CAHIERS GUTenberg

MANUEL DE PRISE EN MAIN POUR TIKZ

Yves SOULET

Cahiers GUTenberg, nº 50 (2008), p. 5-87.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_2008___50_5_0>

MANUEL DE PRISE EN MAIN POUR TIKZ

Yves SOULET

RÉSUMÉ. — Cet article est un manuel concis et original permettant de prendre en main le système Ti*k*Z de Till Tantau. Une attention particulière est portée aux applications concrètes.

ABSTRACT. — This is a concise manual for getting aquainted with the Ti*k*Z drawing system by Till Tantau. A special attention is given toward applications from the real world.

INTRODUCTION

Si l'on utilse beaucoup PSTricks, pour les figures mais aussi pour de très nombreux éléments de composition tels que numérotations sur des onglets, flèches adaptées à la longueur et à l'inclinaison des lettres, etc., on peut se poser la question suivante : est-ce que PDFTricks, avec sa multitude de fichiers PostScript intermédiaires et leurs transformés en PDF entre les deux passes, est la bonne solution pour produire directement du PDF ?

Le *Cahier GUTenberg* numéro 48 (avril 2007) contient un tutoriel Ti*k*Z, écrit par Till Tantau et traduit par Yvon Henel, fort agréable à lire et donnant envie d'utiliser l'extension `tikz` de LaTeX en allant plus loin. Il semble bien que cette extension apporte une réponse alternative à la question posée. Mais alors, « aller plus loin » signifie se plonger dans plusieurs centaines de pages de la documentation de référence très touffue et dans laquelle on a parfois des difficultés pour rassembler ce qui est utilisable et passer rapidement sur ce qui est superflu ou qui constitue des explications sur le fonctionnement interne des commandes et des options de l'extension.

Cahiers GUTenberg n° 50 — Avril 2008, p. 5-87.

Ces quelques pages ont le modeste objectif d'aider à avancer très vite dans l'utilisation de l'extension `tikz`. Elles n'ont pas le style, le naturel et l'aisance du tutoriel cité mais elles contiennent l'essentiel pour l'utilisation de Ti*k*Z et le nécessaire pour s'initier rapidement aux classes de diagrammes non abordées telles que les réseaux de Petri par exemple. Seules les applications courantes et celles se prêtant bien à l'adaptation à d'autres applications sont présentées. De toute manière, l'extension `tikz`, dans son état actuel, ne développe que certaines applications ; ce sont les spécialistes de chaque discipline qui sont à même de développer ces applications (on se rend aisément compte que l'on pourrait écrire une bibliothèque supplémentaire pour les diagrammes de Feynman, il y a tous les éléments de base nécessaires).

On s'est imposé de faire suivre chaque nouvelle commande et chaque nouvelle option d'un exemple où le code se trouve juste à gauche de la figure. Compte tenu de l'étroitesse de l'espace disponible au format de la revue, il a fallu parfois déroger à cette règle : quelques mots simples (sur l'execice précédent, voici un exemple qui sera étudié en page suivante, etc.) permettent au lecteur de suivre le fil de la logique de la progression.

Des petites macros sont introduites pour diminuer la longueur de la saisie. On considère que le lecteur est un utilisateur de LaTeX ayant quelques petites notions sur les extensions `xcolor` et `calc`.

Les références sont de deux types : [GUT x] et [GUT x.x] réfèrent aux sections et sous-sections du tutoriel publié dans les *Cahiers*, [x], [x.x] et [x.x.x] réfèrent aux sections, sous-sections et paragraphes de la documentation de référence qui est accessible sur le site de l'institut de Till Tantau.

L'index contient toutes les commandes et options et mentionne le numéro de page de leur définition s'il y a lieu, de leur première utilisation ; éventuellement, il donne aussi les numéros de page contenant une nouvelle utilisation spécifique de ces commandes et options. Il ne reprend pas les titres de chapitre, section et sous-section qui se trouvent dans la table des matières détaillée ; par contre, il signale certaines spécificités ne faisant l'objet ni d'une commande ou option ni d'un titre de sectionnement, par exemple : difficulté rencontrée avec le caractère deux-points rendu actif.

Certaines traductions méritent des explications :

— *node* relatif a une liaison inter-nœud et *label* ont été traduits

par « label », souvent utilisé dans certaines disciplines, notamment en informatique ;

— *node*, dans le sens usuel de lettrage (lettres ou autres caractères souvent mathématiques rajoutés sur les figures), a été traduit par « lettrage » ;

— *pin* a été conservé (il est utilisé par les adolescents depuis des années et il représente bien un graphisme spécifique).

— *mindmap* a été également conservé (en réalité ce mot ne figure pas dans un des dictionnaires servant de référence pour les traductions).

Certaines dénominations sont un peu anciennes, lignes brisées et lignes courbes par exemple ; elles ont été choisies car elles traduisent bien le fait que les commandes de tracé sont différentes ; on pourra peut-être en remarquer quelques autres.

Pour avoir des petites figures face au code correspondant, on a toujours utilisé des petites dimensions et des petites tailles de caractère ; pour les arbres par exemple, on s'est limité dans les exemples à une ou deux générations de descendants. Pour faciliter l'impression, la couleur n'est presque pas utilisée : les différenciations nécessaires du point de vue pédagogique sont assurées par l'utilisation de différents niveaux de gris. Il n'y a que trois pages (19, 20, 62 et 81) où l'utilisation de la couleur a été indispensable. Pour apprécier les positionnements (alignements en particulier), il est recommandé d'utiliser la loupe du visualisateur (Gsview, Ghostview, etc.).

Enfin, il faut avertir le lecteur que l'extension `tikz` est en fait une couche de code permettant l'utilisation du langage PGF utilisé par d'autres styles ou classes, `beamer` par exemple. Ce langage comprend lui-même deux couches : la couche dite « de base » et la couche plus profonde dite « système » (respectivement parties VI et VII de la documentation de référence). Il est parfois nécessaire d'utiliser une commande de la couche de base ; cela a été fait parfois (par exemple diminuer la dimension des glyphes représentant les points utilisés pour certains tracés de courbe) ; le nom de ces commandes commence toujours par `\pgf...` Le lecteur de ce petit manuel n'a pas besoin de consulter la partie correspondante de la documentation de référence, sauf par curiosité ou par désir d'approfondissement.

Table des matières

CHAPITRE 1

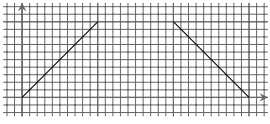
Systèmes de coordonnées

Dans ce premier (et tout petit) chapitre, on va introduire les notations pour la représentation d'un point par ses coordonnées. Les points seront les extrémités des segments tracés avec la commande `\draw` et la spécification `--` qui seront vues dans le prochain chapitre. Une option de cette commande, `xshift`, a été utilisée pour translater l'origine afin de bien séparer les différentes parties de l'exemple. Une grille d'un pas de 1 millimètre et des axes (ne figurant pas sur les listings des exemples), permettent de visualiser plus facilement l'effet des commandes. On ajoute
`\usepackage{tikz}`
dans le préambule et on est prêt à faire le premier pas !

1. Coordonnées cartésiennes

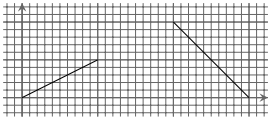
En cartésiennes, le point de coordonnées $x = a$ cm et $y = b$ cm est noté `(a cm,b cm)` ou, puisque le `cm` est l'unité par défaut, `(a,b)`. Mais on peut aussi utiliser le `mm` ou le `pt` [8.1], [10.2.1].

```
\begin{tikzpicture}
\draw(0,0)--(1,1);
\draw(2cm,28.5pt)--(30mm,0pt);
\end{tikzpicture}
```



On peut aussi changer les unités par défaut en ajoutant une option
`\begin{tikzpicture}[x=10mm,y=5mm]`
et utiliser la notation sans unité. Cependant, on peut encore utiliser des unités et alors, ces unités prévalent sur les unités définies par l'option.

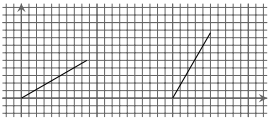
```
\begin{tikzpicture}[x=10mm,y=5mm]
\draw(0,0)--(1,2);
\draw(2,2)--(3cm,0cm);
\end{tikzpicture}
```



2. Coordonnées polaires

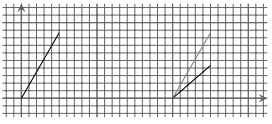
En polaires, le point de coordonnées $\rho = r$ cm et $\theta = w$ degrés est noté `(w:r cm)` ou `(w:r)` [10.2.1].

```
\begin{tikzpicture}
\draw(0,0)--(30:1cm);
\draw[xshift=2cm](0,0)--(60:1);
\end{tikzpicture}
```



La notation sans unité ne peut s'utiliser que si les unités suivant x et suivant y sont identiques. Sinon, on obtient un point d'abscice correcte mais dont l'ordonnée est réduite par le facteur (longueur de l'unité suivant les y)/(longueur de l'unité suivant les x).

```
\begin{tikzpicture}[x=10mm,y=5mm]
\draw(0,0)--(30:1cm);
\draw[xshift=2cm](0,0)--(60:1);
\draw[xshift=2cm,color=gray]
               (0,0)--(60:1cm);
\end{tikzpicture}
```



3. Déplacements et tracés relatifs

Après avoir vu comment un point est noté, on va voir maintenant un complément à cette notation [8.1], [10.3] :

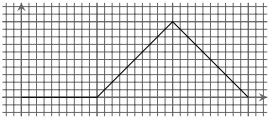
`+(a,b)` désigne le point déplacé de a suivant les x et de b suivant les y par rapport à la position précédente qui est conservée,

`++(a,b)` désigne le point déplacé de a suivant les x et de b suivant les y par rapport à la position précédente puis amène la position précédente au point déplacé : on trouvera ci-dessous un chemin noté simplement avec la notation initiale, puis avec deux variantes du code pour montrer le rôle des commandes + et ++.

Tout ceci est valable en cartésiennes mais aussi en polaires.

Il faut remarquer que ces commandes, + et ++, jouent des rôles semblables à la commande `rmoveto` du langage PostScript, elles sont particulièrement commodes car, avec la spécification `--` elles jouent le rôle de la commande `rlineto` (faire l'exercice qui suit).

```
\begin{tikzpicture}
\draw(0,0)--(1,0)--(2,1)--(3,0);
% ou \draw(0,0)--(1,0)--+(1,1)--(3,0);
% ou \draw(0,0)--(1,0)--++(1,1)--+(1,-1);
\end{tikzpicture}
```



CHAPITRE 2

Lignes brisées

Il a été choisi de consacrer un chapitre aux lignes brisées, ouvertes ou fermées, car la syntaxe correspondante, relativement simple, se prête bien à l'introduction des options dont la plupart se retrouveront dans la syntaxe correspondant aux lignes courbes. Dans tous les exemples qui vont être proposés, on utilisera le `mm` pour unité (ce qui permet d'éviter l'usage des nombres décimaux) et on ajoutera parfois une grille et des axes comme précédemment.

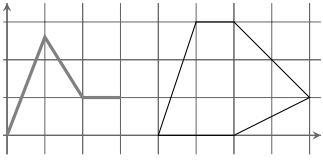
1. Syntaxe de base

Cette syntaxe de base est la commande [2.1] [12.2] :
`\path [options] (spécifications) ;`

Les options sont par exemple la commande de tracé [1], l'épaisseur du trait, la couleur du trait, etc. Les spécifications sont, pour les cas les plus simples, les coordonnées des extrémités des segments de la ligne brisée, le tracé des segments (`--`) et la fermeture (`--cycle`) s'il s'agit d'une ligne brisée fermée ; mais il y en a beaucoup d'autres.

Voici un exemple de ligne brisée ouverte et un exemple de ligne brisée fermée :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,very thick,gray](0,0)
       --(5,13)--(10,5)--(15,5);
\path[xshift=20mm,draw](0,0)--(5,15)
   --(10,15)--(20,5)--(10,0)--cycle;
\end{tikzpicture}
```



Les rôles des options se trouvant dans les exemples ci-dessus sont [8.3] :

1. On peut définir un chemin pour le remplir ou pour découper une partie de figure et cela sans vouloir le tracer.

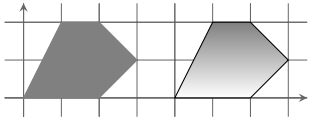
`draw` : trace la ligne spécifiée,
`very thick` : choisit l'épaisseur de la ligne (1.2 pt),
`gray` : choisit la couleur,
`xshift` : translate l'origine suivant les x.

2. Options principales

Les options principales pour une ligne sont [8.3] :
`draw` : trace la ligne spécifiée,
`fill` : colore l'intérieur de la ligne fermée spécifiée [12.3],
`shade` : ombre l'intérieur de la ligne fermée spécifiée [12.4],
`clip` : découpe l'intérieur de la ligne fermée spécifiée [12.16].

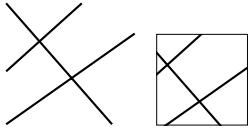
Voici des exemples des trois dernières options, la première ayant été utilisée au chapitre 1 sous sa forme abrégée (abréviation définie après les deux exemples suivants).

```
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,fill,gray](0,0)--(5,10)
     --(10,10)--(15,5)--(10,0)--cycle;
\path[xshift=20mm,draw,shade]
        (0,0)--(5,10)--(10,10)--(15,5)
                       --(10,0)--cycle;
\end{tikzpicture}
```



On note tout de suite que la commande `--cycle` trace le dernier segment alors que la commande `--(0,0)` fermerait aussi la ligne brisée mais imparfaitement sur le plan graphique. Cette imperfection n'est visible que pour les épaisseurs de trait assez grandes et ne pourra être comprise que plus loin.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,thick](-2,-2)--(15,10) (-2,5)--(8,14) (-2,14)--(12,-2);
\end{tikzpicture}
\hskip2mm
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,very thin](1,1)--(13,1)
         --(13,13)--(1,13)--cycle;
\path[clip](1,1)--(13,1)--(13,13)
                  --(1,13)--cycle;
\path[draw,thick](-2,-2)--(15,10) (-2,5)--(8,14) (-2,14)--(12,-2);
\end{tikzpicture}
```



Il faut encore noter que l'option `clip` ne découpe que les tracés qui suivent ; on peut tracer la ligne brisée de découpe mais sans option : la

commande `\path [draw,clip]` est acceptée mais on ne peut donner aucune option supplémentaire à l'option `draw` ; c'est parce que l'on voulait une ligne de découpe en trait très fin (`very thin`) que l'on a dû faire le tracé avant de faire la découpe ; une possibilité consiste à choisir l'option `very thin` comme option d'épaisseur courante.

On donne quatre abréviations qui seront continuellement utilisées [8.3] :
`\draw [options]` et `\fill [options]`
équivalent respectivement à
`\path [draw,options]` et `\path [fill,options]`
et
`\filldraw [options]` et `\shadedraw [options]`
équivalent respectivement à
`\path [draw,fill,options]` et `\path [draw,shade,options]`

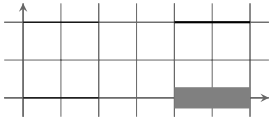
3. Options de tracé

3.1. Epaisseur de traits

L'option pour l'épaisseur des traits est (défaut : 0.4 pt) [12.2.1] :
`line width=`(dimension)

On dispose aussi de styles prédéfinis (défaut : `thin`) :
`style=ultra thin`, `very thin`, `thin`, `semithick`, `thick`, `very thick` et `ultra thick`
qui correspondent respectivement à 0.1 pt, 0.2 pt, 0.4 pt, 0.6 pt, 0.8 pt, 1.2 pt et 1.6pt.

