par des virgules

`pair` ab, a, b`;` : trois couples ab, a, b à initialiser.

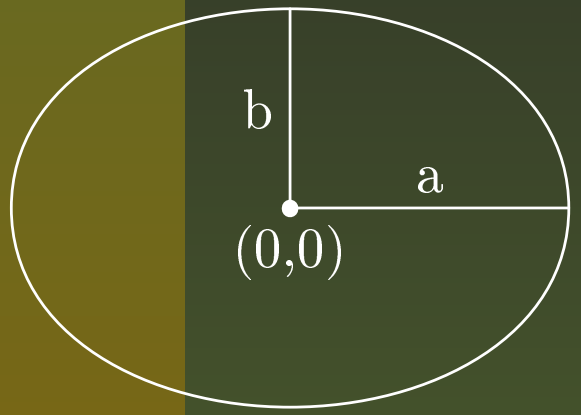
La déclaration `type` $toto[]$ permet de disposer de $totoi$, i variant de 0 à ce que l'on veut.

Du texte dans les graphiques




```
beginfig(6) ;
numeric a,b ;
a=.7in ; b=.5in ; z0=(0,0) ;
z1=-z3=(a,0) ; z2=-z4=(0,b) ;
path p ; p = z1..z2..z3..z4..cycle ;
draw p withcolor white ;
draw z1--z0--z2 withcolor white ;
label.top("a", .5[z0,z1]) ;
label.lft("b", .5[z0,z2]) ;
dotlabel.bot("(0,0)", z0) ;
endfig ;
```

Du texte dans les graphiques




```
beginfig(6) ;
numeric a,b ;
a=.7in ; b=.5in ; z0=(0,0) ;
z1=-z3=(a,0) ; z2=-z4=(0,b) ;
path p ; p = z1..z2..z3..z4..cycle ;
draw p withcolor white ;
draw z1--z0--z2 withcolor white ;
label.top("a", .5[z0,z1]) ;
label.lft("b", .5[z0,z2]) ;
dotlabel.bot("(0,0)", z0) ;
endfig ;
```

`label`<*suffixe d'étiquette*>(<*chaîne ou dessin*>,<*paire*>)

Des étiquettes composées

Dans la déclation

`label`<*suffixe d'étiquette*>(<*chaîne ou dessin*>,<*paire*>)

si la <*chaîne ou dessin*> est

Des étiquettes composées

Dans la déclation

`label`<*suffixe d'étiquette*>(<*chaîne ou dessin*>,<*paire*>)

si la <*chaîne ou dessin*> est

`btex`<*commandes de mise en forme*>`etex`

Des étiquettes composées

Dans la déclation

`label`<*suffixe d'étiquette*>(<*chaîne ou dessin*>,<*paire*>)

si la <*chaîne ou dessin*> est

`btex`<*commandes de mise en forme*>`etex`

le paramètre <*commandes de mise en forme*> est traité par TEX (ou LATEX) et traduit en une figure (*picture*).

Des étiquettes composées

Dans la déclation

`label`<*suffixe d'étiquette*>(<*chaîne ou dessin*>,<*paire*>)

si la <*chaîne ou dessin*> est

`btex`<*commandes de mise en forme*>`etex`

le paramètre <*commandes de mise en forme*> est traité par TEX (ou LATEX) et traduit en une figure (*picture*).

- `label.lrt(btex $ \sqrt x $ etex, (3,sqrt 3)*u);`

  place l'étiquette $\sqrt{x}$ sous et à droite du point `(3,sqrt 3)*u`

Des étiquettes composées

Dans la déclation

`label`<*suffixe d'étiquette*>(<*chaîne ou dessin*>,<*paire*>)

si la <*chaîne ou dessin*> est

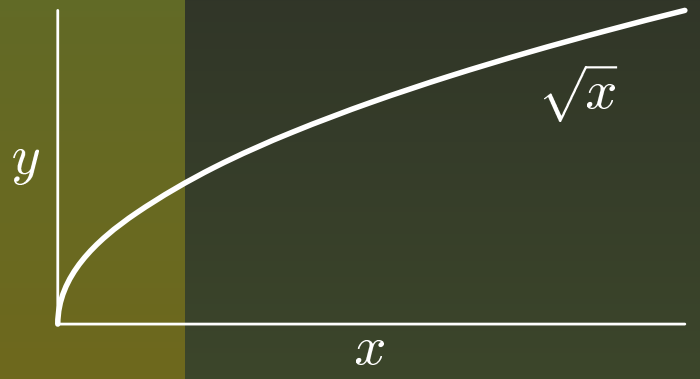
`btex`<*commandes de mise en forme*>`etex`

le paramètre <*commandes de mise en forme*> est traité par TEX (ou LATEX) et traduit en une figure (*picture*).

- `label.lrt(btex $ \sqrt x $ etex, (3,sqrt 3)*u);`

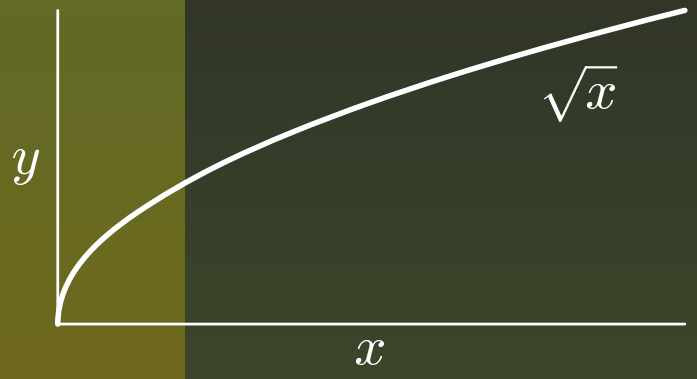
  place l'étiquette $\sqrt{x}$ sous et à droite du point `(3,sqrt 3)*u`
- `label.lft(btex $\displaystyle y={2\over 1+\cos x}$ etex, (120ux, 4uy));` place l'étiquette $\frac{2}{1+\cos x}$ au point `(120ux, 4uy)`

Des étiquettes composées : exemples



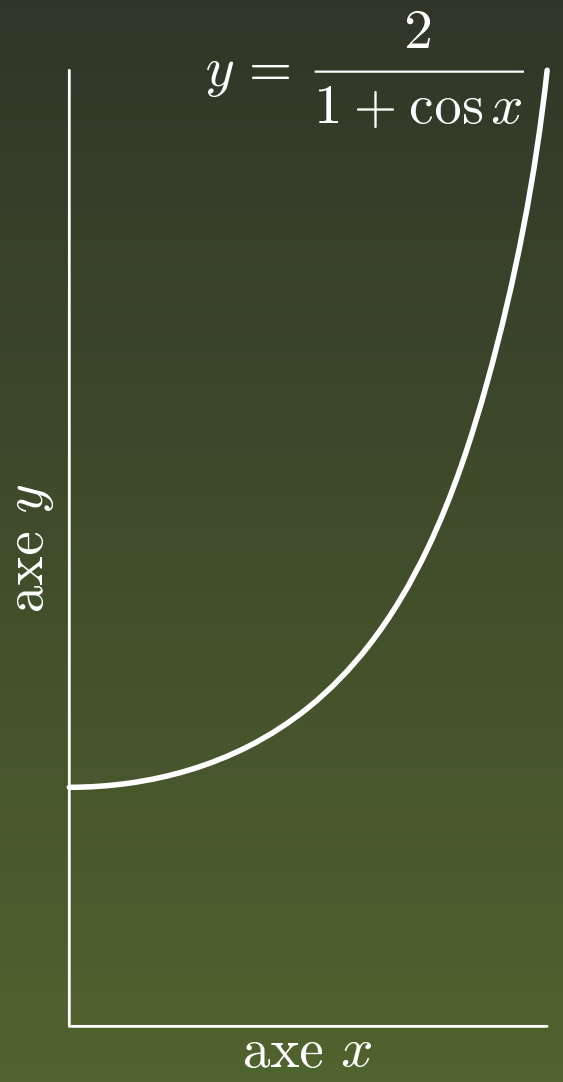

```
label.lrt(btex $\sqrt x$
etex,(3,sqrt 3)*u) ;
```

Des étiquettes composées : exemples




```
label.lrt(btex $\sqrt x$
etex, (3,sqrt 3)*u) ;

label.lft(btex
$\displaystyle
y={2\over 1+\cos x}$ etex,
(120ux, 4uy));
```



Des étiquettes composées : suite et fin

Pour éviter de coder en TEX et pour utiliser LATEX

Des étiquettes composées : suite et fin

Pour éviter de coder en TEX et pour utiliser LATEX il suffira de rajouter en en-tête :

```
verbatimtex
\documentclass{article}
\usepackage[latin1]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{amsmath}
\usepackage[frenchb]{babel}
\begin{document}
etex ;
```

Des étiquettes composées : suite et fin

Pour éviter de coder en TEX et pour utiliser LATEX il suffira de rajouter en en-tête :

```
verbatimtex
\documentclass{article}
\usepackage[latin1]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{amsmath}
\usepackage[frenchb]{babel}
\begin{document}
etex ;
```

pour retrouver le confort de LATEX et coder $\dfrac{2}{1+\cos x}$ plus naturellement par

```
$ \dfrac{2}{1+\cos x}$
```

et non

```
$\displaystyle y={2\over
1+\cos x}$
```

Des étiquettes composées : suite et fin

Pour éviter de coder en TEX et pour utiliser LATEX il suffira de rajouter en en-tête :

```
verbatimtex
\documentclass{article}
\usepackage[latin1]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{amsmath}
\usepackage[frenchb]{babel}
\begin{document}
etex ;
```

pour retrouver le confort de LATEX et coder $\frac{2}{1+\cos x}$ plus naturellement par

```
$ \dfrac{2}{1+\cos x}$
```

et non

```
$\displaystyle y={2\over
1+\cos x}$
```

Sans oublier la variable d'environnement TEX=latex

De la couleur...

De la couleur...

avec `withcolor`<*couleur*> pour colorer la courbe tracée avec `draw`

draw z0 -- z1 .. z2 withcolor red

De la couleur...

avec `withcolor`<*couleur*> pour colorer la courbe tracée avec `draw`

draw z0 -- z1 .. z2 withcolor red

- Couleurs prédéfinies : `black`, `white`, `red`, `green`, `blue`;

De la couleur...

avec `withcolor`<*couleur*> pour colorer la courbe tracée avec `draw`

draw z0 -- z1 .. z2 withcolor red

- Couleurs prédéfinies : `black`, `white`, `red`, `green`, `blue` ;
- *color macouleur ;* et *macouleur = (x, y, z)* ;

  ($0 \leq x, y, z \leq 1$ composantes rouge, verte et bleue de *macouleur*) ;

De la couleur...

avec `withcolor`<*couleur*> pour colorer la courbe tracée avec `draw`

draw z0 -- z1 .. z2 withcolor red

- Couleurs prédéfinies : `black`, `white`, `red`, `green`, `blue` ;
- *color macouleur ;* et *macouleur = (x, y, z)* ;

  ($0 \leq x, y, z \leq 1$ composantes rouge, verte et bleue de *macouleur*) ;
- black = (0, 0, 0) et white = (1, 1, 1) ;

De la couleur...

avec `withcolor`<*couleur*> pour colorer la courbe tracée avec `draw`

draw z0 -- z1 .. z2 withcolor red

- Couleurs prédéfinies : `black`, `white`, `red`, `green`, `blue` ;
- *color macouleur ;* et *macouleur = (x, y, z)* ;

  ($0 \leq x, y, z \leq 1$ composantes rouge, verte et bleue de *macouleur*) ;
- black = (0, 0, 0) et white = (1, 1, 1) ;
- Gris = (.4, .4, .4) = 0.4white ;

De la couleur...

avec `withcolor`<*couleur*> pour colorer la courbe tracée avec `draw`

draw z0 -- z1 .. z2 withcolor red

- Couleurs prédéfinies : `black`, `white`, `red`, `green`, `blue` ;
- *color macouleur ;* et *macouleur = (x, y, z)* ;

  ($0 \leq x, y, z \leq 1$ composantes rouge, verte et bleue de *macouleur*) ;
- black = (0, 0, 0) et white = (1, 1, 1) ;
- Gris = (.4, .4, .4) = 0.4white ;
- *macouleur = 0.3[red, green]* pour les mélanges.

De la couleur...