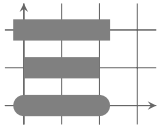
```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[thin](0,10)--+(10,0);
\draw(0,0)--+(10,0);
\draw[thick](20,10)--+(10,0);
\draw[line width=6pt,gray]
             (20,0)--+(10,0);
\end{tikzpicture}
```



3.2. Terminaison des traits

Elle est produite par l'otion (défaut : `butt`) [12.2.1] :
`cap=rect`, `butt` ou `round`
Les trois possibilités sont montrées sur l'exemple. L'option `butt` permet de revenir à la terminaison par défaut.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[line width=8pt,cap=rect,
         gray](0,10)--+(10,0);
\draw[line width=8pt,gray]
           (0,5)--+(10,0);
\draw[line width=8pt,cap=round,
           gray](0,0)--+(10,0);
\end{tikzpicture}
```

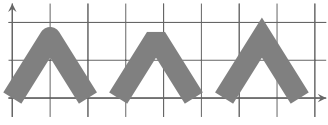


On constate que le premier et le dernier choix conduisent à un dépassement d'une demi épaisseur de trait. Il est évident que cette option joue un rôle important pour les grandes épaisseurs de trait.

3.3. Jonction des traits

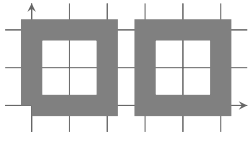
Elle est déterminée par l'option (défaut : `miter`) [12.2.1] :
`join=round`, `bevel` ou `miter`
Les trois possibilités sont montrées sur l'exemple. L'option par défaut permet de revenir à la terminaison par défaut.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[line width=8pt,join=round,
   gray](0,0)--++(5,8)--+(5,-8);
\draw[line width=8pt,join=bevel,
  gray](14,0)--++(5,8)--+(5,-8);
\draw [line width=8pt,gray]
  (28,0)--++(5,8)--+(5,-8);
\end{tikzpicture}
```



Il est encore évident que cette option joue un rôle important pour les grandes épaisseurs de trait. Et voilà enfin la raison pour laquelle il faut toujours fermer une ligne (brisée ou courbe) par `--cycle` [11.4] :

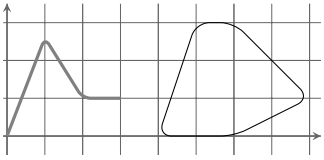
```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[line width=8pt,gray](0,0)
--(10,0)--(10,10)--(0,10)--(0,0);
\draw[line width=8pt,gray](15,0)
--(25,0)--(25,10)--(15,10)--cycle;
\end{tikzpicture}
```



3.4. Arrondissement des angles

Il est réalisé grâce à l'option (défaut : `sharp corners`) [11.6]
`rounded corners=`(dimension)
où cette dimension est le rayon de courbure souhaité. L'option par défaut pemet encore de revenir aux angles vifs si nécessaire.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw [very thick,gray,rounded corners
   =3pt](0,0)--(5,13)--(10,5)--(15,5);
\draw [xshift=20mm,rounded corners
   =5pt](0,0)--(5,15)--(10,15)--(20,5)
                      --(10,0)--cycle;
\end{tikzpicture}
```



3.5. Pointillés et traitillés

Les pointillés sont obtenus et abandonnés avec l'option (défaut : `solid`, pas de pointillés) [12.2.2] :

`style=densely dotted`, `dotted`, `loosely dotted` ou `solid`

Les traitillés sont obtenus et abandonnés avec l'option (défaut : `solid`, pas de traitillés) [12.2.2] :

`style=densely dashed`, `dashed`, `loosely dashed` ou `solid`

On peut aussi définir son propre motif de pointillé avec une commande telle que `dash pattern=on 2pt off 3pt on 3pt off 2pt` où le motif ainsi défini se répète indéfiniment, commme en langage Metafont ; on l'utilise avec le style `dashed`.

Il faut signaler encore qu'il est possible de décaler le motif au début du traitillé avec l'option :

`dash phase=`(dimension)

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[thick,loosely dotted](0,10)--+(10,0);
\draw[thick,dotted](0,5)--+(10,0);
\draw[thick,densely dotted](0,0)--+(10,0);
\draw[very thick,loosely dashed,gray]
                    (20,10)--+(10,0);
\draw[very thick,dashed,gray]
             (20,5)--+(10,0);
\draw[very thick,densely dashed,gray](20,0)--+(10,0);
\end{tikzpicture}
```

3.6. Flèches

On dispose d'un très grand nombre de possiblilités pour composer des flèches en chargeant la bibliothèque `arrows`. Cependant, sans cette bibliothèque, il y a un choix suffisant pour les applications courantes ; ceci d'autant plus que, dans une même série de figures, on n'utilise qu'un très petit nombre de styles de flèches.

Voilà la syntaxe utilisée [12.2.4] :

— >=(type de pointe)

permet de définir la forme de la pointe de la flèche ; les types les plus courants et définis dans l'extension de base sont `to` (forme élémentaire désuète), `latex` (forme des flèches des fontes cm), `stealth` (forme des flèches de PSTrics) et `|` (un simple petit trait perpendiculaire).

— le choix du type de la flèche se fait avec les options :

`->` donne la flèche simple usuelle,

`<->` donne la flèche double usuelle,

`>->` donne la flèche avec les deux pointes dans le même sens,

`->>` donne une flèche avec double pointe,

`|<->|` donne la flèche pour coter en ajoutant la dimension avec une option appropriée qui sera vue plus loin.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[thick,->](0,10)--+(10,0);
\draw[thick,>=latex,->](0,5)--+(10,0);
\draw[thick,>=stealth,|->]
                  (0,0)--+(10,0);
\draw[thick,>=stealth,->>]
                (20,10)--+(10,0);
\draw[thick,>=stealth,<->]
                 (20,5)--+(10,0);
\draw[line width=4pt,>=stealth,->,gray](20,0)--+(18,0);
\end{tikzpicture}
```

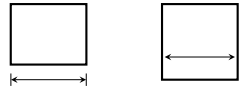
On dispose encore de la possibilité de placer l'origine et l'extrémité en retrait d'une distance donnée en utilisant les options :

`shorten<=`(dimension)

`shorten>=`(dimension)

où la dimension donnée est celle du retrait.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[thick](0,10)--++(10,0)--++(0,-8)--++(-10,0)--cycle;
\draw[very thin,>=stealth,|<->|]
                     (0,0)--++(10,0);
\draw[thick](20,10)--++(10,0)
       --++(0,-10)--++(-10,0)--cycle;
\draw[very thin,shorten >=1pt,
   shorten <=1pt,>=stealth,<->](20,3)--+(10,0);
\end{tikzpicture}
```

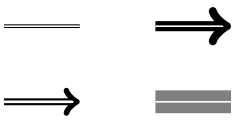


3.7. Double trait

Pour avoir un double trait, on dispose de l'option (défaut : `white`) [12.2.5] :
`double=`(couleur)
Cette option trace deux traits d'épaisseur courante séparés par un trait d'épaisseur déterminée par l'option (défaut : épaisseur de 0,6 pt) :
`double distance=`(dimension)

La largeur du double trait est évidemment deux fois l'épaisseur de trait courante plus la distance courante de séparation.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[thin,double](0,10)--+(10,0);
\draw[thick,double distance=1pt,->]
                        (0,0)--+(10,0);
\draw[ultra thick,double,->]
                      (20,10)--+(10,0);
\draw[line width=4pt,double,gray]
                       (20,0)--+(10,0);
\end{tikzpicture}
```



3.8. Couleur

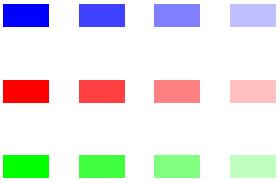
Il est hors de question dans ce petit manuel de reprendre tout ce qui est disponible dans le domaine de la couleur. On peut par exemple se reporter à la documentation de l'extension `xcolor`.

On rappelle simplement les couleurs prédéfinies dans cette extension :

Non	Couleur	Non	Couleur	Non	Couleur
`red`		`yellow`		`black`	
`green`		`orange`		`darkgray`	
`blue`		`violet`		`gray`	
`cyan`		`purple`		`lightgray`	
`magenta`		`brown`		`white`	

On rappelle aussi comment on peut varier l'intensité des couleurs avec la syntaxe suivante :
(couleur) ! (nombre entre 100 et 0)
dont on donne quelques exemples utilisant un tracé prédéfini (`rectangle`) qui sera vu à la sous-section suivante.

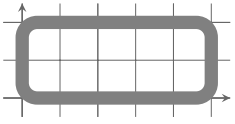
```
\begin{tikzpicture}[x=1mm,y=1mm]
\fill[blue](0,10)rectangle+(6,3);
\fill[color=blue!75](10,10)
                rectangle+(6,3);
\fill[color=blue!50](20,10)
                rectangle+(6,3);
\fill[color=blue!25](30,10)
                rectangle+(6,3);
--idem avec red--idem avec green--
\end{tikzpicture}
```



3.9. Tracés prédéfinis : `rectangle` et `grid`

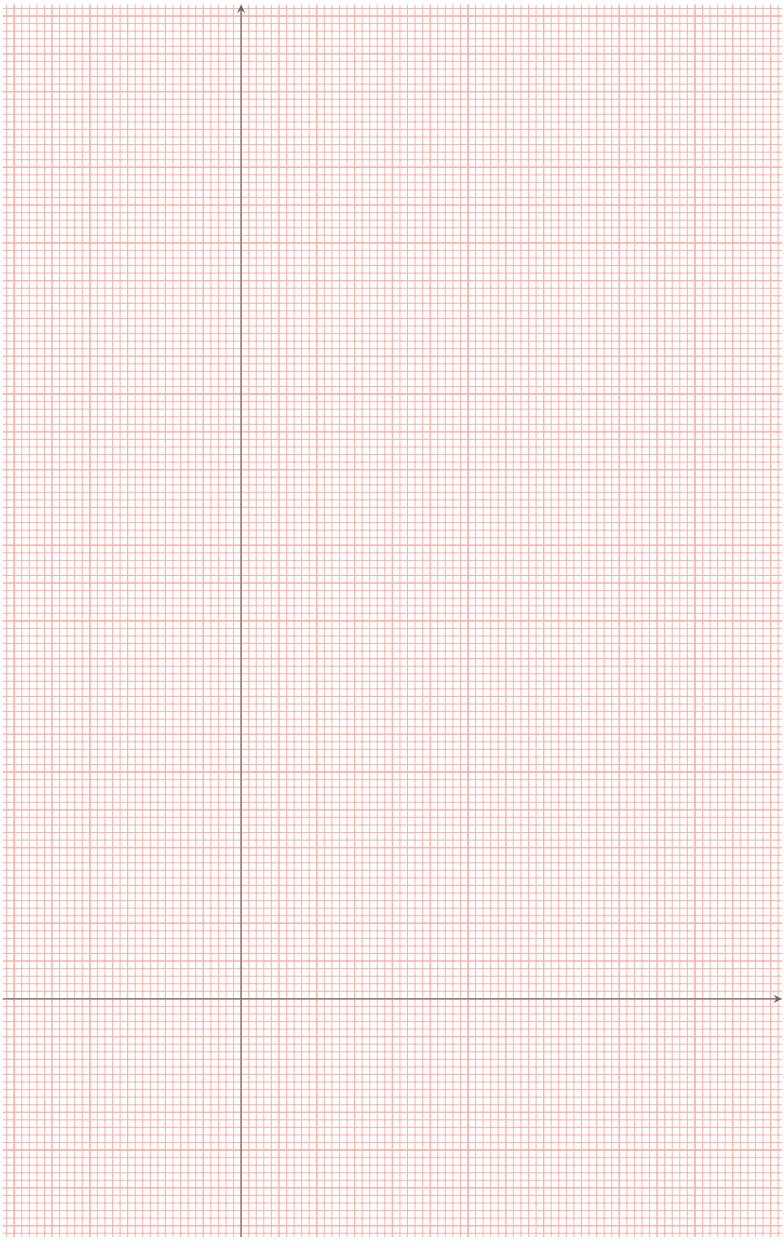
La commande [11.5] :
`\draw [options] (a,b) rectangle (c,d) ;`
trace un rectangle dont les coordonnées des points bas-gauche et haut-droit sont respectivement `(a,b)` et `(c,d)` :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[line width=5pt,gray,rounded
corners=2mm](0,0)rectangle(25,10);
\end{tikzpicture}
```



Il y a aussi un autre tracé prédéfini qui utilise des lignes droites : la grille qui s'obtient avec [11.9] :
`\draw[options] (a,b) grid (c,d)`
où `(a,b)` et `(c,d)` ont la même signification que dans le tracé ci-dessus. En voici une utilisation simulant le papier millimétré du siècle précédent (on a ajouté les axes ; la figure ci-après est obtenue avec la commande :
`\axespapiermilli(-31.5mm,-21.5mm)(71.5mm,31.5mm)`
dont la définition est :

```
\def\axespapiermilli(#1)(#2){
  \draw[step=1mm,line width=0.1pt,color=black!60](#1)grid(#2);
--idem avec 5mm et 0.2pt--idem avec 10mm et 0.3pt--
\def\xmin(##1,##2){##1}\def\xmax(##1,##2){##1}
\def\ymin(##1,##2){##2}\def\ymax(##1,##2){##2}
\draw[->,>=stealth,line width=0.4pt,color=black!60]
                              (\xmin(#1),0mm)--(\xmax(#2),0mm);
\draw[->,>=stealth,line width=0.4pt,color=black!60]
                             (0mm,\ymin(#1))--(0mm,\ymax(#2));}
```



CHAPITRE 3

La machinerie Ti*k*Z

Un titre inattendu pour un chapitre où on a rassemblé beaucoup de choses n'ayant pas toujours de relations directes entre elles mais qui sont nécessaires pour pouvoir profiter rapidement des immenses possibilités graphiques de Ti*k*Z.

1. Préparations préliminaires

1.1. Installation des fichiers

On peut se procurer le fichier `pgf-1.10.tar.gz` sur le site : `http://sourceforge.net/projets/pgf/`.

Une fois décompacté, on a 5 répertoires. L'installation peut se faire automatiquement, mais l'auteur de ces quelques lignes a toujours une peur bleue de ces méthodes.

Voici donc comment installer Ti*k*Z à la main (distribution MikTEX) :

— le répertoire `latex` contient un sous-répertoire `pgf` qu'il faut placer dans le sous-répertoire `texmf/tex/latex` de l'arborescence,

— on suit la même procédure avec les répertoires `plain` et `context`,

— le répertoire `generic` contient un sous-répertoire `pgf` qui est à placer dans le sous-répertoire `texmf/tex/generic` ;

— le sous-répertoire `doc/generic` contient un sous-répertoire `pgf` que l'on doit placer le sous-répertoire `texmf/doc/generic`.

Pour terminer, il ne faut surtout pas oublier de regénérer la base de données de TEX pour qu'il puisse aller chercher les fichiers là où ils ont été placés !

1.2. Ajouts au préambule

Il faut appeler l'extension avec la commande [GUT 1.2.1], [9.1] :

```
\usepackage{tikz}
```

L'extension `tikz` appelle, pour son « fonctionnement de base », les extensions nécessaires, `xcolor` et `calc` en particulier.

Mais, pour des figures complexes, il sera nécessaire d'utiliser les bibliothèques d'éléments disponibles que l'on chargera avec la commande :
`\usetikzlibrary{`(biblio)`}`
ou (biblio) est un nom ou une suite de noms de bibliothèques séparés par des vigules.

Il est évidemment astucieux de mettre les macros Ti*k*Z dans un fichier spécifique `mactikz.tex` que l'on appellera avec la commande `\input`.

2. Utilisation

2.1. Environnement de base

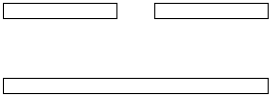
Avec Ti*k*Z, on fait une figure en plaçant les commandes graphiques à l'intérieur de l'environnement graphique spécifique comme suit [9.2.1] :

```
\begin{tikzpicture}
...commandes...
\end{tikzpicture}
```

Toutes les commandes graphiques doivent être à l'intérieur de cet environnement, excepté la commande `\tikzstyle` permettant la définition de styles par l'utilisateur et abordée à la section suivante.

Heureusement, Ti*k*Z possède un moyen d'abréger l'écriture pour une figure composée d'une seule commande graphique (éventuellement d'un très petit nombre de telles commandes) ; on écrit [9.2.2] :
`\tikz [options] {`(commandes)`}`
ou
`\tikz [options]` (commande) ;
C'est très commode : en voici un exemple de chaque syntaxe, la deuxième étant réservée au cas d'une seule commande car un seul point-virgule est accepté.

```
\tikz[x=1mm,y=1mm]
     {\draw(0,0)rectangle(15,2);
    \draw(20,0)rectangle(35,2);}
\tikz[x=1mm,y=1mm]
      \draw(0,0)rectangle(35,2);
```



2.2. Intégration des figures Ti*k*Z

Toute figure créée avec Ti*k*Z en respectant les directives précédentes se comporte comme une boîte TEX ; en effet, une `BoundingBox` a été créée par l'environnement `tikzpicture`. Cette boîte se place dans la page TEX comme un `parbox` ou une `minipage` ; par défaut, le bas de la figure se place sur la ligne de base de la ligne courante [9.2.1].

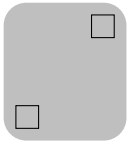
On peut monter ou descendre la figure avec l'option de figure :
`baseline=`(dimension)
qui place l'origine des coordonnées de la figure à la distance (dimension) de la ligne de base de la ligne courante (la dimension nulle ou l'absence de valeur, font que l'origine des coordonnées est sur cette ligne de base). D'une manière générale, chacun peut utiliser son arsenal de méthodes de placement habituel et, en particulier, l'environnement `figure` de LATEX.

A propos du placement des figures, on va citer l'option de figure :
`show background rectangle`
qui permet « d'habiller » les figures avec un fond, un cadre, etc. Voilà l'exemple du cadre (attention : il faut charger la bibliothèque `backgrounds` [18] qui contient beaucoup d'autres possibilités) :

```
\begin{tikzpicture}[x=1mm,y=1mm,
      show background rectangle]
\draw(0,0)rectangle(3,3);
\draw(32,12)rectangle(35,15);
\end{tikzpicture}
```

Pour obtenir un fond légèrement grisé, il faut écraser la définition du style par défaut fournie par la bibliotèque avec la re-définition :
`\tikzstyle{background rectangle}=[fill=lightgray]`
On peut en outre faire un contour en trait double, arrondir les angles, ombrer au lieu de remplir, etc. La distance entre la `BoundingBox` initiale et le `background rectangle` peut être modifiée avec l'option :
`inner frame sep=`(dimension)

```
\tikzstyle{background rectangle}=[fill=lightgray,
      inner frame sep=4mm,rounded corners=3mm]
\begin{tikzpicture}[x=1mm,y=1mm,
      show background rectangle]
\draw(0,0)rectangle(3,3); \draw(10,12)rectangle(13,15);
\end{tikzpicture}
```



3. Manipulation des options

Jusqu'à maintenant, on a placé les options entre `[]` après les commandes `\path` ou `\draw` (exepté les options de choix d'unités et quelques autres qui ont été placées comme option de l'environnement `tikzpicture`) : on va voir toutes les manières de placer les options ainsi que les propriétés qui résultent de ces placements.

3.1. Placement en option de figure

Les options qui seront presque toujours utilisées dans une figure vont être placées comme options de l'environnement `tikzpicture` (comme on l'a fait et fera dans les exemples pour le choix des unités). Si, par exemple, on veut avoir dans toute une figure des pointillés très fins, des flèches du type PostScript et une épaisseur de 1 pt pour presque tous les traits, on écrit [9.2.1] :

```
\begin{tikzpicture}
   [densely dotted,>=stealth,line width=1pt]
```

Toutes ces options vont devenir des options par défaut pour cette figure et, bien entendu, prévalent sur les options par défaut de Ti*kZ*.

3.2. Placement en option d'une partie de figure

Les options qui ne sont utilisées que pour une partie déterminée d'une figure (options différentes des options par défaut de Ti*kZ* ou options différentes de celles installées par défaut comme options de figure) vont être placées comme options d'un environnement spécial prévu à cet effet. Par exemple, si on veut une partie d'une figure avec des traits gris et très fins, on écrit [9.2.2] :

```
\begin{scope}[draw=gray,very thin]
...commandes de la partie de figure...
\end{scope}
```

Ces deux options deviennent des options par défaut dans cette partie de la figure ; elles prévalent sur toutes les options par défaut courantes à l'entrée de l'environnement `scope`. Remarquons que, si l'on avait écrit `gray` seulement, le gris serait utilisé pour les tracés et les remplissages.

3.3. Placement en option de commande

C'est ce qui avait été fait jusqu'à maintenant (sauf pour le choix des unités et quelques exceptions). Les options ainsi placées prévalent sur

toutes les options par défaut courantes à la position de la commande et sont utilisées jusqu'à la fin de cette commande. La notion d'option en tant qu'option de commande est introduite dans [GUT 1.7].

Cette notion de prévalance de certaines options sur d'autres (en fonction de leur positionnement) appelle une remarque très importante pour l'écriture des macros. Si l'on écrit une macro sans aucune option, lors de son exécution, elle va utiliser les options par défaut courantes à la position de l'exécution et l'on risque d'avoir des résultats non souhaités ! Le seul moyen est de placer toutes les option souhaitées dans la macro ou d'envelopper le code de la macro dans une environnement `scope` auquel on affecte toutes les options souhaitées.

On trouvera ci-dessous des exemples des trois possibilités de placement précédentes. On veut écrire une macro qui trace un trait en traitillé épais (option de figure `thick`) et noir (option par défaut de Ti*k*Z `black`). Ce trait est facilement identifiable par le traitillé : seule la macro `\traitb` donne toujours le résultat souhaité. Par contre, la macro `\traita` prend les options courantes qui varient suivant sa position : par exemple, le trait devient très épais (2 mm) à l'intérieur de l'environnement `scope`. A titre d'exercice, le lecteur peut déterminer les options actives à chaque tracé en tenant compte des règles de prévalance.

```
\def\traita#1{\draw[dashed](0,#1)--+(20,0);}
\def\traitb#1{\draw[dashed,thick,
                black](0,#1)--+(20,0);}
\begin{tikzpicture}[x=1mm,y=1mm,thick,
            gray]\draw(0,40)--+(20,0);
\traita{35}  \traitb{30}
\begin{scope}[line width=2mm]
\draw(0,25)--+(20,0);
\draw[ultra thin,black](0,20)--+(20,0);
\traita{15}  \traitb{10}
\end{scope}
\draw[double,black](0,5)--+(20,0);
\draw[lightgray,line width=1mm](0,0)--+(20,0);
\end{tikzpicture}
```

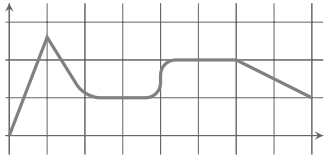
3.4. Placement des options dans les spécifications d'une commande

Enfin, il y a des options qui peuvent être données dans les spécifications mêmes d'une commande [9.4].

Un cas typique d'option pour laquelle ce n'est pas possible est l'option de couleur : cette option est valable pout toute la commande ; si plusieurs couleurs sont données dans la spécification d'une commande, c'est la dernière couleur qui prévaut (l'explication de cette restriction exigerait d'entrer dans les entrailles de Ti*kZ*, ce que l'on ne va pas faire).

Par contre, cette possibilité est présente pour des options telles que celle commandant l'arrondissement des angles d'une ligne brisée, le dédoublement de la ligne, etc. Il suffit alors de préciser le domaine auquel on veut affecter l'option par des `{}`, comme on a l'habitude de faire avec TEX. Voici un exemple de ligne brisée dont seulement trois sommets sont arrondis.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[very thick,gray](0,0)--(5,13)
    {[rounded corners=2mm]--(10,5)--
  (20,5)--(20,10)}--(30,10)--(40,5);
\end{tikzpicture}
```



4. Définition de styles par l'utilisateur

Le contenu de cette section devient particulièrement utile lorsque, ayant acquis une certaine aisance, on se lance dans la production de nombreuses figures : la définition et l'utilisation de styles spécifiques permet d'assurer une forte homogénéité de l'ensemble des figures d'un même document tout en respectant certaines directives de composition d'ordre pédagogique ou esthétique voulues par l'auteur.

4.1. Commande de définition d'un style

L'utilisateur définit ses propres styles avec la commande [GUT 1.8], [9.5] :

`\tikzstyle{nom du style}=[options]` ;

qui est la seule commande qui peut être placée en dehors de l'environnement `tikzpicture`. Ces styles sont alors appelées comme les options (ce ne sont en fait que des groupes d'options), là où ils sont nécessaires et grâce à la commande :

`style=`(nom du style)
Cette définition a les propriétés suivantes :

— elle permet d'utiliser un style en tant qu'option dans la définition d'un autre style ;

— la modification d'un style déjà défini par l'ajout d'options supplémentaires.

Par exemple, si `stylea` est un style déjà défini, on peut écrire :

— `\tikzstyle{styleb}=[stylea,red]` ;

qui définit le nouveau style `styleb` ayant en plus, par rapport au style `stylea`, l'option de couleur `red` ;

— `\tikzstyle{stylea}+=[red]` ;

qui redéfinit le style `stylea` en lui ajoutant l'option de couleur `red`.

Il y a encore une autre possibilité intéressante qui permet de modifier provisoirement un style en lui ajoutant des options par l'utilisation de l'option :

`set style={{stylea}+=[red]}`

L'intérêt de cette option de modification est que son domaine d'action s'étend suivant sa position, exactement comme une simple option : sur toute une figure ou sur une partie d'une d'une figure (elle est sans intérêt pour une simple commande car il est alors plus simple d'écrire simplement la commande avec le crochet d'options `[stylea,red]`).

4.2. Définitions de styles automatiquement utilisés

On peut aussi définir un style avec la commande :

`\tikzstyle{every dacos}=[options]`

Ce style va entrer en action chaque fois qu'un certain « dacos [2] » va être appelé.

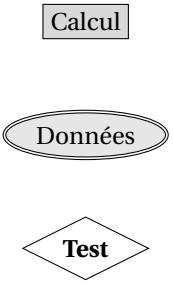
On va donner un exemple en page suivante concernant la présentation d'un algorithme où l'on a différentes formes de nœuds [13.2] : par exemple, des nœuds rectangulaires pour les opérations, des nœuds elliptiques pour les données et les résultats et des nœuds en forme de diamant pour les bifurcations. Les « dacos » seront ici `rectangle node`, `ellipse node` et `diamond node` et on aura la possibilité de varier automatiquement le contour du nœud, la couleur du fond, la fonte du texte

2. Mot passe-partout de l'occitan ; en français : qu'est-ce-que-c'est ou késako (sa version phonétique en occitan) ; cf. le *whatsit* du TEXBook.

contenu, etc. suivant le type de nœud tout en conservant une homogénéité rigoureuse (attention : il faut charger la bibliothèque `shapes`).

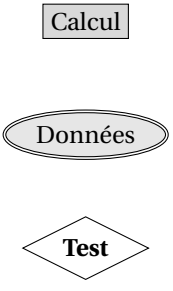
Il y a un exemple [11.12.3] destiné à marquer les points servant à construire une courbe et le « dacos » est alors la construction `plot`. Un autre exemple encore [11.13] sert a configurer le tracé de lignes de jonction d'un nœud à un autre, le « dacos » étant alors la construction `to`. Les « dacos » possibles et utilisés dans la documentation de référence sont listés dans son index (mot `every`). Bien évidemment on a les possibilités `every picture`, `every scope`, `every node`, `every label`, etc. Les cas les plus intéressants seront vus dans les prochains chapitres. Ce type de définition n'apporte rien de nouveau du point de vue graphisme mais elle prend une très grande importance pour assurer la rapidité et l'homogénéité dans le cas de la production d'un ensemble important de figures (figures d'un même ouvrage par exemple).

```
\tikzstyle{every rectangle node}
                  =[draw,fill=gray!30]
\tikzstyle{every ellipse node}
           =[draw,double,fill=gray!20]
\tikzstyle{every diamond node}
       =[aspect=2,draw,font=\bfseries]
\begin{tikzpicture}[x=1mm,y=1mm]
\node(a)at(20,30)[rectangle]{Calcul};
\node(b)at(20,15)[ellipse]{Données};
\node(c)at(20,0)[diamond]{Test};
\end{tikzpicture}
```



Pour terminer ce chapitre, on va montrer que l'on peut aussi définir des macros permettant de réduire encore la saisie. On reprend pour cela l'exemple précédent.

```
\def\Nrec(#1)(#2)#3{\node(#1)at(#2)
[rectangle,draw,fill=gray!30]{#3};}
\def\Nell(#1)(#2)#3{\node(#1)at(#2)
    [ellipse,draw,double,fill=gray!20]{#3};}
\def\Ndia(#1)(#2)#3{\node(#1)at(#2)
[diamond,draw,aspect=2,font=\bfseries]{#3};}
\begin{tikzpicture}[x=1mm,y=1mm]
\Nrec(a)(20,30){Calcul}
\Nell(b)(20,15){Données}
\Ndia(c)(20,0){Test}
\end{tikzpicture}
```



Il faut noter que les noms des nœuds `a`, `b`, etc. n'ont pas encore été utilisés à ce stade ; ils joueront un rôle lorsque l'on fera des connexions entre ces nœuds. La syntaxe complète des nœuds et la syntaxe de ces connexions seront exposées par la suite.

CHAPITRE 4

Lignes courbes

Ce chapitre expose les deux manières pratiques pour tracer des courbes. Contrairement à ce qu'on avait fait pour le chapitre concernant les lignes brisées, on va commencer par donner les constructions prédéfinies.

1. Constructions prédifinies

1.1. Cercle et ellipse

Les commandes [11.7], [11.8] :

```
\draw (a,b) circle (r) ;
\draw (a,b) ellipse (r,s) ;
```

construisent respectivement un cercle centré en $x = a$, $y = b$ et de rayon r et une ellipse centrée en $x = a$, $y = b$ et de rayons respectifs r suivant les x et s suivant les y. Si l'on ne donne pas d'unités, ce sont les unités par défaut qui sont utilisées ; le cercle n'est un cercle que si les unités suivant x et suivant y sont identiques. On ne peut utiliser que les options de tracé vues pour les lignes brisées qui gardent un sens : épaisseur du trait, terminaison, pointillés et traitillés, double trait et couleur.

1.2. Arc

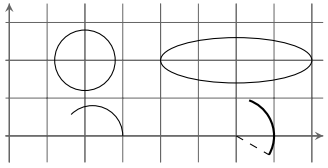
La commande [11.8] :

```
\draw (a,b) arc (u:v:r) ;
```

trace un arc d'origine en $x = a$, $y = b$, d'angle d'origine u degrés, d'angle d'extrémité v degrés et de rayon r. Si l'on ne donne pas d'unité, les unités par défaut sont utilisées. Presque toutes les options de tracé deviennent utilisables.

On donne un exemple de ces trois constructions prédéfinies. 0n remarquera comment le remplacement de `(a,b)` par `(a,b) +(u:r)` fait que `(a,b)` sont les coordonnées du centre du deuxième arc.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw(10,10)circle(4);
\draw(30,10)ellipse(10 and 3);
\draw(15,0)arc(0:135:4);
\draw[thick](30,0)+(-30:5)
                arc(-30:70:5);
\draw[dashed](30,0)--+(-30:5);
\end{tikzpicture}
```



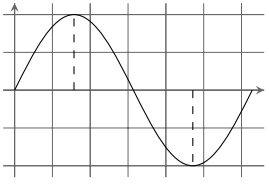
1.3. Sinus et cosinus

Les commandes [11.11] :

```
\draw (a,b) sin (c,d) ;
\draw (a,b) cos (c,d) ;
```

tracent respectivement le sinus et le cosinus sur l'intervalle $[0, \pi/2]$ transformés linéairement de telle manière que le point correspondant à l'angle 0 devient le point $x = a$, $y = b$ et le point correspondant à l'angle $\pi/2$ devient le point $x = c$, $y = d$. On donne le tracé d'une sinussoïde sur l'intervalle $[0, 2\pi]$

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw(0,0)sin(7.854,10)cos(15.708,0)
        sin(23.562,-10)cos(31.416,0);
\draw[dashed,thin](7.854,0)--+(0,10);
\draw[dashed,thin]
                (23.562,0)--+(0,-10);
\end{tikzpicture}
```



2. Courbes de Bézier

2.1. Syntaxe générale

Ces courbes ont été initialement utilisées par Bézier pour proposer des formes de capots de voiture chez Renault. Depuis que l'informatique s'est introduite dans toutes les disciplines, les courbes de Bézier sont très utilisées dans les disciplines graphiques, la création de fontes notamment. Sans aborder leur aspect mathématique on va seulement donner la méthode pour les construire.

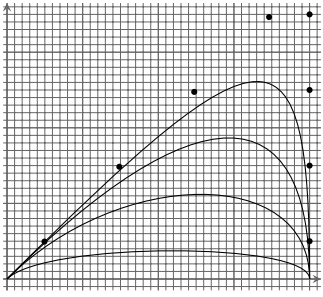
Une courbe de Bézier est obtenu par la commande [GUT 1.4], [11.3] :

```
\draw (a,b) .. controls (u,v) and (w,t) .. (c,d) ;
```

qui trace une courbe allant du point `(a,b)` au point `(c,d)` et dont la forme est déterminée par les points de contrôle `(u,v)` et `(w,t)` : la courbe part du point `(a,b)` dans la direction du point `(u,v)` et arrive

au point `(c,d)` depuis la direction du point `(w,t)` (voir page suivante).