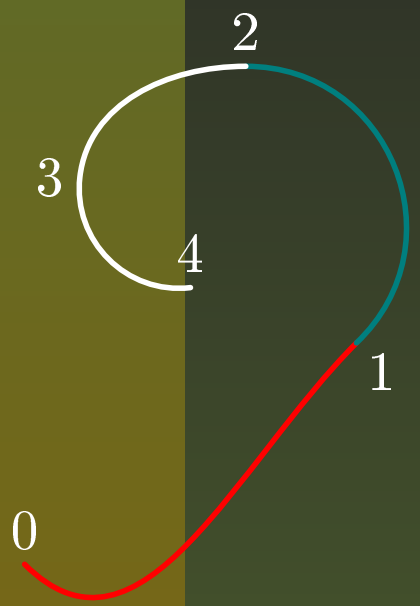
avec `withcolor`<*couleur*> pour colorer la courbe tracée avec `draw`

draw z0 -- z1 .. z2 withcolor red

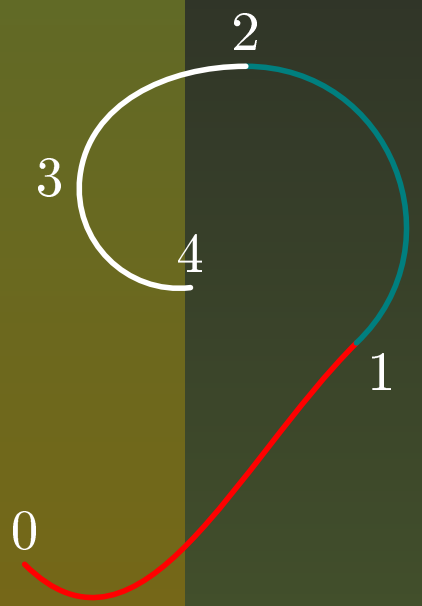
- Couleurs prédéfinies : `black`, `white`, `red`, `green`, `blue` ;
- *color macouleur ;* et *macouleur = (x, y, z)* ;

  ($0 \leq x, y, z \leq 1$ composantes rouge, verte et bleue de *macouleur*) ;
- black = (0, 0, 0) et white = (1, 1, 1) ;
- Gris = (.4, .4, .4) = 0.4white ;
- *macouleur = 0.3[red, green]* pour les mélanges.
- Additionner, soustraction des couleurs, multiplication, division par un numérique.

De la couleur...

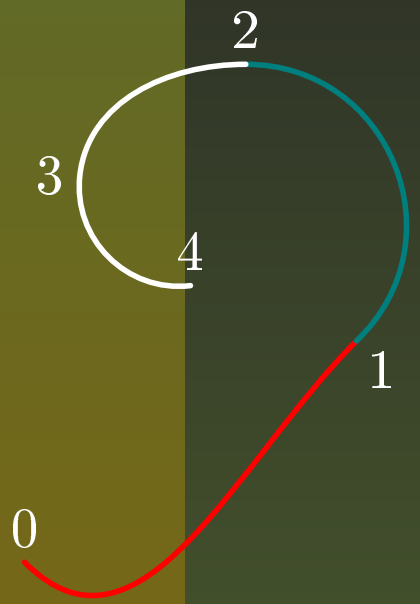


De la couleur...




```
draw z0{1,-1} .. z1{dir 45} withcolor red;
draw z1 .. z2{left} withcolor 0.5[blue,green];
draw z2{left} .. z3 .. z4 withcolor (1,1,1);
```

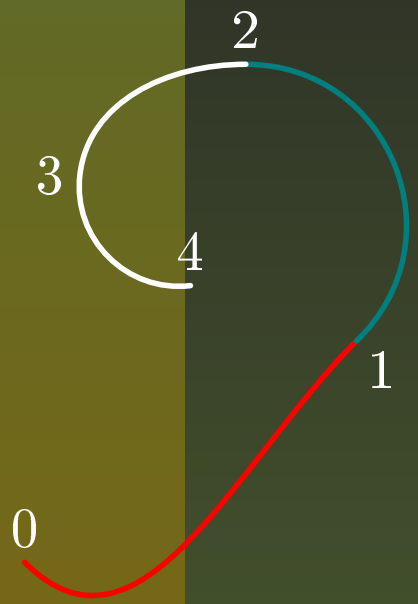
De la couleur...




```
draw z0{1,-1} .. z1{dir 45} withcolor red;
draw z1 .. z2{left} withcolor 0.5[blue,green];
draw z2{left} .. z3 .. z4 withcolor (1,1,1);
```

- `drawoptions(withcolor 0.3[red,white])` définit la couleur courante pour la figure ;

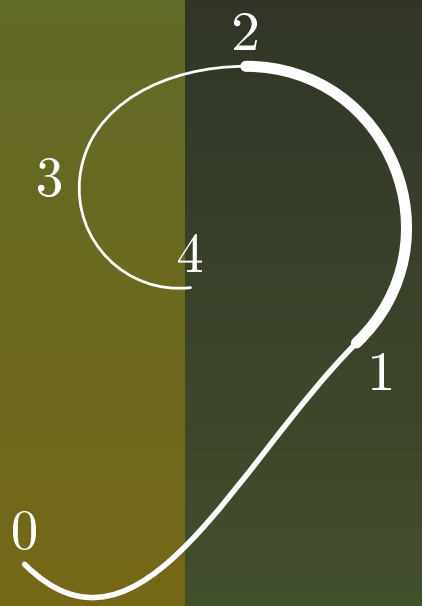
De la couleur...



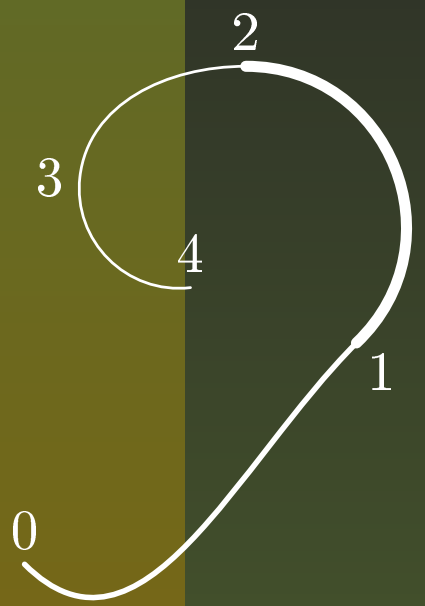

```
draw z0{1,-1} .. z1{dir 45} withcolor red;
draw z1 .. z2{left} withcolor 0.5[blue,green];
draw z2{left} .. z3 .. z4 withcolor (1,1,1);
```

- `drawoptions(withcolor 0.3[red,white])` définit la couleur courante pour la figure ;
- `drawoptions( )` restaure la couleur par défaut.

Des plumes...

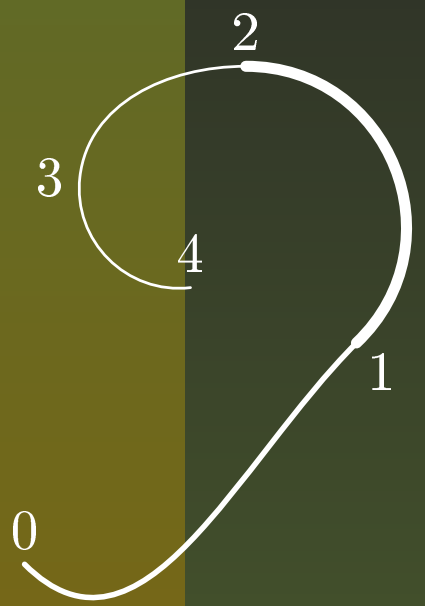


Des plumes...




```
draw z0{1,-1} .. z1{dir 45} withpen pencircle
scaled 1bp;
draw z1 .. z2{left} withpen pencircle scaled 2bp;
draw z2{left} .. z3 .. z4 withpen pencircle
scaled 0.5bp;
```

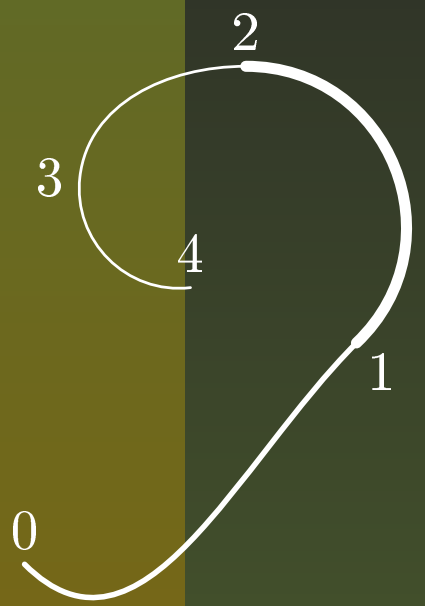
Des plumes...




```
draw z0{1,-1} .. z1{dir 45} withpen pencircle
scaled 1bp;
draw z1 .. z2{left} withpen pencircle scaled 2bp;
draw z2{left} .. z3 .. z4 withpen pencircle
scaled 0.5bp;
```

- `draw` <*chemin*> `withpen` <*expression stylo*>

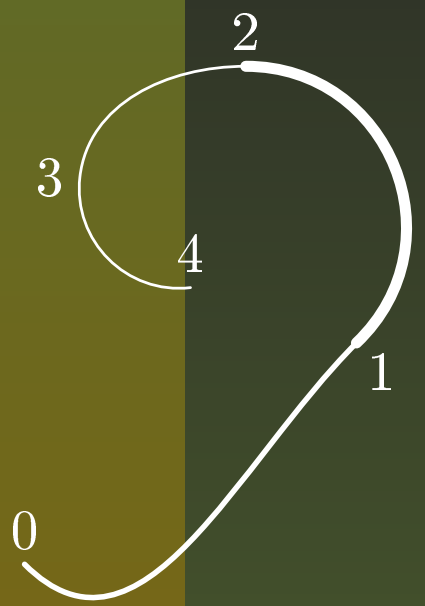
Des plumes...




```
draw z0{1,-1} .. z1{dir 45} withpen pencircle
scaled 1bp;
draw z1 .. z2{left} withpen pencircle scaled 2bp;
draw z2{left} .. z3 .. z4 withpen pencircle
scaled 0.5bp;
```

- `draw` <*chemin*> `withpen` <*expression stylo*>
- `drawoptions(withpen pencircle scaled 1mm)` définit la plume courante pour la figure ;

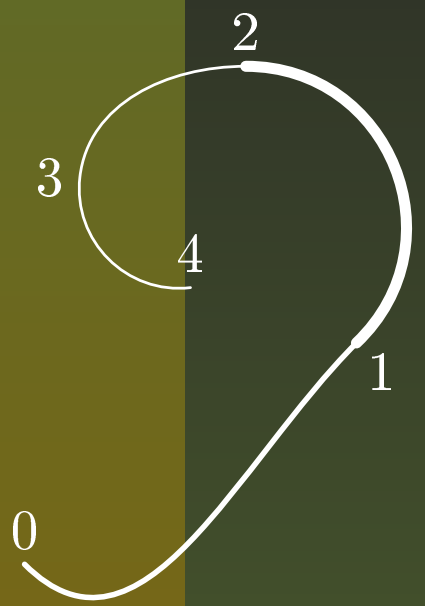
Des plumes...




```
draw z0{1,-1} .. z1{dir 45} withpen pencircle
scaled 1bp;
draw z1 .. z2{left} withpen pencircle scaled 2bp;
draw z2{left} .. z3 .. z4 withpen pencircle
scaled 0.5bp;
```

- `draw` <*chemin*> `withpen` <*expression stylo*>
- `drawoptions(withpen pencircle scaled 1mm)` définit la plume courante pour la figure ;
- `pickup pencircle scaled 1mm;`

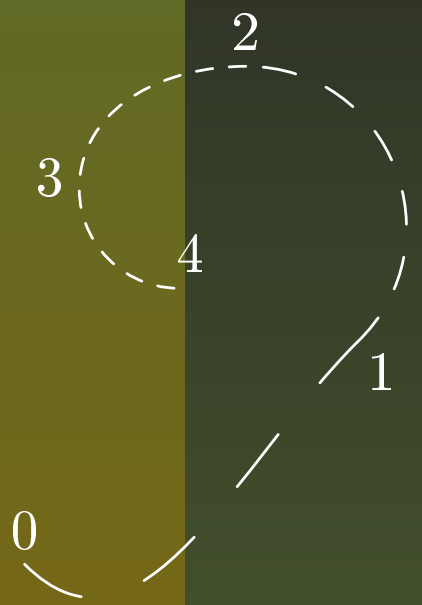
Des plumes...