```
\begin{tikzpicture}[x=1mm,y=1mm]
\def\point(#1){\fill(#1)circle(0.4);}
\draw(0,0)..controls(45:7)
              and(40,5)..(40,0);
\draw(0,0)..controls(45:21)
             and(40,15)..(40,0);
\draw(0,0)..controls(45:35)
             and(40,25)..(40,0);
\draw(0,0)..controls(45:49)
             and(40,35)..(40,0);
\point(45:7)  \point(40,5)
\point(45:21)  \point(40,15)
\point(45:35)  \point(40,25)
\point(45:49)  \point(40,35)
\end{tikzpicture}
```



Pour comprendre le rôle des points de contrôle, on les déplace et on suit les déformations successives de la courbe : on constate que les points de contrôle, représentés par •, en plus de déterminer la direction de la tangente à la courbe à son point de départ et à son point d'arrivée, « tirent la courbe » vers eux.

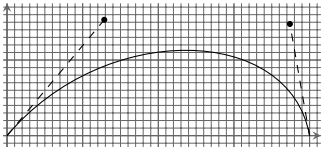
2.2. Syntaxe spéciale et applications

Dans la commande de tracé d'une coube de Bézier, les déplacements relatifs prennent un sens différent et très commode pour les applications. Si on écrit [10.3] :

`draw (a,b) .. controls +(u,v) and +(w,t) .. +(c,d);`

`+(u,v)` désigne alors un point déplacé comme vu à la section 3 du premier chapitre par rapport au point `(a,b)` ; `+(w,t)` désigne un point déplacé par rapport au point d'arrivée de la courbe ; `+(c,d)` désigne le point déplacé par rapport au point `(a,b)` : le point d'arrivée de la courbe n'est pas alors `(c,d)` mais `(a,b)+(c,d)`.

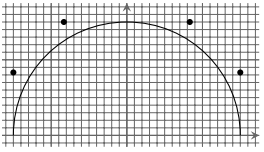
```
\begin{tikzpicture}[x=1mm,y=1mm]
\def\point(#1){\fill(#1)circle(0.4);}
\draw(0,0)..controls+(50:20)
               and +(100:15)..(40,0);
\draw[dashed,thin](0,0)--+(50:20);
\draw[dashed,thin](40,0)--+(100:15);
\point(50:20)
\point({40,0)+(100:15})
\end{tikzpicture}
```



Cette syntaxe permet d'imposer facilement la direction des tangentes aux extrémités de la coube en utilisant les coordonnés polaires.

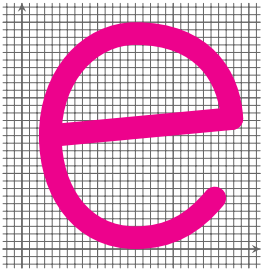
D'une façon générale, on appelle courbe de Bézier une courbe formée de plusieurs courbes du type qui vient d'être défini ; cela nous amène naturellement au tracé du cercle. La meilleure approximation que l'on peut obtenir pour un demi-cercle est donnée dans l'exemple suivant.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\def\point(#1){\fill(#1)circle(0.4);}
\draw(-15,0)..controls(-15,8.325)
     and(-8.325,15)..(0,15)..controls
     (8.325,15)and(15,8.325)..(15,0);
\point(-15,8.325)\point(-8.325,15)
\point(8.325,15)\point(15,8.325)
\end{tikzpicture}
```



L'exemple suivant montre le dessin de la lettre *e* minuscule à l'aide d'une courbe de Bézier passant par 5 points (8 points de contrôle) et d'un segment.

```
\begin{tikzpicture}[x=1mm,y=1mm,
   line width=3mm,lightgray,cap=round]
\draw (25.5,6.75)..controls+(-130:4.5)
and+(0:4.5)..(15,1.5)..controls+(180:7)
and+(-90:7.5)..(3.75,15)..controls+(90:8)
and+(180:6)..(15,28.5)..controls+(0:9)
          and+(93:5.5)..(27.75,17.25);
\draw (27.75,17.25)--(3.75,15);
\end{tikzpicture}
```



Ces courbes de Bézier sont à la base de tous les dessins de caractères des fontes logicielles disponibles ; un exemple est donné à la fin de ce chapitre : on est frappé par le peu de nombres nécessaires pour obtenir un très beau caractère.

3. Courbes passant par des points donnés

C'est sûr que les courbes de Bézier sont bien belles quand on choisit bien les points de contrôle mais, généralement, on se trouve confronté à des problèmes du type suivant : tracer une courbe qui passe par plusieurs points en n'ayant que très peu d'information supplémentaire (dans le meilleur des cas, on ne dispose jamais des points de contrôle). Comme expliqué dans le manuel de Metafont, l'idée de base est que le

programme puisse, avec les coordonnées des points guides (ou points de passage) et le peu d'information supplémentaire disponible, déterminer des points de contrôle pour tracer (sans qu'on le sache) des courbes de Bézier.

3.1. Tracer une courbe avec « plot »

La commmande [11.12.5] :

```
\draw plot[smooth,options]coordinates{(a,b)(c,d)...(y,z)};
\draw plot[smooth,cycle,options]
   coordinates{(a,b)(c,d)...(y,z)} ;
```

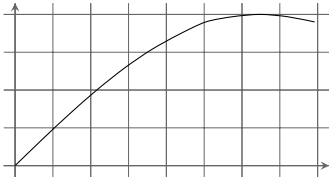
tracent respectivement les courbes ouverte et fermée passant par les points de coordonnées `(a,b)`, `(c,d)`, ..., `(y,z)`.

Si les coordonnées des points guides sont données dans un fichier, les commandes deviennent :

```
\draw plot[smooth,(options)]file{chemin/fichier} ;
\draw plot[smooth,cycle,(options)]file{chemin/fichier};
```

Sans l'option `smooth` on aurait une ligne brisée, ce qui peut être obtenu plus simplement comme on a vu à la section 1 du chapitre 2. Une autre possibilité consiste à donner la fonction à représenter à l'aide d'une syntaxe qui demande à TEX de faire une pause pour faire produire à Gnuplot un tableau de coordonnées de points guides qui seront utilisées de la même manière que les coordonnées données directement ou lues dans un fichier ; on verra tout cela en détail plus loin. On donne un exemple de courbe ouverte (sinus sur l'intervalle $[0, \pi/2]$) ; on remarquera l'utlisation des options `xscale` et `yscale` pour utiliser un tableau de nombres entier disponible tout en produisant une figure d'une taille adaptée à la mise en page choisie.

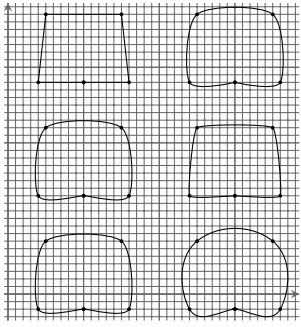
```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw plot[smooth,xscale=3.6,
yscale=0.02]coordinates{(0,0)(1,174)
 (2,342)(3,500)(4,643)(5,766)(6,866)
      (7,950)(8,985)(9,1000)(10,985)
                          (11,950)};
\end{tikzpicture}
```



Sur l'exemple suivant (en page suivante) on a, dans l'ordre : pas d'option `smooth` (ligne brisée), option `smooth` (il y a un point anguleux), option `smooth cycle` (plus de point anguleux), option supplémentaire

avec trois valeurs de la tension : 0.2, 0.5 et 0.9.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\pgfsetplotmarksize{0.5pt}
\draw plot[yshift=30mm]coordinates{(10,-2)(4,-2)
          (5,7)(15,7)(16,-2)(10,-2)};
\draw plot[smooth,xshift=20mm,
      yshift=30mm]coordinates{(10,-2)
    (4,-2)(5,7)(15,7)(16,-2)(10,-2)};
\draw plot[smooth cycle,
      yshift=15mm]coordinates{(10,-2)
    (4,-2)(5,7)(15,7)(16,-2)(10,-2)};
\draw plot[smooth cycle,tension=0.2,
xshift=20mm,yshift=15mm]coordinates
{(10,-2)(4,-2)(5,7)(15,7)(16,-2)(10,-2)};
\draw plot[smooth cycle,tension=0.5]
 coordinates{(10,-2)(4,-2)(5,7)(15,7)
                    (16,-2)(10,-2)};
\draw plot[smooth cycle,tension=0.9,xshift=20mm]
coordinates{(10,-2)(4,-2)(5,7)(15,7)(16,-2)(10,-2)};
\end{tikzpicture}
```



L'option des trois dernières courbes [11.12.5] :
`tension=`(nombre entre 0 et 1)
commande le « gonflement » de la courbe (qu'elle soit ouverte où fermée). La courbe passant par les sommets d'un carré est pratiquement le cercle circonscrit du carré si on a pris l'option `tension=1` ; si l'option est absente mais si l'option `smooth` est donnée, alors la la valeur de la tension est (par défaut) 0,55 ; sans l'option `smooth`, la valeur prise est 0, la courbe est une suite de segements rectilignes (en interne un segment est une courbe de Bézier particulière ; cela se voit plus facilement dans les langages Metafont et Metapost).

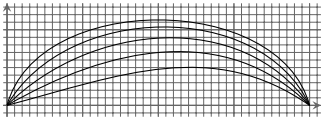
Les points guides peuvent être marqués avec un petit cercle plein, un signe plus et une petite croix de Saint-André grâce à l'option [11.12.4] :
`mark=*` ou `+` ou `x`
La dimension des marques a été modifiée par la commande [23.3] :
`\pgfsetplotmarksize{0.5pt}` On peut ne marquer qu'un point sur 2 ou 3, etc., en écrivant [11.12.4] :
`mark repeat=2` ou 3 ou ...
On peut aussi décaler le début du marquage ou ne marquer que certains points avec les options [11.12.4] ;

`mark phase=`(nombre de points non marqués au début)
`mark indices={`(numéros des points à marquer)`}`

3.2. Tracer une courbe avec « to »

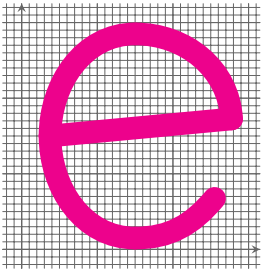
On aborde une deuxième possibilité pour tracer une courbe lorsqu'on dispose, en plus des coordonnées des points guides, des directions des tangentes en ces points. En effet, la commande [11.13] :
`draw (a,b) to[out=u,in=v,`(options)`] (c,d)`
trace une courbe allant du point `(a,b)` au point `(c,d)` et qui part du premier point dans la direction u degrés et arrivant vers le deuxième point depuis la direction v degrés. Voici quelques courbes ayant les mêmes extrémités.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw(0,0)to[out=15,in=145](40,0);
--idem avec [out=30,in=135]--
--idem avec [out=45,in=125]--
--idem avec [out=60,in=115]--
--idem avec [out=75,in=105]--
\end{tikzpicture}
```



On se rend compte de la grande différence avec le tracé des courbes de Bézier de la section 2.1 : malgré l'augmentation des angles de départ et d'arrivée par rapport à la ligne joignant ces deux points, la courbe reste relativement « collée » à cette ligne. On reprend le tracé de la lettre *e* de la section 2.2 en gardant les mêmes points guides et les mêmes directions des tangentes en ces points … et quelle surprise !

```
\begin{tikzpicture}[x=1mm,y=1mm,
line width=3mm,lightgray,cap=round]
\draw(25.5,6.75)to[out=-130,in=0]
                              (15,1.5);
\draw(15,1.5)to[out=180,in=-90](3.75,15);
\draw(3.75,15)to[out=90,in=180](15,28.5);
\draw(15,28.5)to[out=0,in=93]
                         (27.75,17.25);
\draw (27.75,17.25)--(3.75,15);
\end{tikzpicture}
```

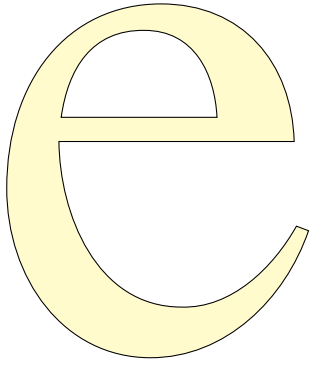


On constate que, si on décompose une ligne courbe en suffisamment d'éléments et si, pour chacun de ces éléments, la direction de la tangente orientée entre ses extrémités varie « raisonnablement », on peut espérer de bons résultats. Pour la courbe de Bézier de la section 2.2, on a donné

non seulement les orientations des tangentes aux points guides mais aussi les distances entre ces points et les points de contrôle voisins : on constate que le résultat du tracé avec l'opération `to` correspond approximativement au résultat de la détermination par « tâtonnement » de ces distances. Rien d'étonnant : le programme trace en fait une courbe de Bézier après avoir déterminé des points de contrôle suivant une méthode prenant en compte les données : points guides et directions des tangentes en ces points.

Pour terminer, voici une lettre *e* où se mêlent art et informatique (exemple cité par Jacques André, *Cahiers GUTenberg* nº 97, p. 97 ; ici présenté codé en Ti*k*Z).

```
\begin{tikzpicture}[x=0.1mm,y=0.1mm]
\draw[fill=gray!25] (402,276)..controls(399,380)and(334,458)
..(226,458)..controls(102,458)and(22,356)..(22,214)..controls
(22,95)and(97,-10)..(212,-10)..controls(312,-10)and(390,68)..
(421,158)--(405,164)..controls(374,109)and(319,55)..(253,57)
..controls(140,57)and(92,181)..(91,276)--cycle;
\draw[fill=white] (94,308)..controls(103,372)and(134,424)..
(204,423)..controls(270,423)and(297,366)..(300,308)--cycle;
\end{tikzpicture}
```



CHAPITRE 5

Nœuds, liaisons et labels : organigrammes, algorithmes…

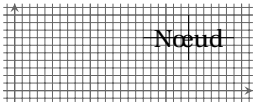
Ce chapitre est consacré aux nœuds, aux liaisons inter-nœud et aux labels, ce qui permet de construire des organigrammes, des algorithmes de toutes sortes, mais aussi de placer des lettrages sur les figures.

1. Nœuds

1.1. Syntaxe générale

La syntaxe pour la création d'un nœud au point de référence de coordonnées $x = a$, $y = b$ et contenant du matériel (c'est-à-dire, au sens du TEXBook, du texte, des formules ou du graphisme) est [13.2] :
`\node [options] (nom du nœud) at (a,b) {matériel} ;`
Le nom du nœud peut être une chaîne de lettres et de nombres avec des espaces (sans lettre accentuée bien entendu) ; on peut par exemple utiliser des noms du type `P_1`, `N_a`, etc. Les options possibles vont être examinées ci-dessous. Sans aucune option, on a le résultat suivant où l'on remarque que le matériel du nœud est simplement centré sur le point de référence.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[thin](17,7)--(29,7);
\draw[thin](23,4)--(23,10);
\node(a)at(23,7){N\oe ud};
\end{tikzpicture}
```



1.2. Options de présentation

Voici la liste des options de présentation dont certaines ont déjà été vues et ne seront données que par leur nom générique [13.2], [13.13].

— `draw=`(couleur) trace un contour de couleur spécifiée autour du matériel.

— Options de tracé valables pour les courbes fermées : épaisseur des traits, double trait, pointillés, traitillés et arrondissement des angles pour la forme `rectangle`.

— `shape=rectangle`, `circle`, `ellipse` ou encore `diamond` définit la forme du contour. Pour la dernière forme, il faut charger la bibliothèque `shapes` [25] qui contient d'autres formes intéressantes.

— `aspect=`(nombre) fixe le rapport (demi-axe horizontal)/(demi-axe vertical) pour la forme `diamond`.

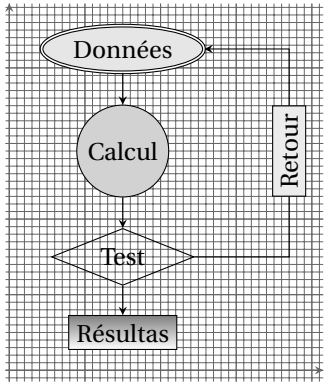
— `fill=`(couleur) remplit l'intérieur du contour tracé avec la couleur spécifiée.

— `shade` suivie des options d'ombrage (facultatives) sert à ombrer l'intérieur du contour (ces options seront vues par la suite).

— `rotate=`(angle) tourne le nœud de l'angle spécifié exprimé en degrés.

L'exemple qui suit montre l'action d'options de chaque type décrit ci-dessus.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\node[draw,ellipse,double,
fill=gray!20](a)at(15,42.5){Données};
\node[draw,circle,fill=gray!35]
                (b)at(15,29){Calcul};
\node[draw,diamond,aspect=2.5]
                  (c)at(15,15){Test};
\node[draw,rectangle,shade]
               (d)at(15,5){Résultas};
\node[draw,fill=gray!15,
      rotate=90](e)at(37,29){Retour};
\draw[>=stealth,->](a)--(b);
\draw[>=stealth,->](b)--(c);
\draw[>=stealth,->](c)--(d);
\draw[>=stealth,->](c)-|(e)|-(a);
\end{tikzpicture}
```

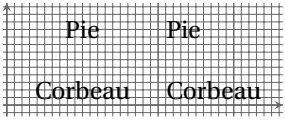


1.3. Options de positionnement

Il s'agit du positionnement du nœud par rapport au point de référence ; on a vu que, sans option de positionnement, le nœud est centré sur le point de référence, ce que l'on peut exprimer en disant que le point de référence est au centre du nœud. Si les points de référence de plusieurs nœuds sont alignés verticalement, les nœuds correspondants

seront alignés par leur centre (alignement par défaut) ; mais on peut imaginer que, dans un organigramme, on veuille aligner les nœuds par leur côtés gauches ; on dispose pour cela de l'option `anchor=west` ou `right` dont le nom est justifié par le fait que le point de référence se trouve sur le côté ouest du nœud ou que le nœud de trouve à droite de ce point. On préférera la première dénomination [13.5].

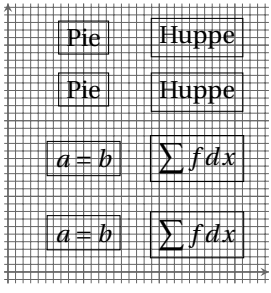
```
\begin{tikzpicture}[x=1mm,y=1mm]
\node(a)at(10,10){Pie};
\node[anchor=west](b)at(20,10){Pie};
\node(c)at(10,2){Corbeau};
\node[anchor=west](d)at(20,2){Corbeau};
\end{tikzpicture}
```



Bien évidemment, on a aussi les valeurs `east`, `north`, `south`, `north east` et les trois autres valeurs composées.

De la même manière, on peut modifier l'alignement horizontal qui est centré par défaut. On dispose pour cela de l'option `anchor=base` qui permet un alignement des lignes de base [13.5]. L'exemple suivant montre bien la différence entre l'alignement par défaut (à la mi-hauteur des rectangles) et l'alignement sur les lignes de base.

```
\tikzstyle{every node}=[draw,rectangle]
\begin{tikzpicture}[x=1mm,y=1mm]
\node(a)at(10,31){Pie} ;
\node(b)at(25,31){Huppe};
\node[anchor=base](c)at(10,23){Pie};
\node[anchor=base](d)at(25,23){Huppe};
\node(e)at(10,15){$a=b$};
\node(f)at(25,15){$\ds\sum fdx$};
\node[anchor=base](g)at(10,4){$a=b$};
\node[anchor=base](h)at(25,4)
                    {$\ds\sum fdx$};
\end{tikzpicture}
```



L'avantage de la première dénomination est que l'on peut combiner les deux types d'option, par exemple `anchor=base west` (toujours dans cet ordre).

Dans le cas d'une seule ligne de texte, le plus simple est de placer un `\strut` en fin de texte (fantôme vertical de la parenthèse) qui assure une hauteur égale de tous les rectangles et donc assure l'alignement des rectangles et des lignes de base.

1.4. Options de composition

Ces options concernent ce que nous avons appelé le matériel du nœud ; l'action de chacune des dernières options est, d'après le nom, suffisammment claire pour les utilisateurs de TEX pour qu'il ne soit pas nécessaire de la décrire [13.4], [13.13].

— `text=`(couleur) détermine la couleur du texte (noir par défaut).

— `font=`(commandes NFSS pour spécifier le corps, la graisse, etc.) ; `font=\itshape\bfseries`, par exemple. Pour modifier la fonte sur une partie du texte seulement, il faut utiliser les commandes usuelles de TEX-LATEX.

— `text width=`(dimension) détermine la largeur du texte du nœud.

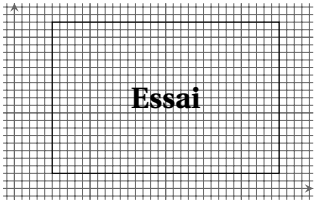
— Il y a ensuite les options `text justified`, `text ragged`, `text badly ragged` (pas de coupure de mot), `text centered`, `text badly centered` (pas de coupure de mot).

```
\begin{tikzpicture}[x=1mm,y=1mm]
\node[draw,rectangle,text width=33mm,
    text badly centered,text=blue,
    font=\Large\bfseries](b)at(20,30)
              {Essai de texte centré};
\node[draw,rectangle,text width=33mm,
       text badly ragged](a)at(20,10)
    {Essai pour voir la justification
    en drapeau sans coupure de mots};
\end{tikzpicture}
```

— `inner sep=`(dimension) détermine la distance entre le texte et le contour du nœud (défaut : une espace intermot nornale). Cette distance peut être différente dans le sens horizontal et le sens vertical grâce aux deux options :
`xinner sep=`(dimension) et `yinner sep=`(dimension)

— `minimum height=`(dimension) et `minimum width=`(dimension) servent à assurer des dimensions minimales au nœud complet.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\node[draw,rectangle,minimum height
      =20mm,minimum width=30mm,font=
\Large\bfseries](a)at(20,17){Essai};
\end{tikzpicture}
```



2. Liaisons et labels attachés

Cette section est consacrée aux liaisons intermots et aux labels qui peuvent leur être attachés. On précise d'abord quelques notations générales valables dans les différents cas envisagés.

— On remarque d'abord sur les deux liaisons `AA` vers `BB` et `FF` vers `GG` du premier exemple qu'il y a deux variantes de la syntaxe : le label est construit avec l'expression [11.13], [13.7] :
`node [options] {matériel}`
qui est placée, soit après la commande qui trace la ligne de liaison, soit à l'intérieur des spécifications de cette commande, juste avant le deuxième nœud. Nous préfèrerons la première syntaxe qui, *lorsqu'elle peut être utilisée*, est plus claire : la liaison d'abord, le label ensuite.

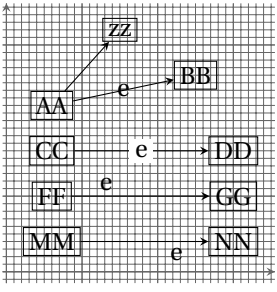
— Le nombre p, $0 \leqslant p \leqslant 1$, désigne la position du label le long de la ligne de liaison ; 0 correspond à l'origine et 1 à l'extrémité ; l'option correspondante s'écrit `pos=p`.

— La position latérale du label par rapport à la ligne de liaison est déterminée par une option ainsi définie : en parcourant la ligne de liaison de l'origine vers l'extrémité, pour que le label se trouve à gauche, il faut donner l'otion `above`, pour qu'il se trouve à droite, il faut donner l'option `below` ; sans option, il est sur la ligne de liaison.

2.1. Liaisons par un seul segment

On remarque bien sur l'exemple la direction du segment de liaison (il provient du centre du nœud de départ et va vers le centre du nœud

```
\def\nrec(#1)(#2){\node(#1)at(#2)[rectangle,draw]{#1#1};}
\begin{tikzpicture}[x=1mm,y=1mm,
        inner sep=2pt,>=stealth]
\nrec(A)(6,22)\nrec(B)(25,26)
\nrec(z)(15,32)\draw[->](A)--(z);
\draw[->](A)--(B)node[pos=0.5]{e};
\nrec(C)(6,16)\nrec(D)(30,16)
\draw[->](C)--node[fill=white]{e}(D);
\nrec(F)(6,10)\nrec(G)(30,10)
\draw[->](F)--(G)node[pos=0.25,above]{e};
\nrec(M)(6,4)\nrec(N)(30,4)
\draw[->](M)--(N)node[pos=0.25,below]{e};
\end{tikzpicture}
```

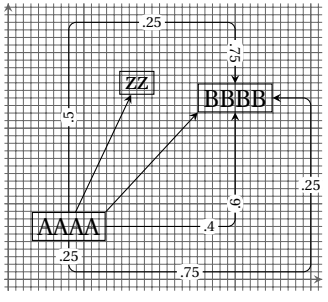


d'arrivée), les différentes possibilités de position du label et le petit fond blanc autour d'un label. En effet, quand le label est sur la liaison (et ceci s'applique aussi aux cas où le label se superpose à des parties de la figure), on ajoute les options :
`rectangle,fill=white` ou `circle,fill=white`
qui découpent ce petit fond blanc rendant le label plus lisible.

2.2. Liaison par deux, trois ou quatre segments parallèles aux axes

On note d'abord sur l'exemple que remplacer `AA` par `AA.north east` fait que la liaison du nœud `AAAA` vers le nœud `BBBB` part de la position nord-est du nœud `AAAA` au lieu de donner l'impression de partir de son centre comme le fait la liaison du nœud `AAAA` vers le nœud `zz`; une modification semblable fait que cette liaison arrive à l'angle sud-ouest du nœud `BBBB` et ne se dirige pas vers le centre de ce nœud. On remarque ensuite une liaison à trois segments et deux liaisons à quatre segments ; ces liaisons, qui nécessitent la donnée d'un point supplémentaire, seront décrites en détail après la figure qui suit.

```
\def\nrec(#1)(#2){\node(#1)at(#2)[rectangle,draw]{#1#1};}
\begin{tikzpicture}[x=1mm,y=1mm,inner sep=2pt,>=stealth]
\nrec(AA)(8,7)\nrec(BB)(30,24)\nrec(z)(17,26)
\tikzstyle{every path}
          =[rounded corners=1mm,->]
\tikzstyle{every node}
            =[fill=white,scale=0.6]
\draw(AA.north east)
                  --(BB.south west);
\draw(AA)--(z);
\draw(AA)-|(BB)node[pos=0.4,sloped]
       {.4}node[pos=0.6,sloped]{.6};
\draw(AA)--node[pos=0.5,sloped]
{.5}(8,34)-|node[pos=0.25,sloped]
{.25}node[pos=0.75,sloped]{.75}(BB);
\draw (AA)|-node[pos=0.25]{.25}
node[pos=0.75]{.75}(40,1)|-node[pos=0.25]{.25}(BB);
\end{tikzpicture}
```



Ces liaisons à plus de deux segments, nécessitant un point supplémentaire, doivent être considérées comme formées de deux éléments différents (l'un avant ce point, l'autre après) représentés par `--` pour un seul segment, par `|-` pour un chemin d'abord suivant les *y* puis sui-

vant les x et par `-|` pour un chemin du type inverse. Cela implique qu'il faut utiliser la deuxième syntaxe pour poser les labels le long de la liaison : l'expression `node[.]{.}` doit être placée à la fin de chaque élément, c'est-à-dire juste avant le deuxième nœud de l'élément, en tenant compte du fait que, pour les éléments contenant un angle, l'angle est à la position $p = 0.5$. Sur la figure les labels donnent la valeur de p déterminant leur position. L'option `sloped` assure que le label a sa ligne de base parallèle à la liaison (option également valable pour les liaisons par des courbes).

La position des commandes `\tikzstyle` appelle deux remarques ; d'une part, si on avait donné l'option `rounded corners=1mm` comme option de figure, les « vrais » nœuds auraient eu aussi les angles arrondis alors que seuls les liaisons et les petits rectangles blancs des labels (qui sont des nœuds du point de vue code) ont les angles arrondis ; d'autre part, l'option `scale=0.6` ne s'applique qu'au matériel des labels mais pas aux « vrais » nœuds définis au paravant.

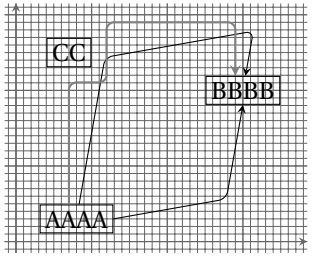
On note enfin que l' on a utilisé les commandes suivantes :

— `\node[coordinate,shift={(c,d)}] (N) at (a,b) {} ;`
crée un nœud vide (un point invisible mais utilisable pour les tracés) de nom `N` situé en `(a,b)` et déplacé de `(c,d)` [39.6] ; le point `(a,b)` peut être remplacé par `(xx)` où `xx` est le nom d'un nœud déjà défini ; ensuite `(c,d)` peut être exprimé en coordonnées polaires `(w:r)`, ce qui permet de définir une droite passant par un point `(a,b)` et faisant un angle de `uu` degrés (droite passant par `(a,b)` et le point défini avec la formule précédente où `(10,uu)` remplace `(c,d)`. L'option `shift=...` peut être remplacée par `xshift=...` ou `yshift=...` en donnant une dimension avec unité sans `()` ni `{}`.

— `(intersection of a,b--c,d and e,f--g,h)`
donne le point d'intersection de la droite passant par `(a,b)` et `(c,d)` et la droite passant par `(e,f)` et `(g,h)` [GUT 1.15] (on remarque la syntaxe spéciale : pas de `()` entourant les coordonnées).

On note que, si l'on a accès au point d'intersection de deux droites, il n'y a pas la même possibilité pour l'intersection de deux courbes comme cela est possible dans les languages Metafont et Metapost. L'exemple suivant montre une liaison à cinq segments, des liaisons avec des segments inclinés par rapport aux axes et la méthode à suivre pour faire partir (arriver) des segments d'un (sur) un même nœud.

```
\def\nrec(#1)(#2){\node(#1)at(#2)[rectangle,draw]{#1#1};}
\begin{tikzpicture}[x=1mm,y=1mm,inner sep=2pt,>=stealth]
\nrec(AA)(8,3)\nrec(BB)(30,20)\nrec(C)(7,25)
\node[shift=(10:10)](AAe)at(AA.east){};
\node[shift=(80:10)](AAn)at(AA){};
\node[shift=(-100:10)](BBs)at(BB.south){};
\node[shift=(80:10)](BBn)at(BB){};
\node[shift=(10:10)](423)at(4,23){};
\node[coordinate,shift={(2mm,-2mm)}]
            (C22)at(C.south east){};
\node[coordinate,xshift=-1mm]
                (BB2)at(BB.north){};
\node[coordinate,xshift=-1mm]
                (AA2)at(AA.north){};
\tikzstyle{every path}
           =[rounded corners=1mm,->]
\draw (AA.east)--(intersection of
      AA.east--AAe and BB.south--BBs)
                      --(BB.south) ;
\draw (AA)--(intersection of AA--AAn
and 4,23--423)--(intersection of 4,23--423 and BB--BBn)--(BB);
\draw[gray,thick](AA2)|-(C22)|-(20,29)-|(BB2);
\end{tikzpicture}
```

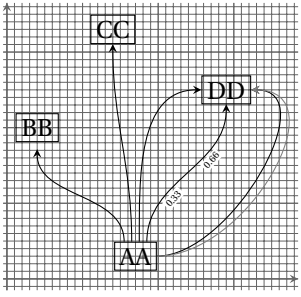


La figure montre d'abord deux liaisons `AAAA` vers `BBBB` de la figure précédente faites cette fois avec des segments faisant un angle de 10° avec les axes (autrement dit : axe des x tourné de $+10°$ et axe des y tourné de $-10°$) ; ces deux liaisons ont un point d'ancrage différent : la première part du point est, `AA.east`, du nœud `AAAA` alors que la seconde provient du centre, `AA`, de ce nœud.

Ensuite, en gris épais, on a une liaison `AAAA` vers `BBBB` avec évitement du nœud `CC`. Le point nécessaire pour l'évitement, `C22`, a été défini à partir du point `C.south east` pour que la liaison « longe » le nœud à la distance de 2 mm, ce qui évite tout tâtonnement ; quant au départ de la liaison vers le haut, il se fait au point `AA2` décalé du point `AA.north` de 1 mm vers les x négatifs (la même remarque vaut pour le point d'arrivée de la liaison). Cette méthode permet de faire arriver ou partir autant de liaisons qu'on veut sur un même nœud sans recouvrement de ces liaisons entre elles.

Les quatre liaisons en trait noir de l'exemple sont obtenues avec l'opération « `to` », c'est-à-dire en donnant seulement les directions des tangentes aux extrémités ; par contre, la liaison en trait gris est une courbe de Bézier dont les points de contrôle ont été choisis pour assurer la bonne direction des tangentes aux extrémités (les distances entre le premier (resp. second) point de contrôle et l'origine (resp. l'extrémité) de la courbe sont de l'ordre de la demi-distance entre les extrémités).