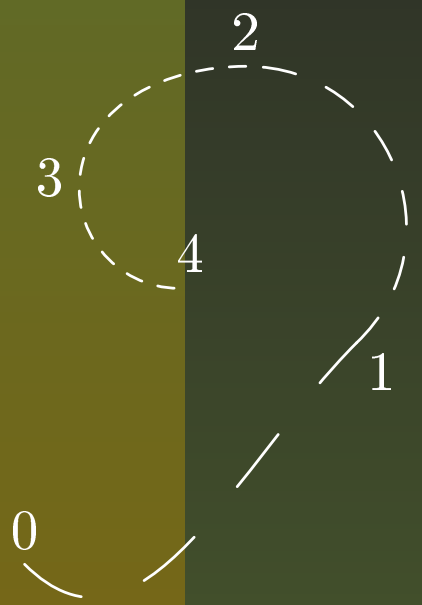

```
draw z0{1,-1} .. z1{dir 45} withpen pencircle
scaled 1bp;
draw z1 .. z2{left} withpen pencircle scaled 2bp;
draw z2{left} .. z3 .. z4 withpen pencircle
scaled 0.5bp;
```

- `draw` <*chemin*> `withpen` <*expression stylo*>
- `drawoptions(withpen pencircle scaled 1mm)` définit la plume courante pour la figure ;
- `pickup pencircle scaled 1mm;`
- `drawoptions( )` ou `pickup defaultpen` restaure la plume par défaut (pencircle scaled 0.5bp).

Des pointillés...

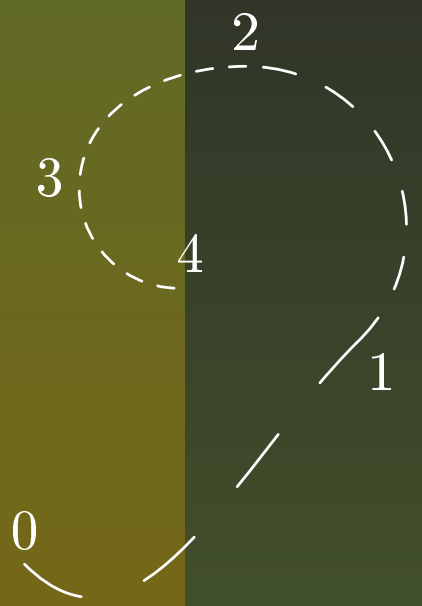


Des pointillés...




```
draw z0{1,-1} .. z1{dir 45} dashed evenly
scaled 4;
draw z1 .. z2{left} dashed evenly scaled 2;
draw z2{left} .. z3 .. z4 dashed evenly;
```

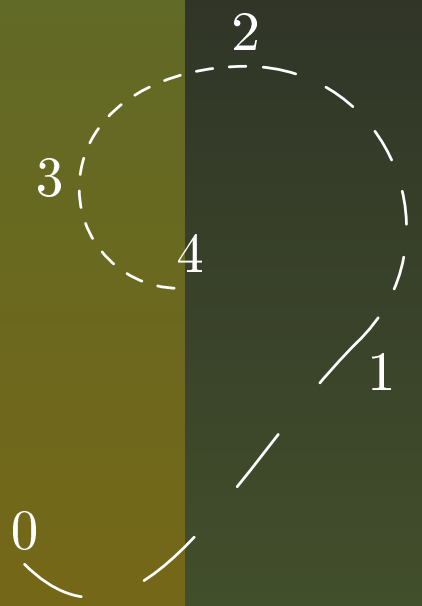
Des pointillés...




```
draw z0{1,-1} .. z1{dir 45} dashed evenly
scaled 4;
draw z1 .. z2{left} dashed evenly scaled 2;
draw z2{left} .. z3 .. z4 dashed evenly;
```

- `draw` <*chemin*> `dashed` <*motif de points*>

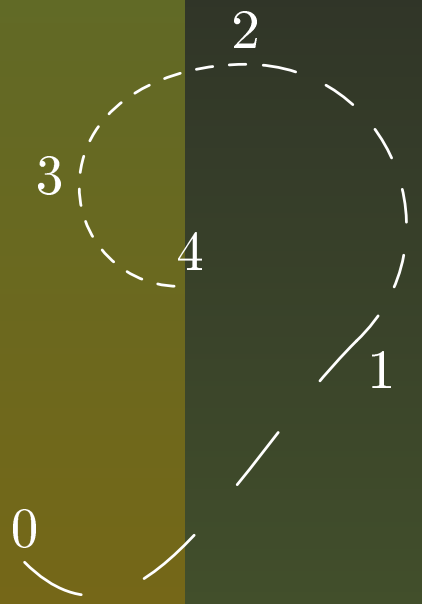
Des pointillés...




```
draw z0{1,-1} .. z1{dir 45} dashed evenly
scaled 4;
draw z1 .. z2{left} dashed evenly scaled 2;
draw z2{left} .. z3 .. z4 dashed evenly;
```

- `draw` <*chemin*> `dashed` <*motif de points*>
- Motifs prédéfinis : `evenly`, `withdots`;

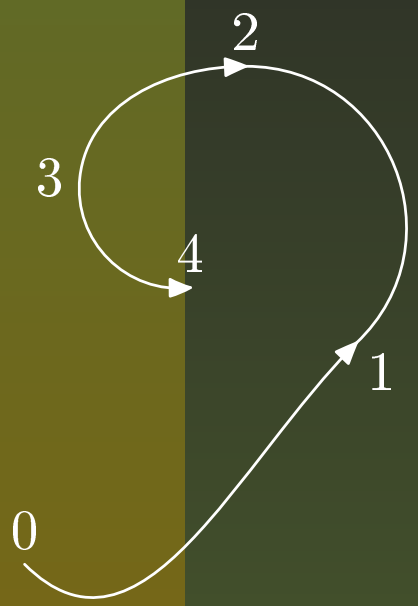
Des pointillés...



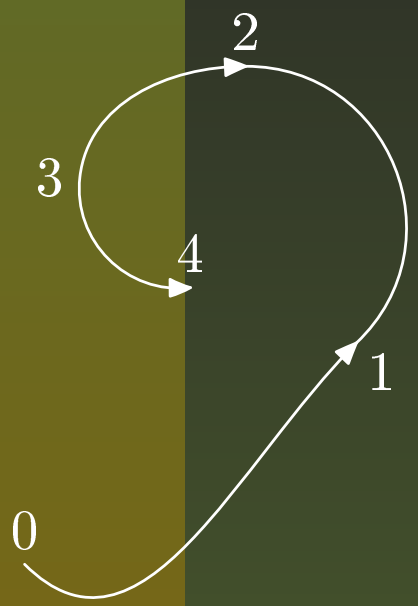

```
draw z0{1,-1} .. z1{dir 45} dashed evenly
scaled 4;
draw z1 .. z2{left} dashed evenly scaled 2;
draw z2{left} .. z3 .. z4 dashed evenly;
```

- `draw` <*chemin*> `dashed` <*motif de points*>
- Motifs prédéfinis : `evenly`, `withdots` ;
- `dashpattern(on 6bp off 12bp on 6bp)` pour un motif de points personnalisé (figure spéciale).

Des flêches...

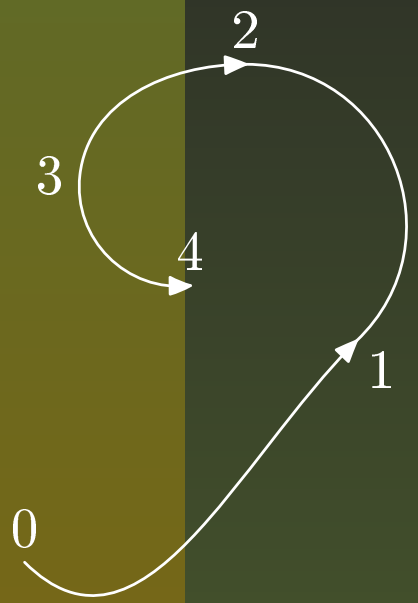


Des flêches...

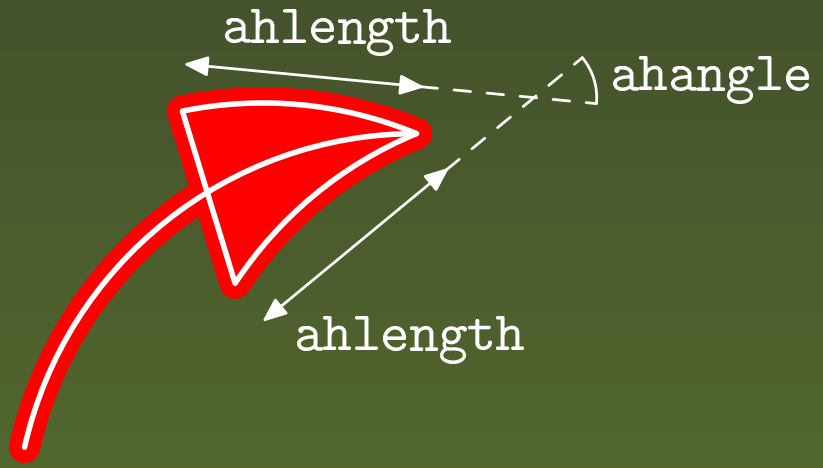



```
drawarrow z0{1,-1} .. z1{dir 45} ;
draw z1 .. z2{left} ;
drawdblarrow z2{left} .. z3 .. z4 ;
```

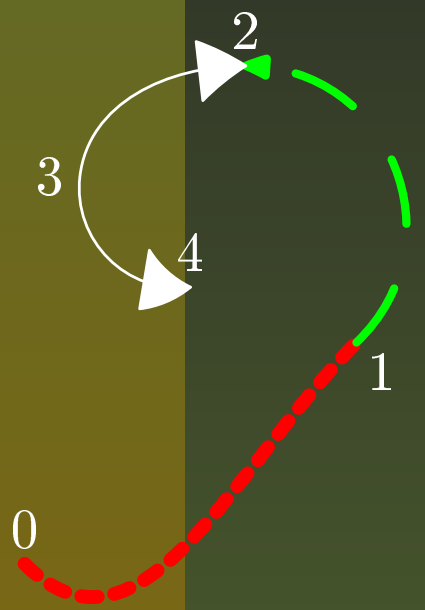
Des flêches...