```
\def\nrec(#1)(#2){\node(#1)at(#2)[rectangle,draw]{#1#1};}
\begin{tikzpicture}[x=1mm,y=1mm,inner sep=2pt,>=stealth]
\nrec(A)(17,3)\nrec(C)(14,33)\nrec(D)(29,25)
\node[rectangle,draw,outer sep=1mm](B)at(4,20){BB};
\node[coordinate,xshift=-1.5mm]
                    (A1)at(A.north){};
\node[coordinate,xshift=-0.5mm]
                    (A2)at(A.north){};
\node[coordinate,xshift=0.5mm]
                    (A3)at(A.north){};
\node[coordinate,xshift=1.5mm]
                    (A4)at(A.north){};
\tikzstyle{every node}=[fill=white,
 scale=0.4,sloped,below,outer sep=3pt]
\draw[->](A1)to[out=90,in=-90](B);
\draw[->](A2)to[out=90,in=-90](C);
\draw[->](A3)to[out=90,in=180](D);
\draw[->](A4)to[out=90,in=-90]
(D)node[pos=0.33]{0.33}node[pos=0.66]{0.66};
\draw[->](A)to[out=0,in=0](D);
\draw[->,gray](A)..controls(32,3)and(44,25)..(D);
\end{tikzpicture}
```



Un point important est à souligner : on a choisi de placer les labels le long de la courbe sur un petit rectangle blanc pour les rendre bien visibles (options `rectangle`, `fill=white`, `inner sep=2pt` et `below`) ; si on s'en tient là, à cause de la concavité ou de la convexité de la courbe, les petits fonds blancs rectangulaires vont « mordre » légèrement sur la courbe : on a ajouté l'option `outer sep=3pt` qui éloigne le label de la courbe (un point paraît suffire, il ne faut pas trop éloigner le label de la courbe). Le rôle primordial de cette option [13.13] est d'écarter les liaisons de leur nœud de départ et de leur nœud d'arrivée (cas du nœud

BB sur la figure), mais on voit qu'elle permet aussi d'éloigner les labels des liaisons auquelles ils sont attachés.

Il semble bien que, pour des organigrammes ou des algorithmes complexes, ce sont les liaisons par segments parallèles aux axes qui assurent le plus de clarté.

3. Labels et « pins » associés aux nœuds

Il est assez évident que, pour ne pas surcharger les figures, les labels et les pins associés aux nœuds doivent être relativement « sobres » (comme les labels attachés aux liaisons). On va retrouver pour ces labels et ces pins une syntaxe rappelant (à une importante différence près) celle des labels attachés aux liaisons : les commandes de labels et de pins sont des options de la commande de nœud auxquel il sont attachés [13.9] ; elles s'écrivent :

`label={[options] u : matériel}`
`pin={[options] u : matériel}`

où `u` est la direction en degrés dans laquelle le label ou le pin doit être placé ; on peut aussi utiliser, au lieu de `u`, une des huit directions `above`, ..., `below right`.

Sur l'exemple qui suit, on a tracé préalablement en gris des rayons montrant les directions dans lesquelles les labels sont placés. On a défini un style de label (`every label`) qui place les labels sur un petit rectangle blanc dont les dimensions sont plus grandes de `inner sep` que les dimensions du matériel du label : cela peut améliorer leur lisibilité mais permet aussi de remarquer que le positionnement directionnel du label se fait de la manière suivante : à l'exception des quatre directions principales, c'est le coin le plus près du centre du nœud qui est placé dans la direction imposée. La distance du label au centre du nœud (partie visible du rayon entre le nœud et le label) est déterminée par l'option de label (défaut : 0 pt) :

`label distance=`(dimension)

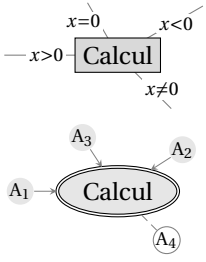
Pour cet exemple, il a fallu augmenter cette dimension pour compenser la petite valeur choisie pour `inner sep`, sinon les labels seraient trop près du nœud.

Deux difficultés sont apparues : d'une part on a été obligé de remettre à 12 le `\catcode` du caractère deux-points (séparateur entre l'angle et le matériel) car l'extension `[frenchb]{babel}` rend ce caractère actif, ce

qui donne une erreur de syntaxe [GUT, note introductive, p. 23] ; d'autre part, le caractère = n'étant pas accepté dans le matériel du label, il a fallu enfermer l'égalité dans une `\hbox` (méthode bien connue et utilisée dans d'autres situations où un caractère donné joue un rôle particulier dans une syntaxe du type mots clés).

On présente maintenant l'exemple annoncé où l'on trouvera quelques exemples de « pins » que l'on décrira ensuite.

```
\def\sc{\scriptstyle}\catcode`\:=12
\def\Nrec(#1)(#2)#3#4{\node(#1)at(#2)
[rectangle,draw,fill=gray!30,#4]{#3};}
\def\Nell(#1)(#2)#3#4{\node(#1)at(#2)[ellipse,draw,double,
                                    fill=gray!20,#4]{#3};}
\tikzstyle{every label}=[rectangle,inner
   sep=1pt,fill=white,label distance=3pt]
\tikzstyle{every pin}=[circle,inner
   sep=1pt,fill=gray!20,pin distance=3mm,
pin edge={>=stealth,<-,shorten <=1pt}]
        \begin{tikzpicture}[x=1mm,y=1mm]
\draw[thin,gray] (30,18) -- +(180:15)
       (30,18)--+(30:13)(30,18)--+(120:8)
                      (30,18)--+(-45:10);
\Nrec(a)(30,18){Calcul}
{label=left:$\sc x>0$,label=30:$\sc x<0$,
              label=120:\hbox{$\sc x=0$},
         label=-45:\hbox{$\sc x\not=0$}}
\Nell(b)(30,0){Calcul}{pin=left:$\sc A_{1}$,pin=30:$\sc A_{2}$,
                 pin=120:$\sc A_{3}$,pin={[fill=white,draw=gray,
     pin edge={densely dashed,shorten <=0.5pt}]-45:$\sc A_{4}$}}
\end{tikzpicture}
```



Les pins sont en fait constitués de deux éléments indépendants : la tête et l'épingle.

Pour les têtes, un style `every pin` a été défini : on reconnait d'abord l'option de forme circulaire des nœuds, l'absence de contour et fond gris dont le rayon dépasse de `inner sep=` le rayon du matériel.

La longueur de l'épingle est déterminée par l'option (défaut : 3 ex) :
`pin edge=`(dimension)

Les options pour sa construction sont données par la commande :
`pin edge={`options`}`
qui est encore une option du nœud d'attachement. Dans le cas présent,

on a introduit une option de type de pointe de flèche, une option de choix de flèche et l'option :
`shorten >=(dimension)`
pour avoir un retrait de la pointe de la flèche et éviter ainsi que cette pointe atteigne la partie blanche du double trait. Les options relatives aux têtes sont indépendantes des options relatives aux épingles : on peut avoir `draw=red` pour le contour de la tête et `draw=blue` (ou `blue` tout simplement) pour l'épingle ; cependant il ne faut pas oublier que la « sobriété » est un gage de lisibilité.

On peut utiliser un deuxième style de pins sur un même groupe de figures ; on peut imaginer une situation où un type de nœud représenterait un élément ayant deux propriétés différentes dont les valeurs associées seraient signalées par des pins de deux styles différents. La figure montre un deuxième type de style de pins : matériel placé sur un fond blanc entouré par un trait fin gris, l'épingle est un traitillé de même gris et elle est légèrement raccourcie (toujours pour la même raison).

CHAPITRE 6

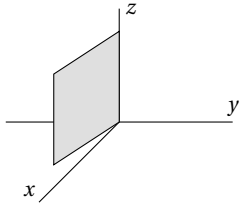
Compléments

Dans ce chapitre on va introduire des possibilités non encore abordées ou partiellement présentées dans les chapitres précédents ainsi que des compléments de syntaxe (transformations et boucles de répétition notamment).

1. Coordonnées en trois dimensions

La syntaxe des coordonnées cartésiennes en trois dimensions est du même type que celle des coordonnées en deux dimensions : `(a,b,c)` représente le point de coordonnées a suivant les x, b suivant les y et c suivant les z. Ti*k*Z représente ce point dans le plan x, y par le point `(a,b) +(`k`c,`k`c)` où $k = -1/\sqrt{2}$ [10.2.1], [34.4]. Cela produit un certain effet de perspective ; pour retrouver les notations habituelles (les x vers l'avant gauche, les y vers la droite et les z vers le haut), on est obligé d'introduire une macro.

```
\def\pt(#1,#2,#3){(#2,#3,#1)}
\footnotesize
\begin{tikzpicture}[x=1mm,y=1mm,
             z=-0.707mm,inner sep=2pt]
\draw[thin]\pt(0,0,0)--\pt(15,0,0)
              node[above left]{$x\!$};
\draw[thin]\pt(0,-15,0)--\pt(0,15,0)
                    node[above]{$y$};
\draw[thin]\pt(0,0,0)--\pt(0,0,15)
                  node[right]{$z\!$};
\draw[draw=black,fill=gray!25]
\pt(0,0,0)--\pt(0,0,12)--\pt(8,-3,12)--\pt(8,-3,0)--cycle;
\end{tikzpicture}
```



Pour les coordonnées cylindriques `\pgfpointcylindical{u}{r}{z}` représente le point de coordonnées (angle : u degrés, rayon : r unités et cote z unités).

Pour les coordonnées sphériques
`\pgfpointspherical{u}{v}{r}` représente le point de coordonnées (longitude : *u* degrés, latitude : *v* degrés et rayon : *r* unités).

Dans les deux cas, l'unité de longueur pour les coordonnées `r` et `z` est l'unité par défaut. L'utilisation pratique de ces deux types de coordonnées telles qu'elles sont définies n'est pas très aisée d'autant plus que les axes n'ont pas les orientations habituelles.

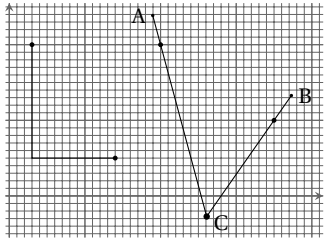
2. Intersection de droites

On a déjà vu et utilisé à la section 2.2 du chapitre 2 que les coordonnées du point d'intersection des droites définies respectivement par les points `(a,b)` et `(c,d)` d'une part et `(e,f)` et `(g,h)` d'autre part sont données par :

```
(intersection of a,b--c,d and e,f--g,h)
```

Le point d'intersection de la droite parallèle à l'axe des *y* (resp. des *x*) passant par le point `(a,b)` et la droite parallèle l'axe des *x* (resp. des *y*) passant par le point `(c,d)` est donné par [GUT 1.15] :

```
(a,b) |- (c,d) (resp. (a,b) -| (c,d))
\def\ptn(#1)(#2){\filldraw(#1)circle(#2);}
\def\Npt(#1)(#2){\node[coordinate](#1)at(#2){};}
\def\Nptd(#1)(#2)#3{\node[coordinate,
        shift={(#3:4)}](#1)at(#2){};}
\begin{tikzpicture}[x=1mm,y=1mm]
\ptn(3,20)(0.25)\ptn(14,5)(0.25)
\draw[thin](3,20)|-(14,5);
\Nptd(A)(20,20){105}
\Nptd(B)(35,10){55}
\Npt(C)(intersection of
                20,20--A and 35,10--B)
\ptn(20,20)(0.25)\ptn(35,10)(0.25)
\ptn(A)(0.15)\ptn(B)(0.15)
\ptn(C)(0.35)\draw[thin](A)--(C);\draw[thin](B)--(C);
\node[left]at(A){A};\node[anchor=180]at(B){B};
\node[anchor=155]at(C){C};
\end{tikzpicture}
```



Cette construction se généralise au cas où la première droite fait l'angle *u* degrés avec l'axe des *y* et où la deuxième fait l'angle *v* degrés avec l'axe des *x* (deuxième partie de l'exemple) :

```
(intersection a,b--A and c,d--B)
```

où `A` (resp. `B`) est un point déplacé par rapport au point `(a,b)` (resp. `(c,d)`) dans la direction u (resp. v) degrés. Dans l'exemple on a porté les noms des nœuds créés pour définir les deux droites de directions respectives u et v degrés ; on va voir maintenant la façon de faire du « lettrage » avec des macros bien adaptées.

3. Lettrages

Le lettrage consiste à placer du texte ou des formules, réduits dans la plupart des cas à un seul caractère, sur une figure faite avec Ti*k*Z ou sur toute autre figure faite par un moyen quelconque. Dans ce dernier cas, on relève les coordonnées des points à lettrer et l'on fait une figure avec Ti*k*Z ne contenant que le lettrage ; ensuite, on la superpose à la figure à lettrer : on arrive à un résultat parfait utilisant les fontes TEX sans utiliser des outils graphiques coûteux. On va distinguer plusieurs cas pour lesquels on procèdera différemment.

Les deux premiers cas ont a déjà été vus ;

— le lettrage concerne un nœud : il est alors réalisé en tant que label (ou éventuellement pin) attaché au nœud (chap. 5, section 3).

— le lettrage concerne une ligne, droite ou courbe : il est alors réalisé en tant que label associé à la ligne (chap. 5, section 2).

Le troisième cas, le plus courant, est le cas le plus élémentaire où l'on veut attacher un lettrage à un point donné représenté par ses coordonnées explicites ou par un nom de nœud vide. On utilisera la commande de nœud en donnant la direction du point par rapport au lettrage (u degrés) avec l'option `anchor=u`. Comme il est plus naturel de parler de la direction du lettrage par rapport au point plutôt que l'inverse, on propose les macros :

`\pose(x)[d]{u}{(lettrage)}`

`\poseb(x)[d]{u}{(lettrage)}` pour avoir les lettrages sur fond blanc

Les conventions concernant les paramètres de ces macros sont :

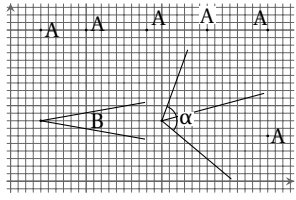
— `x` représente soit les coordonnées `a,b` soit le nom `A` du nœud vide représentant le point auquel on veut associer le lettrage,

— `u` est la direction en degrés dans laquelle on veut positionner le lettrage (entre −180 et 180),

— `d` est une correction facultative de la distance entre le point et le lettrage exprimée en points mais saisie sans unité (en général de l'ordre de ±0.5 mais parfois beaucoup plus grande) ; il y a de multiples

situations où la correction s'impose : on cite seulement le cas de certains caractères qui paraissent trop ou insuffisamment éloignés et le cas d'un lettrage se trouvant entre deux droites se coupant à son point d'attache avec un angle assez aigu.

```
\footnotesize\catcode`\:=12
\newcount\dir\newtoks\argo
\def\ptn(#1)(#2){\fill(#1)circle(#2pt);}
\newc{\posee}[3][0]{\dir=#2\advance\dir by-180
      \node[anchor=\the\dir,shift={(#2:#1pt)},
          shift={(#2:-1pt)}] at (\argu) {#3};}
\newc{\posbb}[3][0]{\dir=#2\advance\dir by-180
   \node[rectangle,fill=white,inner sep=0.5pt,
    anchor=\the\dir,shift={(#2:#1pt)},
      shift={(#2:2pt)}]at(\argu){#3};}
\def\pose(#1){\def\argu{#1}\posee}
\def\poseb(#1){\def\argu{#1}\posbb}
\begin{tikzpicture}[x=1mm,y=1mm]
\ptn(4,20)(0.5)\pose(4,20){0}{A}
\ptn(10,20)(0.5)\pose(10,20){30}{A}
\ptn(18,20)(0.5)\pose(18,20){45}{A}
\ptn(26,20)(0.5)\poseb(26,20){90}{A}
\ptn(34,20)(0.5)\pose(34,20){120}{A}
\ptn(4,8)(0.5)\draw(4,8)--+(10:14)
(4,8)--+(-10:14);\pose(4,8)[17]{0}{B}
\ptn(20,8)(0.5)\draw(20,8)--+(70:10)(20,8)--
 +(15:14)(20,8)--+(-40:12);\poseb(20,8)[4]{5}%
  {$\alpha$};\draw(20,8)+(-40:2)arc(-40:70:2);
\node at(34,6)[circle,fill,inner sep=0.3pt,
          label={[label distance=-2pt]0:A}]{};
\end{tikzpicture}
```



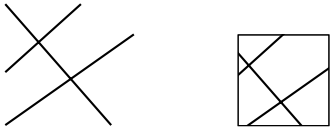
On peut aussi imaginer la possibilité de tracer le point noir et le lettrage associé avec la même commande (voir la fin de l'exemple) ; cependant il vaut mieux, tout au moins en débutant, bien séparer les types de commandes.

4. Découpage

Cette puissante opération est commune aux langages de l'environnement de TEX : Metafont, Metapost, PostScript… Elle consiste à supprimer toute la partie de la figure déjà construite située en dehors d'un contour donné [12.6]. On reprend le petit exemple donné à la section 2 du cha-

pitre 2 pour montrer l'effet de l'option `clip` de la commande `\path`. Une situation courante est la suivante : on fait une figure avec tout son savoir pédagogique et, lorsqu'elle est terminée, on réalise qu'elle est beaucoup trop grande ; on pense naturellement à la réduire (on verra plus loin comment procéder), mais on se rend compte que la partie de la figure la plus importante du point de vue pédagogique sera alors trop réduite pour y porter les lettrages indispensables ! La solution consiste à découper la partie intéressante et à supprimer l'autre partie : c'est l'option `clip` qui fait le nécessaire pour cela.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,thick](-2,-2)--(15,10)
      (-2,5)--(8,14) (-2,14)--(12,-2);
\end{tikzpicture}
\hskip2mm
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,very thin](1,1)--(13,1)
             --(13,13)--(1,13)--cycle;
\path[clip](1,1)--(13,1)--(13,13)
                      --(1,13)--cycle;
\path[draw,thick](-2,-2)--(15,10)
      (-2,5)--(8,14) (-2,14)--(12,-2);
\end{tikzpicture}
```



On rappelle que le contour de découpage n'est pas tracé par défaut. Si l'on ajoute l'option `draw`, il est tracé mais avec les options de tracé courantes ; pour le tracer en traits très fins, il a fallu le tracer par une commande de tracé préalable.