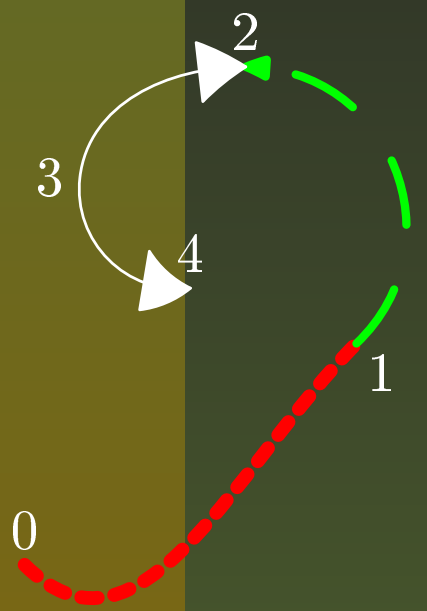

```
drawarrow z0{1,-1} .. z1{dir 45} ;
draw z1 .. z2{left} ;
drawdblarrow z2{left} .. z3 .. z4 ;
```



En résumé

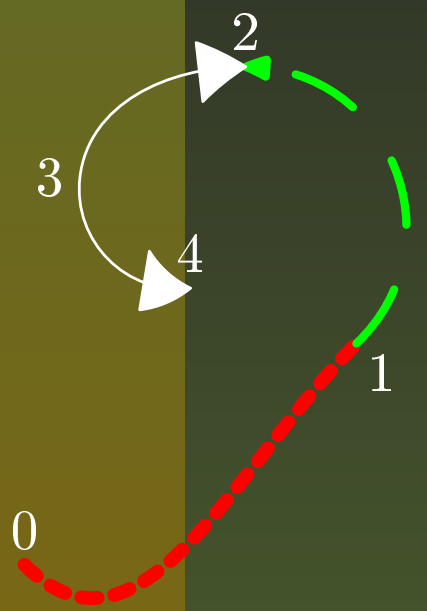


En résumé



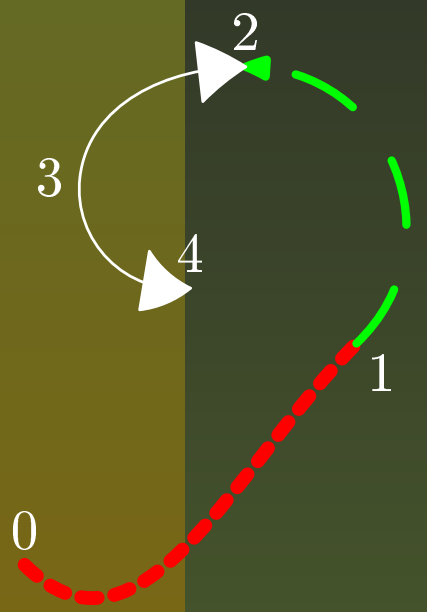

```
draw z0{1,-1} .. z1{dir 45} withcolor red
dashed evenly withpen pencircle scaled 2.5bp ;
```

En résumé



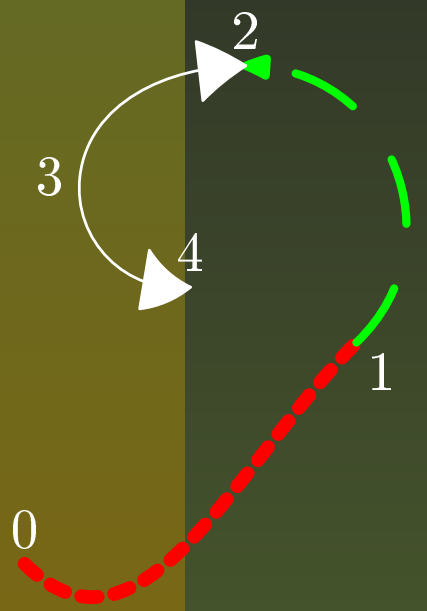

```
draw z0{1,-1} .. z1{dir 45} withcolor red
dashed evenly withpen pencircle scaled 2.5bp ;
drawarrow z1 .. z2{left} withcolor green
dashed evenly scaled 4 withpen
pencircle scaled 1.5bp ;
```

En résumé




```
draw z0{1,-1} .. z1{dir 45} withcolor red
dashed evenly withpen pencircle scaled 2.5bp ;
drawarrow z1 .. z2{left} withcolor green
dashed evenly scaled 4 withpen
pencircle scaled 1.5bp ;
ahlength := 10bp ;
ahangle := 65 ;
```

En résumé




```
draw z0{1,-1} .. z1{dir 45} withcolor red
dashed evenly withpen pencircle scaled 2.5bp;
drawarrow z1 .. z2{left} withcolor green
dashed evenly scaled 4 withpen
pencircle scaled 1.5bp;
ahlength := 10bp;
ahangle := 65;
drawdblarrow z2{left} .. z3 .. z4;
```

Des chemins

Des chemins

- Déclaration : path p, q ;

Des chemins

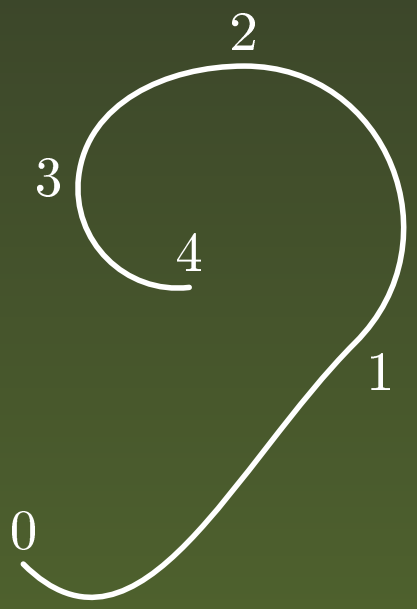
- Déclaration : path p, q ;
- Initialisation : p = z0{1, -1} .. z1{dir 45} .. z2{left} .. z3 .. z4 ; (ouvert)

Des chemins

- Déclaration : path p, q ;
- Initialisation : p = z0{1, -1} .. z1{dir 45} .. z2{left} .. z3 .. z4 ; (ouvert)
- Initialisation : q = p .. cycle ; (fermé)

Des chemins

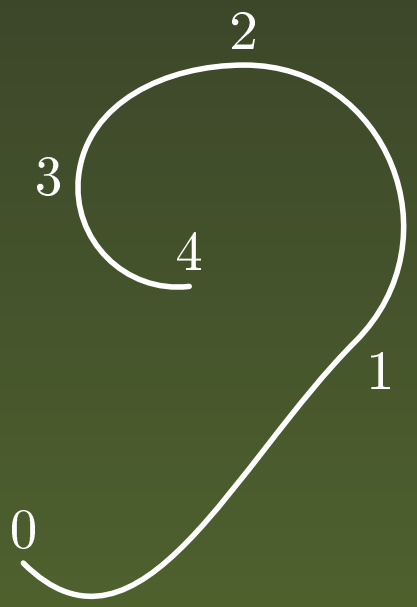
- Déclaration : path p, q ;
- Initialisation : p = z0{1, -1} .. z1{dir 45} .. z2{left} .. z3 .. z4 ; (ouvert)
- Initialisation : q = p .. cycle ; (fermé)
- Dessin : draw p ;



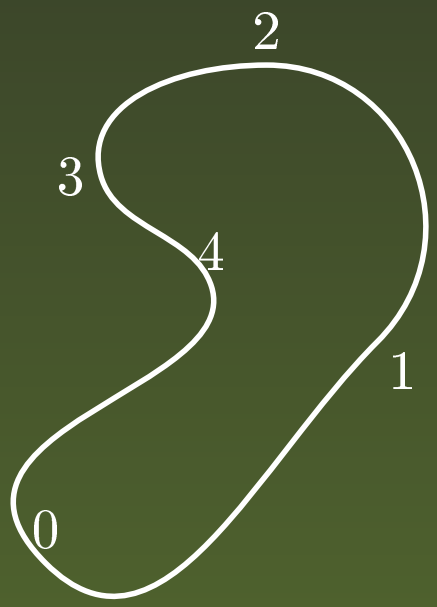
Chemin p

Des chemins

- Déclaration : path p, q ;
- Initialisation : p = z0{1, -1} .. z1{dir 45} .. z2{left} .. z3 .. z4 ; (ouvert)
- Initialisation : q = p .. cycle ; (fermé)
- Dessin : draw p ; draw q ;



Chemin p



Chemin q

Des chemins particuliers

Des chemins prédéfinis du *package* plain de METAPOST

Des chemins particuliers

Des chemins prédéfinis du *package* plain de METAPOST

- `fullcircle` : cercle de diamètre 1bp centré en (0, 0) ;

Des chemins particuliers

Des chemins prédéfinis du *package* plain de METAPOST

- `fullcircle` : cercle de diamètre 1bp centré en (0, 0) ;
- `halfcircle` : moitié supérieure du précédent (sens trigonométrique) ;

Des chemins particuliers

Des chemins prédéfinis du *package* plain de METAPOST

- `fullcircle` : cercle de diamètre 1bp centré en (0, 0) ;
- `halfcircle` : moitié supérieure du précédent (sens trigonométrique) ;
- `quartercircle` : moitié droite du précédent (sens trigonométrique) ;

Des chemins particuliers

Des chemins prédéfinis du *package* plain de METAPOST

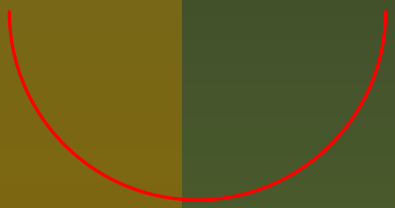
- `fullcircle` : cercle de diamètre 1bp centré en (0, 0) ;
- `halfcircle` : moitié supérieure du précédent (sens trigonométrique) ;
- `quartercircle` : moitié droite du précédent (sens trigonométrique) ;
- `unitsquare` : carré passant par les points (0, 0), (1, 0), (1, 1) et (0, 1) ;

Des chemins fermés et de la couleur

Colorer l’intérieur d’un chemin fermé par la macro `fill`, `unfill` pour « laver ».

Des chemins fermés et de la couleur

Colorer l'intérieur d'un chemin fermé par la macro `fill`, `unfill` pour « laver ».



```
beginfig(18);
drawoptions(withcolor white);
path p;
p = (-1cm,0)..(0,-1cm)..(1cm,0);
draw p withcolor red;
```

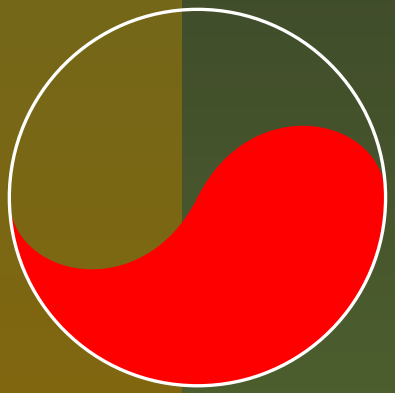
Des chemins fermés et de la couleur

Colorer l'intérieur d'un chemin fermé par la macro `fill`, `unfill` pour « laver ».

```
beginfig(18);
drawoptions(withcolor white);
path p;
p = (-1cm,0)..(0,-1cm)..(1cm,0);
draw p withcolor red;
fill  p{up}..(0,0){-1,-2}..{up}cycle
withcolor red;
```

Des chemins fermés et de la couleur

Colorer l'intérieur d'un chemin fermé par la macro `fill`, `unfill` pour « laver ».



```
beginfig(18);
drawoptions(withcolor white);
path p;
p = (-1cm,0)..(0,-1cm)..(1cm,0);
draw p withcolor red;
fill p{up}..(0,0){-1,-2}..{up}cycle
withcolor red;
draw p..(0,1cm)..cycle;
endfig;
```

Construire des chemins fermés...

...automatiquement par la macro `buildcycle`.

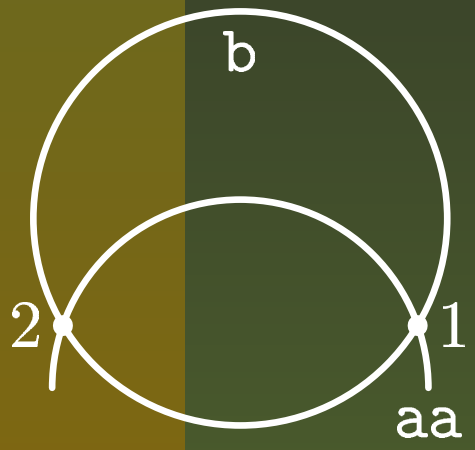
Construire des chemins fermés...

...automatiquement par la macro `buildcycle`.

```
path aa, b, cheminferme ;
aa = halfcircle scaled 2cm ;
draw aa ;
```

Construire des chemins fermés...

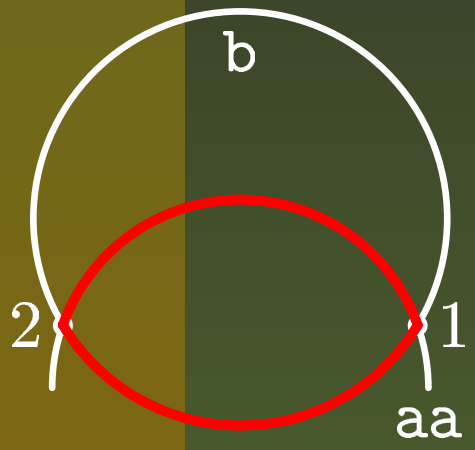
...automatiquement par la macro `buildcycle`.




```
path aa, b, cheminferme ;
aa = halfcircle scaled 2cm ;
draw aa ;
b = fullcircle scaled 2.2cm shifted
(0, 0.9cm) ;
draw b ;
```

Construire des chemins fermés...

...automatiquement par la macro `buildcycle`.




```
path aa, b, cheminferme ;
aa = halfcircle scaled 2cm ;
draw aa ;
b = fullcircle scaled 2.2cm shifted
(0, 0.9cm) ;
draw b ;
cheminferme = buildcycle(aa, b) ;
draw cheminferme withcolor red ;
```

Intersections des courbes de BÉZIER

Intersections des courbes de BÉZIER

- `a intersectionpoint b` renvoie les coordonnées du point en question s'il existe, sinon il renvoie un message d'erreur ;

Intersections des courbes de BÉZIER

- `a intersectionpoint b` renvoie les coordonnées du point en question s'il existe, sinon il renvoie un message d'erreur ;
- `a intersectiontimes b` renvoie les valeurs de t sur chacune des courbes pour lesquelles il y a intersection. S'il y a plusieurs points d'intersection, la macro renvoie les valeurs les plus petites (*cf.* METAFONTbook), sinon le couple $(-1, -1)$;

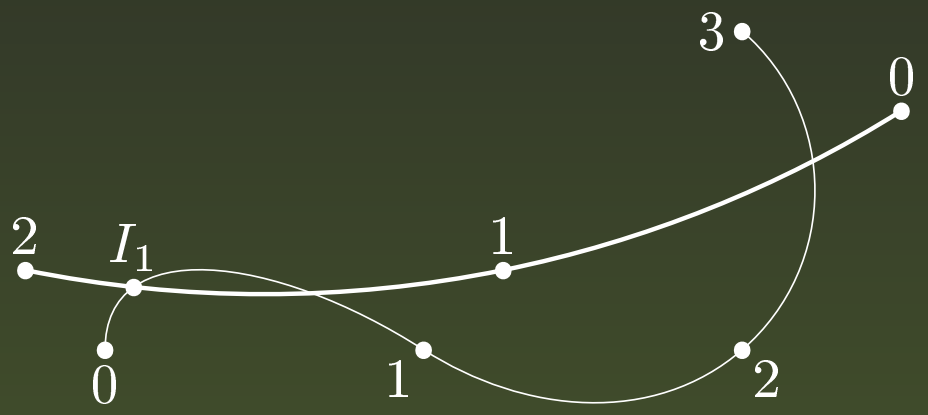
Intersections des courbes de BÉZIER

- `a intersectionpoint b` renvoie les coordonnées du point en question s'il existe, sinon il renvoie un message d'erreur ;
- `a intersectiontimes b` renvoie les valeurs de t sur chacune des courbes pour lesquelles il y a intersection. S'il y a plusieurs points d'intersection, la macro renvoie les valeurs les plus petites (*cf.* METAFONTbook), sinon le couple $(-1, -1)$;
- il pourra être nécessaire de parcourir un ou les deux chemins en sens inverse avec la macro `reverse` appliquée à un chemin.

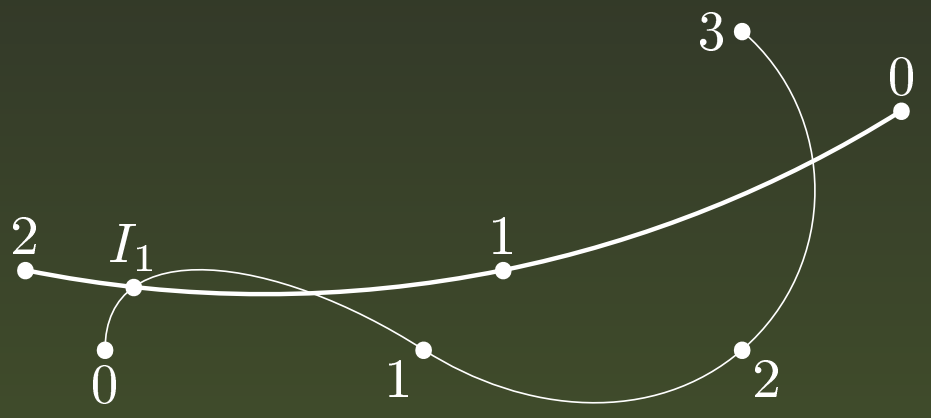
Intersections des courbes de BÉZIER

- `a intersectionpoint b` renvoie les coordonnées du point en question s'il existe, sinon il renvoie un message d'erreur ;
- `a intersectiontimes b` renvoie les valeurs de t sur chacune des courbes pour lesquelles il y a intersection. S'il y a plusieurs points d'intersection, la macro renvoie les valeurs les plus petites (*cf.* METAFONTbook), sinon le couple $(-1, -1)$;
- il pourra être nécessaire de parcourir un ou les deux chemins en sens inverse avec la macro `reverse` appliquée à un chemin.

Intersection de courbes de Bézier

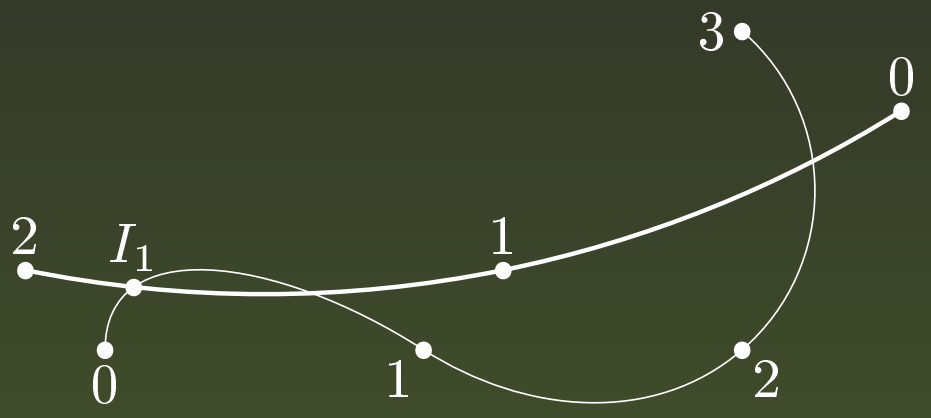


Intersection de courbes de Bézier



- Point I_1 :

Intersection de courbes de Bézier

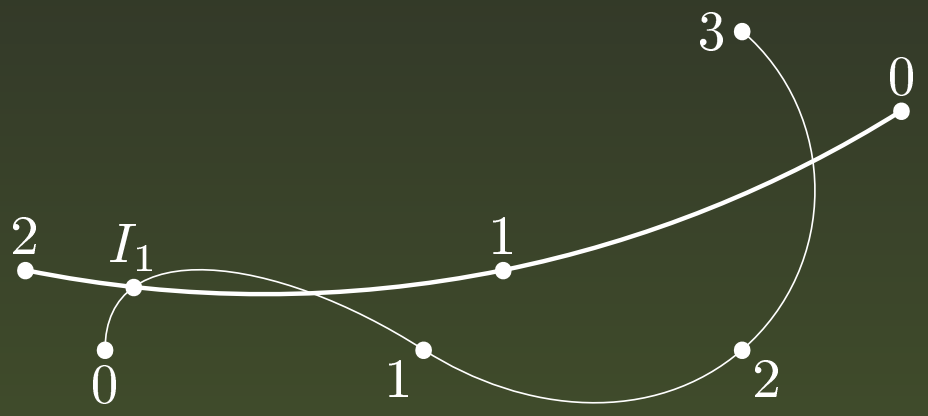


- Point I_1 :

  z1 = a intersectionpoint b ;

  dotlabel.top(btex I_1 etex, z1) ;

Intersection de courbes de Bézier



- Point I_1 :

  z1 = a intersectionpoint b ;

  dotlabel.top(btex I_1 etex, z1) ;
- METAPOST renvoie l'information (0.2501, 1.77225) à la commande `a intersectiontimes b`.

Encore des courbes de BÉZIER...

Encore des courbes de BÉZIER...

- `point`<*numérique*>`of`<*chemin*> renvoie les coordonnées du point du chemin à l'« instant » t considéré par *numérique*. $0 \leq t \leq n$, n correspondant au dernier point du chemin ;

Encore des courbes de BÉZIER...

- `point`<*numérique*>`of`<*chemin*> renvoie les coordonnées du point du chemin à l'« instant » t considéré par *numérique*. $0 \leq t \leq n$, n correspondant au dernier point du chemin ;
- `length`<*chemin*> renvoie la longueur du *chemin*, i.e. la valeur finale atteinte par t sur le *chemin* ;

Encore des courbes de BÉZIER...

- `point`<*numérique*>`of`<*chemin*> renvoie les coordonnées du point du chemin à l'« instant » t considéré par *numérique*. $0 \leq t \leq n$, n correspondant au dernier point du chemin ;
- `length`<*chemin*> renvoie la longueur du *chemin*, i.e. la valeur finale atteinte par t sur le *chemin* ;
- `subpath`(t_1, t_2)`of`<*chemin p*> définit le sous-chemin entre les valeurs t_1 et t_2. Si $t_1 > t_2$, le sous-chemin suit *p* en sens opposé ;

Encore des courbes de BÉZIER...

- `point`<*numérique*>`of`<*chemin*> renvoie les coordonnées du point du chemin à l'« instant » t considéré par *numérique*. $0 \leq t \leq n$, n correspondant au dernier point du chemin ;
- `length`<*chemin*> renvoie la longueur du *chemin*, i.e. la valeur finale atteinte par t sur le *chemin* ;
- `subpath`(t_1, t_2)`of`<*chemin p*> définit le sous-chemin entre les valeurs t_1 et t_2. Si $t_1 > t_2$, le sous-chemin suit *p* en sens opposé ;
- *p* `cutbefore` *q* trace la portion du chemin *p* après le chemin *q* ;

Encore des courbes de BÉZIER...

- `point`<*numérique*>`of`<*chemin*> renvoie les coordonnées du point du chemin à l'« instant » t considéré par *numérique*. $0 \leq t \leq n$, n correspondant au dernier point du chemin ;
- `length`<*chemin*> renvoie la longueur du *chemin*, i.e. la valeur finale atteinte par t sur le *chemin* ;
- `subpath`(t_1, t_2)`of`<*chemin p*> définit le sous-chemin entre les valeurs t_1 et t_2. Si $t_1 > t_2$, le sous-chemin suit *p* en sens opposé ;
- *p* `cutbefore` *q* trace la portion du chemin *p* après le chemin *q* ;
- *p* `cutafter` *q* trace la portion du chemin *p* avant le chemin *q* ;

Toujours des courbes de BÉZIER...

Toujours des courbes de BÉZIER...

- `direction`<*numérique*>`of`<*chemin*> renvoie le vecteur tangente au *chemin* au temps donné par *numérique* ;

Toujours des courbes de BÉZIER...

- `direction`<*numérique*>`of`<*chemin*> renvoie le vecteur tangente au *chemin* au temps donné par *numérique* ;
- `directiontime`<*paire*>`of`<*chemin*> renvoie le temps t quand le *chemin* a la direction donnée ;

Toujours des courbes de BÉZIER...

- `direction`<*numérique*>`of`<*chemin*> renvoie le vecteur tangente au *chemin* au temps donné par *numérique* ;
- `directiontime`<*paire*>`of`<*chemin*> renvoie le temps t quand le *chemin* a la direction donnée ;
- `directionpoint`<*paire*>`of`<*chemin*> trouve le premier point de la direction donnée ;

Toujours des courbes de BÉZIER...

- `direction`<*numérique*>`of`<*chemin*> renvoie le vecteur tangente au *chemin* au temps donné par *numérique* ;
- `directiontime`<*paire*>`of`<*chemin*> renvoie le temps t quand le *chemin* a la direction donnée ;
- `directionpoint`<*paire*>`of`<*chemin*> trouve le premier point de la direction donnée ;
- `arclength`<*chemin*> donne la longueur du *chemin* ;

Toujours des courbes de BÉZIER...

- `direction`<*numérique*>`of`<*chemin*> renvoie le vecteur tangente au *chemin* au temps donné par *numérique* ;
- `directiontime`<*paire*>`of`<*chemin*> renvoie le temps t quand le *chemin* a la direction donnée ;
- `directionpoint`<*paire*>`of`<*chemin*> trouve le premier point de la direction donnée ;
- `arclength`<*chemin*> donne la longueur du *chemin* ;
- `arctime` a `of` p donne la valeur de t telle que `arclength` `subpath` $(0, t)$ `of` $p = a$ (avec a entier).