On découvrira dans la suite des utilisations de plus en plus astucieuses de cette possibilité de découpage.

5. Remplissage

Le remplissage est une opération très simple lorsque l'on a un domaine simplement connexe ; ce domaine est limité par un contour fermé, par exemple un disque limité par un cercle. Mais tout se complique dès que l'on a un domaine multiplement connexe (c'est-à-dire avec des trous comme on dit quand on n'est pas matheux pur) ou encore pire quand le domaine n'est pas connexe.

On commence avec l'exemple d'un disque possédant un trou circulaire ; on est tenté de dire : facile ! on colore le disque en entier, ensuite on

colore le petit disque intérieur en blanc … mais si on travaille en couleur et avec un papier coloré ? On pourrait se contenter de cette approche lorsqu'on travaille des cas simples en noir et blanc.

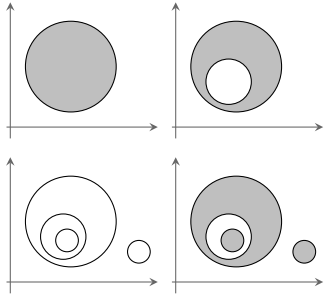
La vraie solution consiste à donner tous les contours (fermés bien entendu) à l'aide d'une commande `\path` (ou `\draw` suivant que l'on ne veut pas ou que l'on veut tracer tous les contours) avec l'option `fill` suivie éventuellement par le choix d'une couleur. Ti*k*Z détermine ensuite les points intérieurs qui seront colorés suivant une méthode que l'on peut illustrer ainsi [12.3.2] : soit un point donné ; on considère un rayon issu de ce point et allant à l'infini ; ce point est intérieur, et sera donc coloré, s'il rencontre

— un nombre pair de contours (option `even odd rule` que l'on préférera),

— autant de contours orientés dans un sens que dans l'autre (option par défaut `nonzero rule` qui est plus complète que la précédente mais plus exigente puisque les sens de tracé des contours jouent un rôle fondamental) ;

sinon le point est extérieur donc non coloré.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,fill=lightgray](8,8)circle(6);
\end{tikzpicture}
\hskip1mm
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,fill=lightgray]
                        (8,8)circle(6);
\path[draw,fill=white](7,6)circle(3);
\end{tikzpicture}
\vskip2mm
\hfill\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw](8,8)circle(6)
   (7,6)circle(3) (7.5,5.5)circle(1.5)
                    (17,4)circle(1.5);
\end{tikzpicture}
\hskip1mm
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,fill=lightgray,even odd rule] (8,8)circle(6)
 (7,6)circle(3) (7.5,5.5)circle(1.5) (17,4)circle(1.5);
\end{tikzpicture}
```



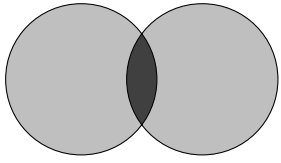
Des cas plus complexes nécessitent d'autres outils ; on verra plus loin.

6. Calques

Les dessinateurs de l'industrie et du bâtiment ont utilisé pendant longtemps les calques ; pour une maison par exemple, il y avait un plan de masse avec les murs, portes et fenêtres ; ensuite il pouvait y avoir un calque pour le sanitaire (tuyaux, éviers, baignoires, etc.), un calque pour le chauffage (tuyaux, chaudière, radiateurs, etc.), un calque pour l'appareillage électrique (câbles, etc.)… Cela évitait d'avoir des plans surchargés : le deuxième calque cité posé sur le premier donnait un plan pour les plombiers, et ainsi de suite. Cette méthode est toujours utilisée par les outils modernes de DAO. Ti*k*Z peut aussi faire des calques, chaque calque est un dessin indépendant et ce sont les superpositions de certains d'entre eux qui donnent les figures voulues [45].

On va donner la syntaxe de la méthode des calques (*layers* en anglais) sur l'exemple consistant à colorer deux disques et leur intersection.

```
\pgfdeclarelayer{union}\pgfdeclarelayer{intersection}%
\pgfsetlayers{union,intersection,main}%
\begin{tikzpicture}[x=1mm,y=1mm]
\axes(-1.5mm,-1.5mm)(16.5mm,16.5mm)
\begin{pgfonlayer}{union}
\path[fill=lightgray,even odd rule]
  (12,12)circle(10) (28,12)circle(10);
\end{pgfonlayer}
\begin{pgfonlayer}{intersection}
\path[clip](12,12)circle(10);
\fill[darkgray](28,12)circle(10);
\end{pgfonlayer}
\draw(12,12)circle(10) (28,12)circle(10);
\end{tikzpicture}
```



Les commandes situées entre
`\begin{pgfonlayer}{`(nom du calque)`}` et `\end{pgfonlayer}`
constituent le calque dont le nom est donné en début d'environnement. Toutes les autres commandes (en dehors de ces environnements) constituent le calque nommé `main` par défaut. On remarque que l'on doit, avant l'environnement `tikzpicture` déclarer les calques (excepté le calque `main` déclaré par défaut) avec la commande
`\pgfdeclarelayer{`(nom du calque)`}`
et donner la liste des calques à utiliser pour la figure avec la commande
`\pgfsetlayers{`(liste des calques)`}`

Voici le « fonctionnement » de l'exemple de la page précédente. Soit G et D les cercles de gauche et de droite ;

— le premier calque colore en gris léger $(G \cup D) \setminus (G \cap D)$ (union de G et D moins leur intersection) ; en effet, l'intersection reste non colorée car, par la règle de l'option `even odd rule`, c'est une région extérieure ;

— le deuxième calque découpe avec le cercle G l'intersection $G \cap D$ dans le cercle D coloré en gris ;

— la dernière commande trace les contours en noir (couleur par défaut) ;

On note que les commandes de déclaration et d'utilisation des calques sont situées avant l'evironnement `tikzpicture` et introduisent donc un espace si elles ne sont pas terminées par un %.

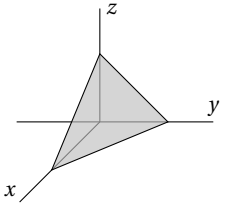
7. Couleur et ombré

On a déjà exposé un petit rappel concernant la couleur à la section 3.8 du chapitre 2 : comment obtenir les couleurs prédéfinies de l'extension `xcolor` et comment faire varier leur intensité.

7.1. Opacité

L'opacité est une propriété très importante, surtout pour des travaux à visée pédagogique, car elle permet de montrer en intensité affaiblie des parties cachées : le plus simple est de se reporter à l'exemple qui est en trois dimensions.

```
\def\pt(#1,#2,#3){(#2,#3,#1)}
\begin{tikzpicture}[x=1mm,y=1mm,
             z=-0.707mm,inner sep=2pt]
\draw[thin]\pt(0,0,0)--\pt(15,0,0)
              node[above left]{$x\!$};
\draw[thin]\pt(0,-11,0)--\pt(0,15,0)
                    node[above]{$y$};
\draw[thin]\pt(0,0,0)--\pt(0,0,15)
                   node[right]{$z\!$};
\fill[lightgray,opacity=0.65]
     \pt(9,0,0)--\pt(0,9,0)--\pt(0,0,9)--cycle;
\draw\pt(9,0,0)--\pt(0,9,0)--\pt(0,0,9)--cycle;
\end{tikzpicture}
```



Ce choix ne signifie pas que cette propriété d'opacité n'a pas d'intérêt en deux dimensions. Grâce aux calques et à un bon choix d'opacité,

on peut montrer par exemple un élément de figure et sa sous-structure correspondante.

L'opacité est déterminée par l'option [12.2.3] :
`opacity=`(nombre entre 0 et 1)
Il faut noter qu'elle est valable pour toutes les opérations de la commande où on l'a donnée : donc il faut deux commandes pour que le remplissage soit transparant sans que le contour le soit (cas de l'exemple où le contour ne doit pas être transparent pour que l'on ait bien l'impression que le triangle est posé en bais dans le trièdre).

7.2. Ombré

On a déjà obtenu un ombré, à la section 2 du chapitre 2, avec l'option [12.4] : `shade`.

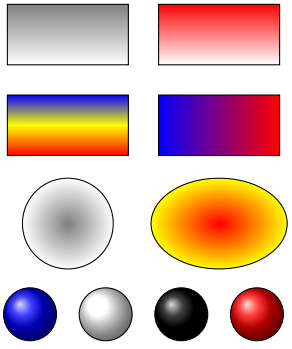
Cette option prend les valeurs par défaut des options que l'on va détailler. L'ombré est caractérisé par un aspect géométrique : les lignes de même couleur sont des droites parallèles ou des cercles concentriques ; un troisième aspect simule une boule éclairée par une source lumineuse située en haut et à gauche. Ces aspects sont choisis à l'aide de l'option [12.4.1] (défaut `axis`) :
`shading=axis`, `radial` ou `ball`

Pour le premier choix, l'angle que font les lignes d'égale couleur avec l'axe des x peut être nul (par défaut) ou valoir 90 degrés grâce à l'option [12.4.1] :
`shading angle=90`
On peut ensuite choisir la couleur du haut ou de gauche avec les options (défaut noir) [12.4.2] :
`top color=`(couleur) ou `left color=`(couleur)
et la couleur du bas ou de droite avec les options :
`bottom color=`(couleur) ou `right color=`(couleur)
suivant la direction choisie. La coloration varie régulièrement de la couleur du haut (de gauche) vers la couleur du bas (de droite) sauf si on donne la couleur du milieu par l'option :
`middle color=`(couleur)
qui doit être donnée après les deux autres options de couleur.

On constate que l'option `shade` (déjà utilisée au chap. 2, § 2) choisit les options `shading=axis`, `shading angle=0` et `top color=black`. Une option seule choisit les valeurs par défaut des options qu'elle nécessite :

par exemple, `top color=red` choisit automatiquement `shading=axis` et `shading angle=0`.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[shade](2,42)rectangle(18,50);
\draw[top color=red](22,42)rectangle(38,50);
\draw[top color=blue,bottom color=red,
middle color=yellow](2,30)
                    rectangle(18,38);
\draw[left color=blue,right color=red]
              (22,30)rectangle(38,38);
\draw[shading=radial](10,21)circle(6);
\draw[inner color=red,outer color
     =yellow](30,21)ellipse(9 and 6) ;
\draw[shading=ball](5,9)circle(3.5) ;
\draw[ball color=white](15,9)
                        circle(3.5) ;
\draw[ball color=black](25,9)
                        circle(3.5) ;
\draw[ball color=red](35,9)circle(3.5) ;
\end{tikzpicture}
```



Le deuxième choix n'a que deux options de couleurs [12.4.2] :
`inner color=`(couleur) (noir par défaut)
`outer color=`(couleur) (blanc par défaut)
et l'on a encore la même facilité : la donnée de la première couleur choisit l'option `shading=radial`.

Le troisième choix n'a qu'une option de couleur (défaut : bleu) [12.4.1] :
`ball color=`(couleur)
et, là encore, le choix de la couleur choisit l'option `shading=ball`. On note que l'on peut aussi choisir la couleur blanche ou le couleur noire.

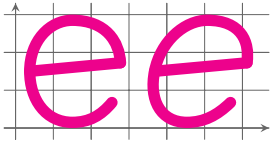
8. Transformations

On a déjà utilisé certaines transformations pour déplacer des parties de figures, par exemple des parties presque identiques mais dont on veut mettre clairement en évidence les différences. On donne la liste des transformations avec leur définition [15.3] : les coordonnées avant transformation suivies des coordonnées après transformation.

— `xshift=a` ; x, y ; $x+a$, y.
— `yshift=b` ; x, y ; x, $y+b$.

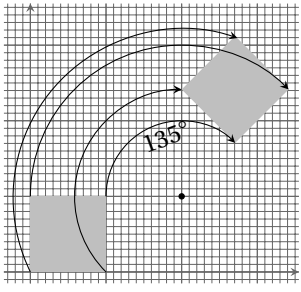
— `shift={(a,b)}` ; x, y ; $x+a$, $y+b$.
— `xscale=k` ; x, y ; kx, y.
— `yscale=k` ; x, y ; x, ky.
— `scale=k` ; x, y ; kx, ky.
— `xslant=k` ; x, y ; $x+ky$, y.
— `yslant=k` ; x, y ; x, $y+kx$.

```
\begin{tikzpicture}[x=0.5mm,y=0.5mm,
 line width=1.5mm,lightgray,cap=round]
\def\lettree{\draw (25.5,6.75)..controls+(-130:4.5)and+(0:4.5)
..(15,1.5)..controls+(180:7)and+(-90:7.5)..(3.75,15)..controls
+(90:8)and+(180:6)..(15,28.5)..
          controls+(0:9)and+(93:5.5)..
                        (27.75,17.25);
\draw (27.75,17.25)--(3.75,15);}
\lettree
\begin{scope}[xshift=15mm,xslant=0.2]
\lettree
\end{scope}
\end{tikzpicture}
```



— `rotate=u` ; rotation de u degrés autour de l'origine.
— `rotate around={u:(a,b)}` ; rotation de u degrés autour du point de coordonnées $x=a$ et $y=b$.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\fill[lightgray](0,0)rectangle(10,10);
\fill[rotate around={-135:(20,10)},
      lightgray](0,0)rectangle(10,10);
\draw[thin,->,>=stealth](0,0)
                  arc(206:71:22.3607);
\draw[thin,->,>=stealth]
                 (0,10)arc(180:45:20);
\draw[thin,->,>=stealth]
                (10,10)arc(180:45:10);
\draw[thin,->,>=stealth](10,0)
                  arc(225:90:14.1421);
\filldraw(20,10)circle(0.35);
\node at (18,18)[rotate=202]
                  {$\sc135^{\circ}$};
\end{tikzpicture}
```



— `cm={a,b,c,d,(e,f)}` ; transformation donnée par

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$$

— `reset cm` annule l'effet de la transformation courante ; « à ne pas utiliser » dit Till Tantau sans donner de raison ; pourtant c'est bien plus simple que d'écrire la transformation inverse de la transformation à annuler. Bien évidemment, si l'on a fait successivement plusieurs transformations, tout se complique et le conseil prend toute sa valeur !

9. Boucles de répétition

Dans tous les langages on trouve une syntaxe spéciale pour répéter un certain nombre de fois une opération ; pour Ti*k*Z, cette répétition se fait, avec la commande [29] :
`\foreach` (variable) `in {`(liste de valeurs)`} {`(commandes à répéter)`}`
où (variable) s'écrit comme une commande TEX sans arguments ; (liste de valeurs) a trois formes possibles :

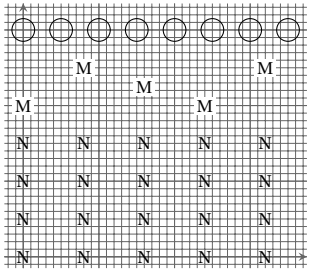
— `{m,n,...,p}` avec `m`, `n` et `p` : nombres réels ; la variable prend les valeurs de m à p avec un pas déterminé par $|n-m|$; exemples de syntaxe et valeurs prises :
`{1,2,...,6}` : 1, 2, 3, 4, 5, 6 ;
`{1,3,...,11}` : 1, 3, 5, 7, 9, 11 ;
`{0,0.1,...,0.5}` ; 0, 0,1, 0,2, 0,3, 0.4, 0,5 ;

— `{a,b,c,d,7,8,9,uv,uw,ux,uy,90,102,254}` : toutes les valeurs données dans la liste sont prises successivement (chaînes de lettres ou de chiffres) ; les éléments de la liste peuvent donc être des noms de nœuds déjà définis ; bien que le séparateur soit la virgule, les paires de coordonnées peuvent, malgré la virgule séparatrice interne, constituer des listes valables : la liste `{(0,0),(0,1),(1,1),(1,2)}` est correcte dans la plupart des cas ;

— `{a,b,9,8,...,1,2,2.125,...,2.5}` : a, b, 9, 8, 7, 6, 5, 4, 3, 2, 1, 2, 2,125, 2,25, 2,375, 2,5 ; c'est en quelque sorte un mélange des deux premières syntaxes.

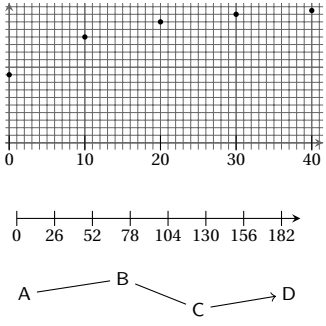
On remarque sur l'exemple suivant (page suivante) l'utilisation de deux commandes `foreach`, l'une à la suite de l'autre, pour répéter des opérations avec deux variables de répétition.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\foreach\i in{0,5,...,35}{\draw(\i,30)
                          circle(1.5);}
\scriptsize
\foreach\a in{(0,20),(8,25),(16,22.5),
     (24,20),(32,25)}{\node[rectangle,
 fill=white,inner sep=1pt] at \a {M};}
\foreach\i in{0,8,...,32}
\foreach\j in{0,5,...,15}
                {\node at(\i,\j){N};}
\end{tikzpicture}
```