Transformations

Transformations

- Déclaration `transform` T`;`

Transformations

- Déclaration `transform` T;
- $q = p$ `transformed` T;

Transformations

- Déclaration `transform` T;
- $q = p$ `transformed` T;
- $(q_x,\ q_y) = (t_x + t_{xx}p_x + t_{xy}p_y,\ t_y + t_{yx}p_x + t_{yy}p_y)$;

Transformations

- Déclaration `transform` T;
- $q = p$ `transformed` T;
- $(q_x,\ q_y) = (t_x + t_{xx}p_x + t_{xy}p_y,\ t_y + t_{yx}p_x + t_{yy}p_y)$;
- $T \overset{\text{déf}}{=} (t_x,\ t_y,\ t_{xx},\ t_{xy},\ t_{yx},\ t_{yy})$

Transformations

- Déclaration `transform` T;
- $q = p$ `transformed` T;
- $(q_x,\ q_y) = (t_x + t_{xx}p_x + t_{xy}p_y,\ t_y + t_{yx}p_x + t_{yy}p_y)$;
- $T \overset{\text{déf}}{=} (t_x,\ t_y,\ t_{xx},\ t_{xy},\ t_{yx},\ t_{yy})$
- s'appliquent aux couples, chemins, figures, stylos et aux transformations.

Transformations

7 transformations requérant un argument numérique ou paire :

$$
\begin{aligned}
(x,\ y)\,\texttt{shifted}\,(a,\ b) &= (x+a,\ y+b);\\
(x,\ y)\,\texttt{rotated}\,\theta &= (x\cos\theta - y\sin\theta,\ x\sin\theta + y\cos\theta);\\
(x,\ y)\,\texttt{slanted}\,a &= (x+ay,\ y);\\
(x,\ y)\,\texttt{scaled}\,a &= (ax,\ ay);\\
(x,\ y)\,\texttt{xscaled}\,a &= (ax,\ y);\\
(x,\ y)\,\texttt{yscaled}\,a &= (x,\ ay);\\
(x,\ y)\,\texttt{zscaled}\,(a,\ b) &= (ax-by,\ bx+ay).
\end{aligned}
$$

Transformations

Ses propres transformations :

Transformations

Ses propres transformations :

- $T =$ <expression transformation> ;

Transformations

Ses propres transformations :

- $T =$ <expression transformation> ;
- $T =$ identity xscaled -1 rotated 90 shifted (-1, 1) ;

Transformations

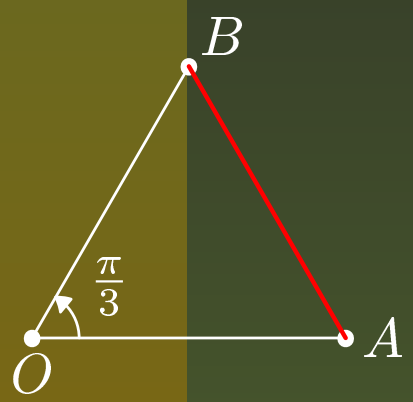
Ses propres transformations :

- $T =$ <expression transformation> ;
- $T =$ identity xscaled -1 rotated 90 shifted (-1, 1) ;
- $(0,\ 1) =$ transformed $T = (3,\ 4)$

  $(1,\ 1) =$ transformed $T = (7,\ 1)$

  $(1,\ 0) =$ transformed $T = (4,\ -3)$

Un exemple



pair A, B, C, O, D, E, F ;

numeric u ; path $trait$;

u = 1cm ; $O = (0,\ 0)$; dotlabel.bot(btex \$O\$ etex, O) ;

$A = (2u,\ 0)$; dotlabel.rt(btex \$A\$ etex, A) ;

$B = A$ rotated 60 ; dotlabel.urt(btex \$B\$ etex, B) ;

draw A -- O -- B ;

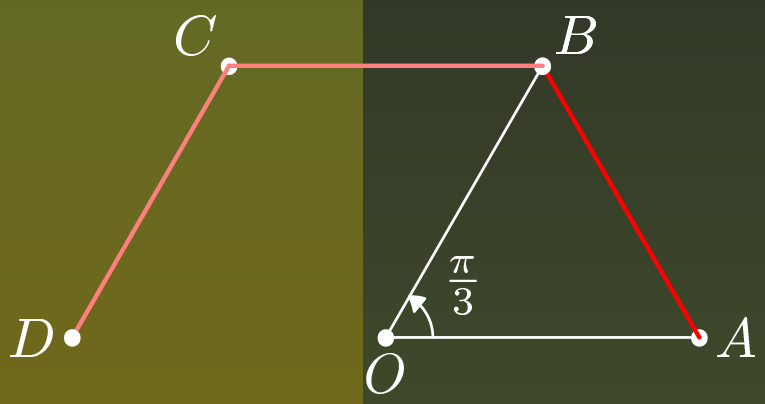
drawarrow quartercircle scaled $0.6u$ cutafter (O -- B) ;

label.urt(btex \$ \frac{\pi}{3}\$ etex, $0.3u$*dir 15) ;

$trait = A$ -- B ;

draw $trait$ withpen pencircle scaled 0.8bp withcolor red ;

Un exemple, la suite



$C = B$ rotated 60 ; dotlabel.ulft(btex \$C\$ etex, C) ;

$trait := (trait$ rotated 60) -- ($trait$ rotated 120) ;

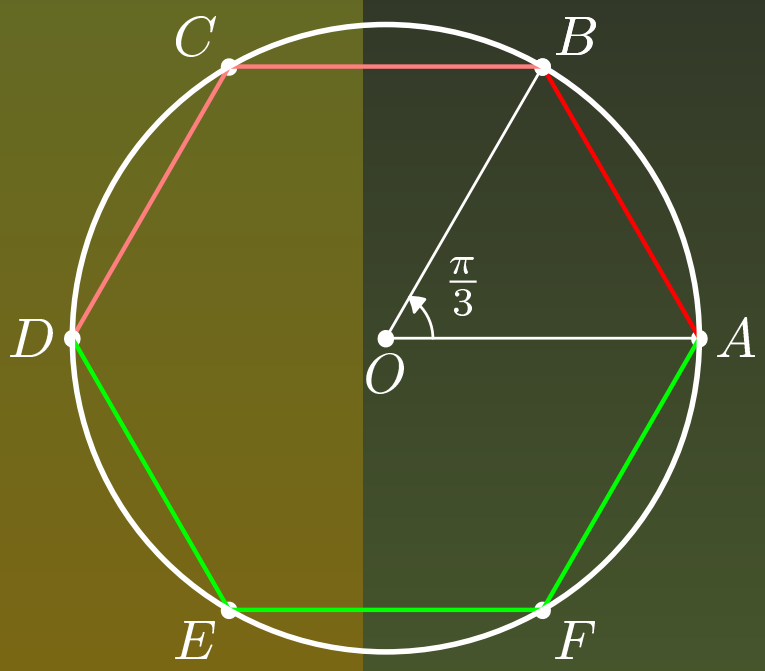
draw $trait$ withpen pencircle scaled 0.8bp

withcolor 0.5[white, red] ;

$D = A$ rotated 180 ;

dotlabel.lft(btex \$D\$ etex, A rotated 180) ;

Un exemple, la fin



dotlabel.llft(btex \$E\$ etex, B rotatedaround (O, 180)) ;

dotlabel.lrt(btex \$F\$ etex, B reflectedabout (A, D)) ;

$trait := (A$ -- $B)$ -- $trait$;

draw ($trait$ reflectedabout (A, D)) withpen pencircle scaled 0.8bp withcolor green ;

draw fullcircle scaled abs($A - D$) withpen pencircle scaled 1bp ;

Les figures

ou tout ce que METAPOST sait dessiner...

Les figures

ou tout ce que METAPOST sait dessiner...

- le résultat de `draw` ou `fill` est stocké dans une variable de type figure : `currentpicture;`

Les figures

ou tout ce que METAPOST sait dessiner...

- le résultat de `draw` ou `fill` est stocké dans une variable de type figure : `currentpicture` ;
- déclaration *picture mafigure* ;

Les figures

ou tout ce que METAPOST sait dessiner...

- le résultat de `draw` ou `fill` est stocké dans une variable de type figure : `currentpicture` ;
- déclaration *picture mafigure* ;
- `thelabel` permet de considérer une étiquette comme une figure ;

Les figures

ou tout ce que METAPOST sait dessiner...

- le résultat de `draw` ou `fill` est stocké dans une variable de type figure : `currentpicture` ;
- déclaration *picture mafigure* ;
- `thelabel` permet de considérer une étiquette comme une figure ;
- la macro `draw` permet de dessiner une figure en précisant la couleur du stylo ;

Les figures

ou tout ce que METAPOST sait dessiner...

- le résultat de `draw` ou `fill` est stocké dans une variable de type figure : `currentpicture` ;
- déclaration *picture mafigure* ;
- `thelabel` permet de considérer une étiquette comme une figure ;
- la macro `draw` permet de dessiner une figure en précisant la couleur du stylo ;
- la macro `fill` permet de remplir une figure en précisant la couleur du stylo ;

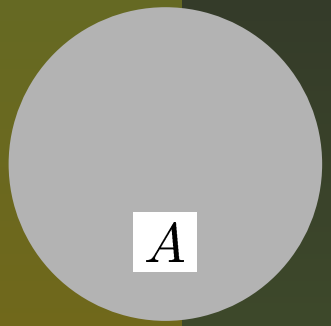
Les figures

ou tout ce que METAPOST sait dessiner...

- le résultat de `draw` ou `fill` est stocké dans une variable de type figure : `currentpicture` ;
- déclaration *picture mafigure* ;
- `thelabel` permet de considérer une étiquette comme une figure ;
- la macro `draw` permet de dessiner une figure en précisant la couleur du stylo ;
- la macro `fill` permet de remplir une figure en précisant la couleur du stylo ;
- `clip`<*figure*>`to`<*chemin (fermé)*> permet de découper une figure suivant un chemin donné.

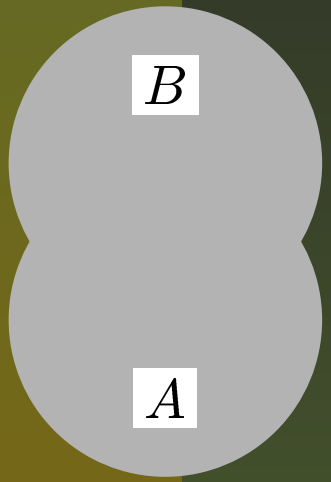
Un exemple

Un exemple




```
path a, b, aa, ab ;
picture pa, pb, u ;
a = fullcircle scaled 2cm ;
fill a withcolor .7white ;
pa = thelabel(btex $A$ etex, (0,-.5cm)) ;
unfill bbox pa ; draw pa ;
```

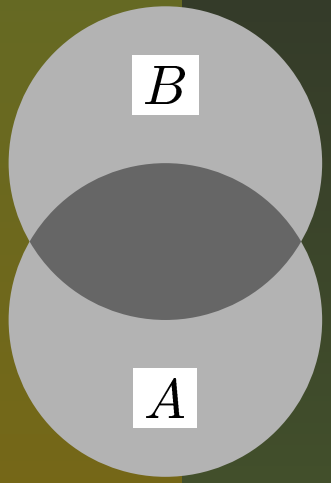
Un exemple




```
path a, b, aa, ab ;
picture pa, pb, u ;
a = fullcircle scaled 2cm ;
fill a withcolor .7white ;
pa = thelabel(btex $A$ etex, (0,-.5cm)) ;
unfill bbox pa ; draw pa ;
```

```
b = a shifted (0,1cm) ;
fill b withcolor .7white ;
pb = thelabel(btex $B$ etex, (0, 1.5cm)) ;
unfill bbox pb ; draw pb ;
```

Un exemple

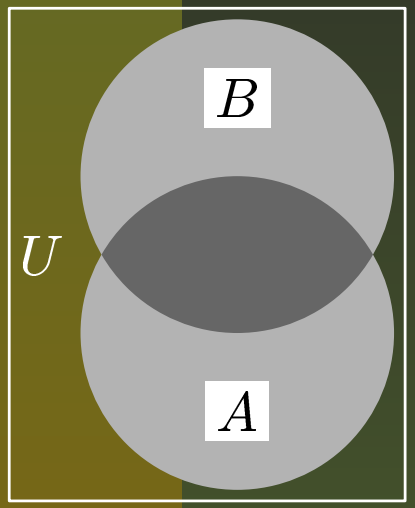



```
path a, b, aa, ab ;
picture pa, pb, u ;
a = fullcircle scaled 2cm ;
fill a withcolor .7white ;
pa = thelabel(btex $A$ etex, (0,-.5cm)) ;
unfill bbox pa ; draw pa ;
```

```
b = a shifted (0,1cm) ;
fill b withcolor .7white ;
pb = thelabel(btex $B$ etex, (0, 1.5cm)) ;
unfill bbox pb ; draw pb ;

ab = buildcycle(a, b) ;
fill ab withcolor .4white ;
```

Un exemple




```
path a, b, aa, ab ;
picture pa, pb, u ;
a = fullcircle scaled 2cm ;
fill a withcolor .7white ;
pa = thelabel(btex $A$ etex, (0,-.5cm)) ;
unfill bbox pa ; draw pa ;
```

```
b = a shifted (0,1cm) ;
fill b withcolor .7white ;
pb = thelabel(btex $B$ etex, (0, 1.5cm)) ;
unfill bbox pb ; draw pb ;

ab = buildcycle(a, b) ;
fill ab withcolor .4white ;

u = thelabel.lft(btex
$U$ etex, (-1cm, .5cm)) ;
draw u withcolor white ;
draw bbox currentpicture withcolor white ;
```

Un autre exemple

Un autre exemple

```
path p[] ;
p1 = (0, 0) .. (5pt, -3pt) .. (10pt, 0) ;
```

Un autre exemple

```
path p[] ;
p1 = (0, 0) .. (5pt, -3pt) .. (10pt, 0) ;
p2 = p1 .. (p1 yscaled -1 shifted
(10pt, 0)) ;
```

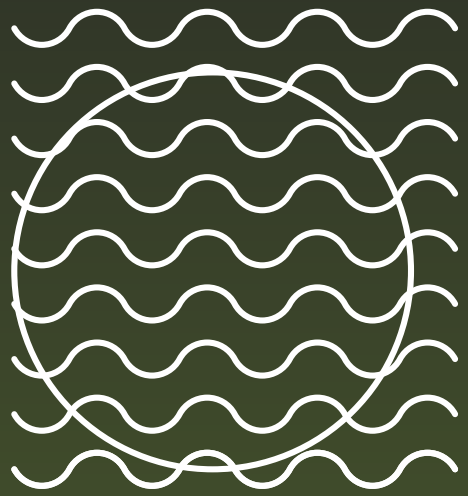
Un autre exemple

```
path p[] ;
p1 = (0, 0) .. (5pt, -3pt) .. (10pt, 0) ;
p2 = p1 .. (p1 yscaled -1 shifted
(10pt, 0)) ;
p0 = p2 ;
for i=1 upto 3 :
p0 := p0 .. p2 shifted (i*20pt,0) ;
endfor
```

Un autre exemple

```
path p[] ;
p1 = (0, 0) .. (5pt, -3pt) .. (10pt, 0) ;
p2 = p1 .. (p1 yscaled -1 shifted
(10pt, 0)) ;
p0 = p2 ;
for i=1 upto 3 :
p0 := p0 .. p2 shifted (i*20pt,0) ;
endfor
for j = 0 upto 8 :
draw p0 shifted (0, j*10pt) ;
endfor
```

Un autre exemple



```
path p[] ;
p1 = (0, 0) .. (5pt, -3pt) .. (10pt, 0) ;
p2 = p1 .. (p1 yscaled -1 shifted
(10pt, 0)) ;
p0 = p2 ;
for i=1 upto 3 :
p0 := p0 .. p2 shifted (i*20pt,0) ;
endfor
for j = 0 upto 8 :
draw p0 shifted (0, j*10pt) ;
endfor
p3 = fullcircle shifted (.5, .5) scaled 72pt ;
draw p3 ;
```

Un autre exemple

```
path p[] ;
p1 = (0, 0) .. (5pt, -3pt) .. (10pt, 0) ;
p2 = p1 .. (p1 yscaled -1 shifted
(10pt, 0)) ;
p0 = p2 ;
for i=1 upto 3 :
p0 := p0 .. p2 shifted (i*20pt,0) ;
endfor
for j = 0 upto 8 :
draw p0 shifted (0, j*10pt) ;
endfor
p3 = fullcircle shifted (.5, .5) scaled 72pt ;
draw p3 ;
clip currentpicture to p3 ;
```

Les macros

Les macros

Elles proviennent de différents *packages*, certains sont livrés avec METAPOST (†) sous forme de fichiers *.mp :

Les macros

Elles proviennent de différents *packages*, certains sont livrés avec METAPOST (†) sous forme de fichiers *.mp :

- plain† (plain.mp) pour la plupart des macros vues précédemment. Il est chargé automatiquement ;

Les macros

Elles proviennent de différents *packages*, certains sont livrés avec METAPOST (†) sous forme de fichiers *.mp :

- plain† (plain.mp) pour la plupart des macros vues précédemment. Il est chargé automatiquement ;
- mfplain† (mfplain.mp) qui s'inspire de METAFONT ;

Les macros

Elles proviennent de différents *packages*, certains sont livrés avec METAPOST (†) sous forme de fichiers *.mp :

- plain† (plain.mp) pour la plupart des macros vues précédemment. Il est chargé automatiquement ;
- mfplain† (mfplain.mp) qui s'inspire de METAFONT ;
- string† (string.mp) permet de manipuler les chaînes de caractères ;

Les macros

Elles proviennent de différents *packages*, certains sont livrés avec METAPOST (†) sous forme de fichiers *.mp :

- plain† (plain.mp) pour la plupart des macros vues précédemment. Il est chargé automatiquement ;
- mfplain† (mfplain.mp) qui s'inspire de METAFONT ;
- string† (string.mp) permet de manipuler les chaînes de caractères ;
- format† (format.mp) : macros pour composer de nombres ;

Les macros

Elles proviennent de différents *packages*, certains sont livrés avec METAPOST (†) sous forme de fichiers *.mp :

- plain† (plain.mp) pour la plupart des macros vues précédemment. Il est chargé automatiquement ;
- mfplain† (mfplain.mp) qui s'inspire de METAFONT ;
- string† (string.mp) permet de manipuler les chaînes de caractères ;
- format† (format.mp) : macros pour composer de nombres ;
- marith† (marith.mp) : pour manipuler de très grandes valeurs (jusqu'à $3,8877 \times 10^{55}$) et de très petites (jusqu'à $1,604 \times 10^{-25}$) ;

Les macros

- sarith† (sarith.mp) pour l’arithmétique des chaînes de caractères représentant des grands nombres ;

Les macros

- sarith† (sarith.mp) pour l'arithmétique des chaînes de caractères représentant des grands nombres ;
- boxes† (boxes.mp) pour faire des boîtes rectangulaires ou ovales ;

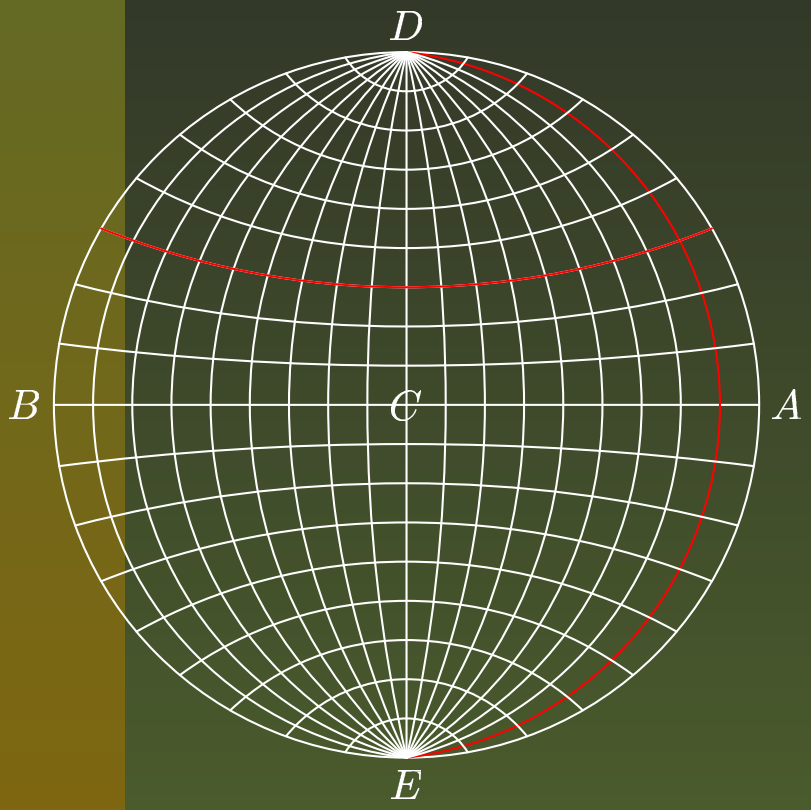
Les macros

- sarith† (sarith.mp) pour l'arithmétique des chaînes de caractères représentant des grands nombres ;
- boxes† (boxes.mp) pour faire des boîtes rectangulaires ou ovales ;
- graph† (graph.mp) qui permet de faire des graphes et que J. HOBBY a documenté ;

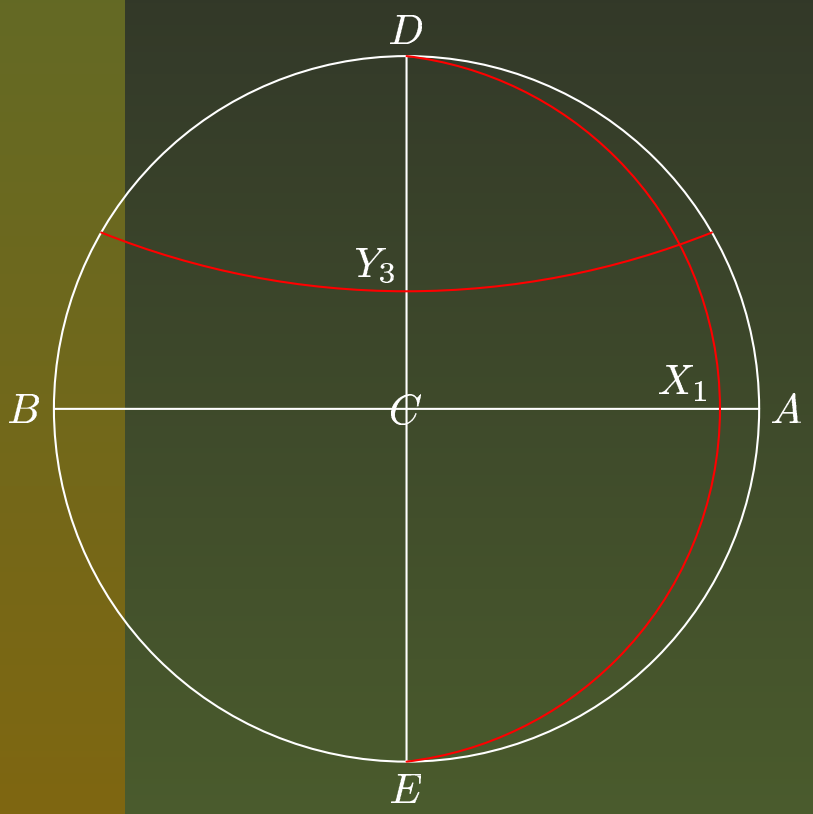
Les macros

- sarith† (sarith.mp) pour l'arithmétique des chaînes de caractères représentant des grands nombres ;
- boxes† (boxes.mp) pour faire des boîtes rectangulaires ou ovales ;
- graph† (graph.mp) qui permet de faire des graphes et que J. HOBBY a documenté ;
- mesmacros (mesmacros.mp) qui regroupera les macros que l'on fabriquera soi-même !

Un exemple de macro

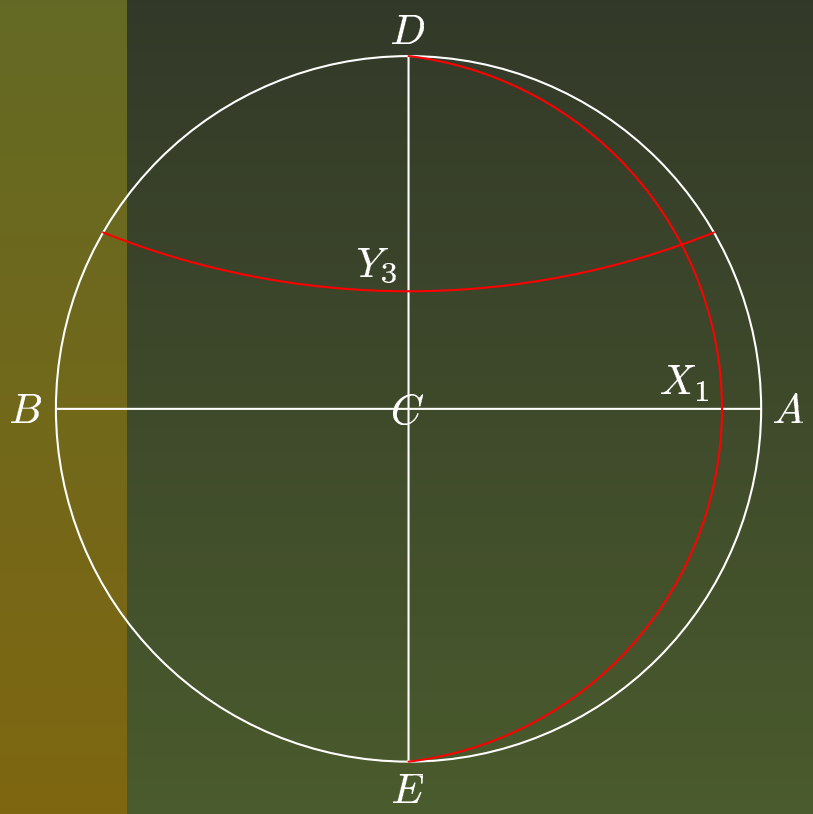


Un exemple de macro




```
vardef centre (expr A, B, C, Ce) =
  Ce - 0.5[A,B] = whatever*(A - B)
  rotated 90 ;
  Ce - 0.5[C,B] = whatever*(C - B)
  rotated 90 ;
enddef ;
```

Un exemple de macro




```
vardef centre (expr A, B, C, Ce) =
Ce − 0.5[A, B] = whatever*(A − B)
rotated 90 ;
Ce − 0.5[C, B] = whatever*(C − B)
rotated 90 ;
enddef ;

X1 = (1/18)[A, B] ;
centre(D, X1, E, I1) ;
pp1 = halfcircle scaled 2abs(I1 − D)
rotated -90 shifted I1 cutbefore p0
cutafter p0 ;
draw pp1 withcolor red ;
```

Boucles et structures de contrôle

Boucles et structures de contrôle

- `for` <*tokens symboliques*> = <*expression*> `upto` <*expression*> :
  <*texte de boucle*> `endfor`
  où `upto` ⇔ `step 1 until`

Boucles et structures de contrôle

- `for` <*tokens symboliques*> = <*expression*> `upto` <*expression*> :
  <*texte de boucle*> `endfor`
  où `upto` ⇔ `step 1 until`
  il existe également `downto` pour `step -1 until`

Boucles et structures de contrôle

- `for` <*tokens symboliques*> = <*expression*> `upto` <*expression*> :
  <*texte de boucle*> `endfor`
  où `upto` ⇔ `step 1 until`
  il existe également `downto` pour `step -1 until`
- `forever` : <*texte de boucle*> `endfor`

Boucles et structures de contrôle

- `for` <*tokens symboliques*> = <*expression*> `upto` <*expression*> :
  <*texte de boucle*> `endfor`
  où `upto` ⇔ `step 1 until`
  il existe également `downto` pour `step -1 until`
- `forever` : <*texte de boucle*> `endfor`
- pour sortir de la boucle *forever* on rajoute une clause de sortie

Boucles et structures de contrôle

- `for` <*tokens symboliques*> = <*expression*> `upto` <*expression*> :
  <*texte de boucle*> `endfor`
  où `upto` ⇔ `step 1 until`
  il existe également `downto` pour `step -1 until`
- `forever` : <*texte de boucle*> `endfor`
- pour sortir de la boucle *forever* on rajoute une clause de sortie
  - `exitif` <*expression booléene*> (vraie) ;

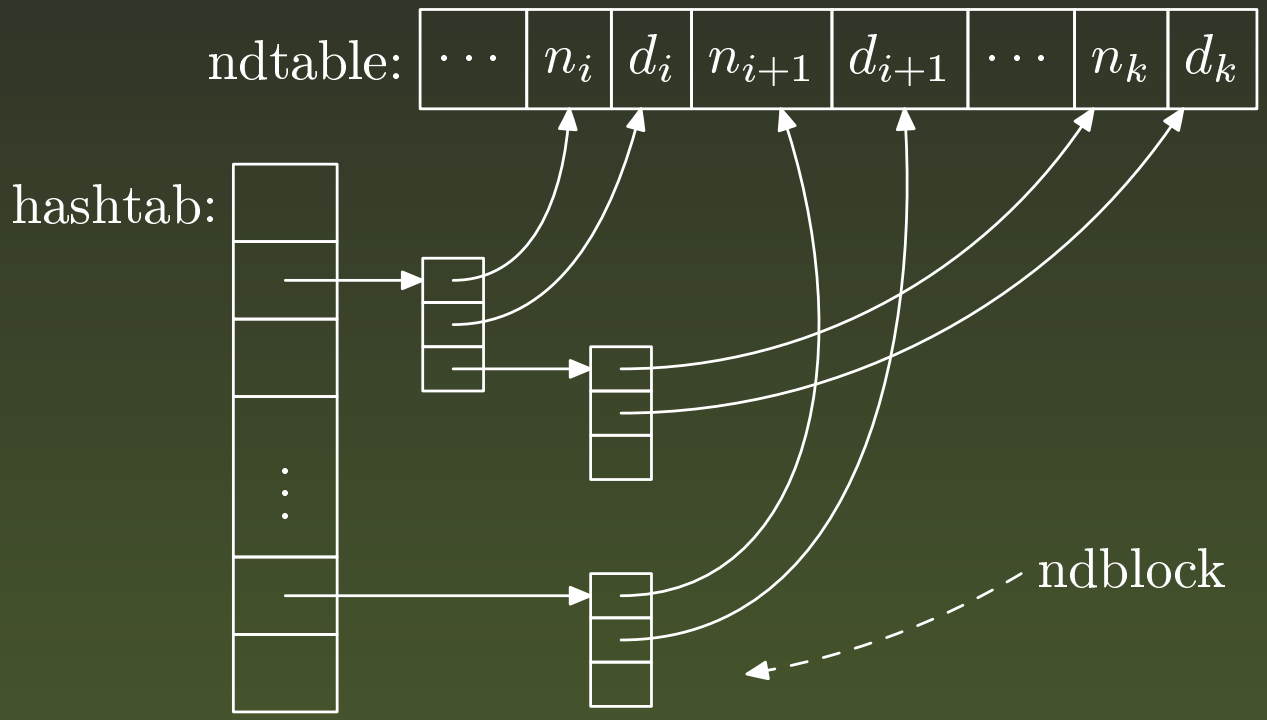
Boucles et structures de contrôle

- `for` <*tokens symboliques*> = <*expression*> `upto` <*expression*> :
  <*texte de boucle*> `endfor`
  où `upto` ⇔ `step 1 until`
  il existe également `downto` pour `step -1 until`
- `forever` : <*texte de boucle*> `endfor`
- pour sortir de la boucle *forever* on rajoute une clause de sortie
  - `exitif` <*expression booléene*> (vraie) ;
  - `exitunless` <*expression booléene*> (fausse) ;

Boucles et structures de contrôle

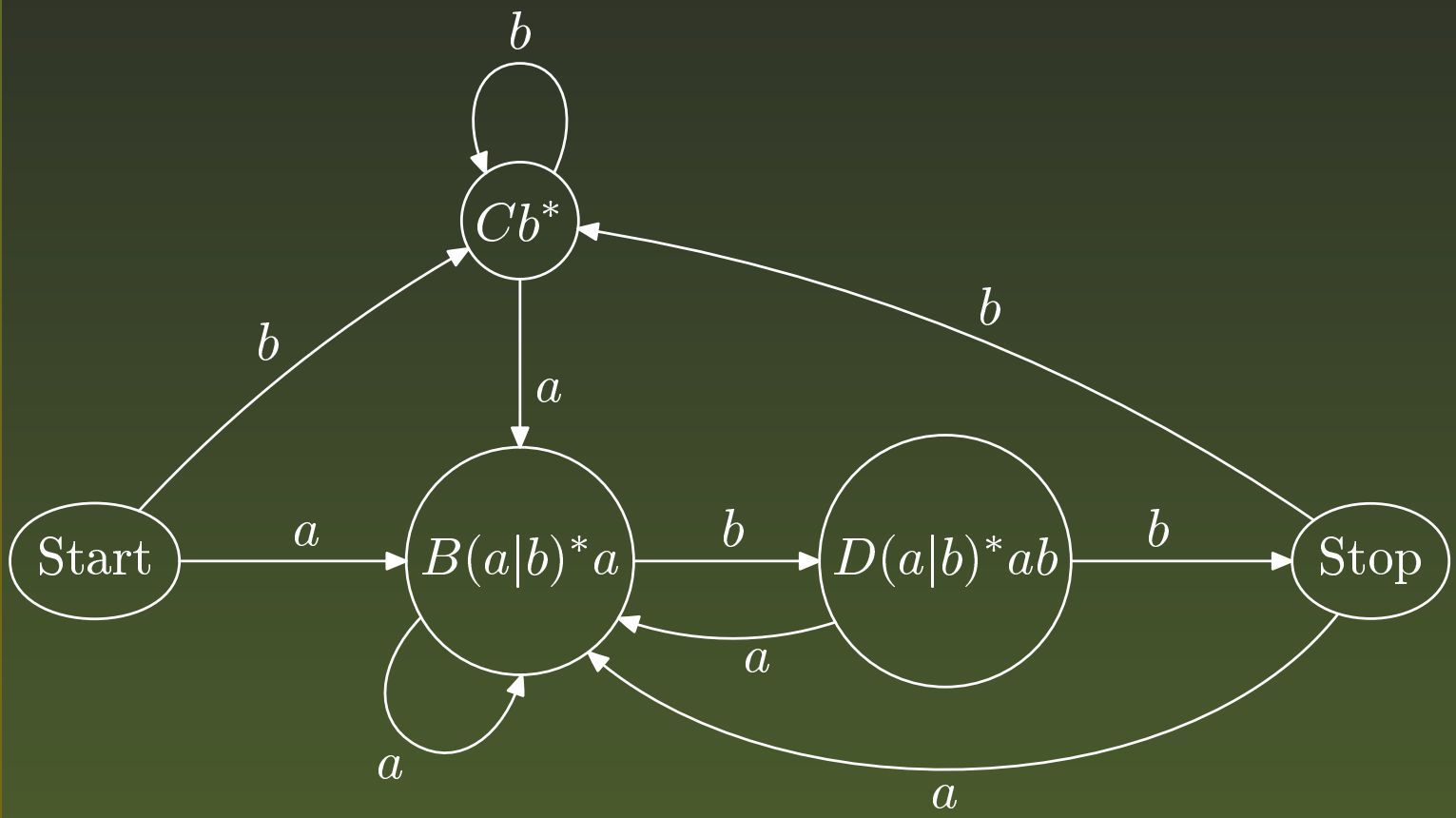
- `for` <*tokens symboliques*> = <*expression*> `upto` <*expression*> :
  <*texte de boucle*> `endfor`
  où `upto` ⇔ `step 1 until`
  il existe également `downto` pour `step -1 until`
- `forever` : <*texte de boucle*> `endfor`
- pour sortir de la boucle *forever* on rajoute une clause de sortie
  - `exitif` <*expression booléene*> (vraie) ;
  - `exitunless` <*expression booléene*> (fausse) ;
  - la boucle *while* en METAPOST s'écrira :
    `forever` : `exitunless` <*expression booléene*> ; <*texte de boucle*> `endfor`

Quelques petites choses



À l'aide du *package* boxes.mp

Quelques petites choses



À l'aide du *package* boxes.mp

Pour aller de l'avant

- Le manuel de l'utilisateur (en anglais ou en français) ;
- Une introduction à METAPOSTde Laurent CHÉNO `http://pauillac.inria.fr/~cheno/metapost/` ;
- Le site de Zoonekind `http://www.math.jussieu.fr/~zonnek/LaTeX/Metapost` ;
- Le site du Loria : `http://tex.loria.fr` dans la rubrique graphics ;
- `http://melusine.eu.org/syracuse/metapost`
- METAPOST comme mot-clé de Google