L'exemple suivant utilise encore `foreach` mais avec des variables doubles et même multiples.

```
\def\ptn(#1)(#2){\fill(#1)circle(#2pt);}
\newc{\posee}[3][0]{\dir=#2\advance\dir
by-180\node[anchor=\the\dir,shift={(#2:#1pt)},
          shift={(#2:-1pt)}]at(\argu){#3};}
\def\pose(#1){\def\argu{#1}\posee}
\newcounter{xx}
\begin{tikzpicture}[x=1mm,y=1mm]
\scriptsize
\foreach\x/\y in{0/9,10/14,20/16,
  30/17,40/17.5}{\draw(\x,-1)--(\x,1);
  \pose(\x,-1){-90}{\x}\ptn(\x,\y)(1)}
\draw[>=stealth,->]
                 (0,-10)--(42.5,-10);
\foreach\x in {1,6,...,36}
            {\draw(\x,-11)--(\x,-9);}
\foreach\x in {1,6,...,36}
{\setcounter{xx}
           {(\number\x-1)*\real{5.20001}}
   \pose(\x,-11){-90}{\the\value{xx}}}
\foreach\x/\y/\l/\L in{2/-20/a/A,15/-18/b/B,25/-22/c/C,
          37/-20/d/D}{\node(\l)at(\x,\y){\textsf{\L}};}
\draw[->](a)--(b)--(c)--(d);
\end{tikzpicture}
```



L'exemple précédent débute avec une variable double ; si les deux variables croissent linéairement, il est plus intéressant de procéder en deux étapes distinctes avec des variables simples car on peut alors utiliser la propriété des `...` pour abréger la saisie de certaines listes de variables ; on remarque une difficulté introduite par l'extension `calc` : pour

multiplier par 5,2 il a fallu multiplier par 5,20001 pour obtenir les valeurs correctes.

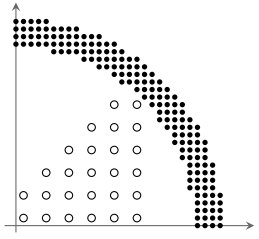
La fin de l'exemple montre l'utilisation d'une variable quadruple pour placer des nœuds en des points donnés, ayant un nom donné et un texte donné ; pour monter que les nœuds sont bien définis, on les joints par une flèche brisée. On doit être prudent en ce qui concerne l'utilisation de paires de coordonnées en tant que variables : la dernière partie de l'exemple ne fonctionne pas si l'on remplace les variables \x et \y par une variable de ce type.

Pour terminer, on va montrer comment introduire des conditions de sortie à l'intérieur des boucles de répétition. On donne deux exemples :

— le premier utilise la commande \breakforeach qui arrête la répétition lorsque la condition est vérifiée ;

— pour le second, \breakforeach ne convient pas ; on opère différemment : la boucle continue et c'est seulement l'action déclenchée à chaque pas de boucle qui s'arrête (cette méthode est plus générale et permet de traiter des cas plus complexes).

```
\def\ptn(#1)(#2){\fill(#1)circle(#2pt);}
\newcounter{cx}
\begin{tikzpicture}[x=1mm,y=1mm]
\foreach\x in {1,4,...,16}
\foreach\y in {1,4,...,16}
{\draw(\x,\y)circle(0.5);
          \ifnum\x<\y\breakforeach\fi}
\foreach\x in{0,1,...,42}
\foreach\y in{0,1,...,42}
{\setcounter{cx}{\number\x*\number\x+
                  \number\y*\number\y}
\ifnum\value{cx}<550\else\ifnum
   \value{cx}<750\ptn(\x,\y)(1)\fi\fi}
\end{tikzpicture}
```



10. Tracé de courbe direct avec plot et Gnuplot

On a déjà vu à la section 3.1 du chapitre 4 la commande pour tracer une courbe en donnant les coordonnées des points de passage.

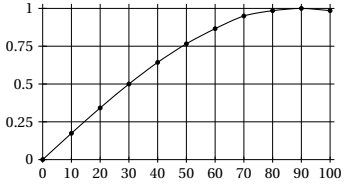
On y a aussi donné la syntaxe de la commande de tracé dans le cas où les coordonnées des points de passage sont dans un fichier ; on va préciser la syntaxe de ce fichier de coordonnées : chaque ligne

correspond à un point ; elle comprend un nombre (abscice), un espace, un nombre (ordonnée), un espace et un `i` pour terminer ; le caractère de commentaire est # [11.13.2] :

```
#Curve 0, 11 points
#x y type
0 0 i
10 0.174 i
20 0.342 i
...
90 1 i
100 0985 i
```

La courbe correspondante est la suivante :

```
\begin{tikzpicture}[x=0.38mm,y=20mm]
\tiny\foreach\x in{0,10,...,100}{\draw(\x,-0.02)--(\x,0.02);
\draw[very thin](\x,0)--(\x,1.03);
\node[anchor=90]at(\x,-0.02){\x};}
\foreach\y in{0,0.25,...,1}
        {\draw(-1.5,\y)--(1.5,\y);
\draw[very thin](0,\y)--(101.5,\y);
\node[anchor=0](a)at(-1.5,\y){\y};}
\pgfsetplotmarksize{0.7pt}
\draw plot[smooth,mark=*]
                file{t/sinus.dat};
\end{tikzpicture}
```



On termine par la troisième commande de tracé de courbe avec l'opération `plot` et le programme Gnuplot [11.12.3] :

`\draw plot[smooth,`(options)`]`
`function{`(formule en syntaxe Gnuplot)`}`

Les options (en plus des options usuelles : couleur, épaisseur des traits, traitillé, etc.) sont :

`samples=`(nombre) (défaut : 25)
permet de donner le nombre de points que l'on veut utiliser pour le tracé ;

`domain=`(nombre) : (nombre) (défaut -5 et 5)
où le premier nombre est l'abcisse de l'origine de l'intervalle du tracé et le deuxième nombre en est l'abcisse de l'extrémité ;

`parametric=true` (défaut `false`)
indique que la courbe tracée est une courbe paramétrique ; dans ce cas

il faut donner (toujours en syntaxe Gnuplot), les deux fonctions, par exemple
`\draw plot ... function{t*sin(t),t*cos{t}}`
et, bien entendu, `domain` sert à donner l'intervalle de variation du paramètre ;
`prefix=`(préfixe) et `id=`(nom)
dont la valeur et le rôle ne pourront être expliqués que plus loin.

Le tracé de la courbe à l'aide de cette commande se fait suivant un des deux processus suivants :

1. Si la possibilité de TEX de faire des appels système est activée[3], alors TEX peut arrêter sa propre tâche, lancer un autre programme par l'intermédaire d'un fichier de commande qu'il a écrit, lire le fichier de résultats écrit par ce programme, et enfin reprendre sa propre tâche avec les données lues dans ce fichier.

Dans le cas qui nous intéresse, TEX commence par chercher le fichier (préfixe)(nom)`.table` :

— s'il ne le trouve pas, alors il écrit le fichier (préfixe)(nom)`.gnuplot` ;

— ensuite, il utilise ce fichier pour lancer Gnuplot et lui faire calculer les coordonnées des points de passage et les écrire dans le fichier (préfixe)(nom)`.table` ;

— enfin, il lit ce dernier fichier et termine en traçant la courbe ;

(Dans ces notations, (préfixe) peut être un chemin (`plots/` par exemple), un nom repérant le travail en cours, etc. (le défaut est `\jobname`, c'est-à-dire le nom du fichier maître) ; (id) est un nom repérant la figure elle-même.)

— s'il trouve ce fichier d'extension `table`, alors TEX trace la courbe passant par les points dont les coordonnées sont contenues dans ce fichier (ce fichier n'est donc créé qu'une seule fois).

2. Si l'on ne dispose pas de cette possibilité (si elle n'est pas activée ou si le système d'exploitation est monotâche), TEX cherche d'abord le fichier (préfixe)(nom)`.table` ;

— s'il ne le trouve pas, il écrit tout de même le fichier (préfixe)(nom)`.gnuplot` ;

3. Il faut pour cela que le compilateur TEX soit lancé avec une option (`-enable-write18` dans le cas de MIKTEX ; `-shell-escape` dans le cas Web2C).

— à l'aide de ce fichier (qui contient en fait la commande complète avec toutes les options nécessaires) on peut alors lancer Gnuplot à la main et obtenir le fichier de coordonnées des points de passage ;

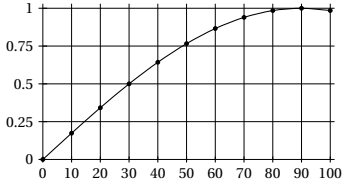
— enfin, on relance TEX en s'assurant que ce fichier de coordonnés est bien accessible, le chemin d'accès doit être conforme à celui donné dans (préfixe) ;

— s'il trouve ce fichier d'extension `.table`, alors il trace la courbe correspondante.

On déduit de ce qui a été expliqué ci-dessus que, si l'on dispose du fichier (prefixe)(nom)`.table`, on peut recompiler le ficher `.tex` sans disposer de Gnuplot.

Voici le même exemple traité par cette dernière méthode.

```
\begin{tikzpicture}[x=0.38mm,y=20mm]
\tiny\foreach\x in{0,10,...,100}
      {\draw(\x,-0.02)--(\x,0.02);
\draw[very thin](\x,0)--(\x,1.03);
\node[anchor=90]at(\x,-0.02){\x};}
\foreach\y in {0,0.25,...,1}
       {\draw (-1.5,\y)--(1.5,\y);
\draw[very thin](0,\y)--(101.5,\y);
\node[anchor=0](a)at(-1.5,\y){\y};}
\pgfsetplotmarksize{0.7pt}
\draw plot[smooth,mark=*,domain=0:100,samples=11,
    prefix=t/,id=sinus]function{sin(x*0.017453)};
\end{tikzpicture}
```



Remarque. — Sur un PC sans Gnuplot, le traitement de cet exemple signale que TEX ne trouve pas le fichier `t/sinus.table` mais crée le fichier `t/sinus.gnuplot`. Pour terminer l'exemple, faire une copie nommée `t/sinus.table` du fichier `t/sinus.dat` de l'exercice précédent, relancer le traitement (on n'a plus le message bien entendu) et l'on obtient la courbe. Pour lancer Gnuplot à la main, il faut que le fichier `t/sinus.gnuplot` soit executable (attribut x sous Unix-Linux, extension `.bat` sous Windows).

CHAPITRE 7

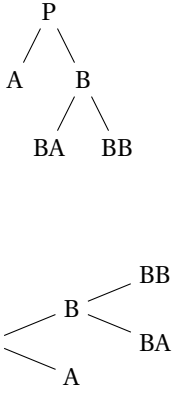
Structures arborescentes

Non, il ne sera pas question de fougères mais essentiellement de graphiques en forme d'arbres pour représenter des structures hiérarchisées. On va donner des exemples simples pour montrer ce qui peut être fait avec les commandes spécifiques de Ti*k*Z et leurs options. Ensuite, ce sera à chaque lecteur de ces lignes, en fonction de sa spécialité, d'adapter l'utilisation de ces commandes et options au style des structures utilisées dans sa discipline.

1. Arbres

On va tout de suite entrer dans le vif du sujet en proposant une première structure simple où n'apparaissent que deux commandes, [8.6] [14] :
`\node` (déja beaucoup utilisée) et `child`
pour créer le nœud « parent » et les nœuds « fils », « petit-fils », etc.

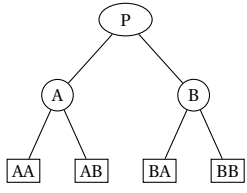
```
\begin{tikzpicture}[level distance
          =11mm,sibling distance=11mm]
\node{P}
  child{node{A}}child{node{B}
    child{node{BA}}child{node{BB}}};
\end{tikzpicture}
\vskip12mm
\begin{tikzpicture}[grow=right,
level distance=9mm,sibling distance=9mm]
\node{P}
  child{node{A}}child{node{B}
    child{node{BA}}child{node{BB}}};
\end{tikzpicture}
```



Ces commandes ont trois options :
`grow=`(angle) (défaut : -90)

donne la direction de croissance de l'arbre ; au lieu de donner un angle (en degrés), on peut donner un des 4 points cardinaux `north`,... (défaut : `south`) ou une des 4 directions `up`,...(défaut `down`) ;
`level distance=`(dimension)
détermine la distance entre les générations dans la direction de croissance ;
`sibling distance=`(dimension)
qui détermine la distance entre les points d'ancrage des descendants ; cette distance ne prend pas en compte la largeur des nœuds des descendants, la distance de séparation ni le nombre de ces descendants. En effet, on se rend tout de suite compte sur l'exemple précédent que l'on ne peut pas compléter la structure obtenue : on ne peut ajouter le petit-fils AB car la place est occupée par le petit-fils BA ; il faut donc que la distance entre descendants de même génération dépende du niveau de descendance. Cela est fait sur l'exemple suivant où l'on donne, pour chaque niveau de descendance, une valeur spécifique de l'option `sibling distance` : cela se fait à l'aide des styles `level 1`, `level 2`, etc., que l'on définit avec la commande `tikzstyle`.

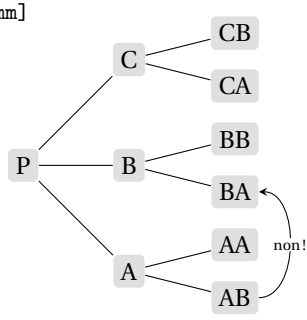
```
\tikzstyle{every node}=[ellipse,draw]
\tikzstyle{level 1}=[sibling distance=18mm,
set style={{every node}=[circle,draw]}]
\tikzstyle{level 2}=[sibling distance=9mm,
set style={{every node}=[rectangle,draw]}]
\begin{tikzpicture}[level distance=10mm]
\scriptsize
\node{\kern1ex P\kern1ex}
  child{node{A}
     child{node{AA}}
     child{node{AB}}}
  child{node{B}
     child{node{BA}}
     child{node{BB}}};
\end{tikzpicture}
```



On voit que cette méthode permet aussi de définir des styles de nœuds différents pour chaque niveau de descendance, si cela est souhaité. La syntaxe générale pour définir les styles de niveaux est :

```
\tikzstyle{(niveau)}=[sibling distance=(distance),
        set style={{every node}=[(options de nœud)]}]
```

```
\tikzstyle{every node}=[rectangle,rounded
                corners=2pt,fill=gray!25]
\tikzstyle{level 1}=[sibling distance=14mm]
\tikzstyle{level 2}=[sibling distance=7mm]
\begin{tikzpicture}[grow=right,
                        level distance=14mm]
\scriptsize
\node(p){P}
  child{node{A}
    child{node(v){AB}}child{node{AA}}}
  child{node{B}
    child{node{BA}}child{node{BB}}}
  child{node{C}
    child{node{CA}}child{node{CB}}};
\draw[>=stealth,->](v)to[out=0,in=0]
       (p-2-1)node[fill=white,pos=0.5,
        inner sep=0.5pt]{\tiny non !};
\end{tikzpicture}
```

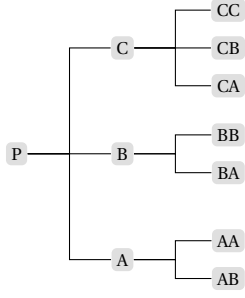


Les nœuds peuvent être nommés comme d'habitude ; s'ils ne sont pas nommés, ils ont un nom interne accessible : `p-1`, `p-1`,… `p-1-2`,… où `p` désigne le nom du parent. On peut alors, voir l'exemple précédent, tracer des liaisons entre les nœuds avec les méthodes déjà vues. On continue en chargeant la bibliothèque nécessaire avec la commande :

`\usetikzlibrary{trees}`

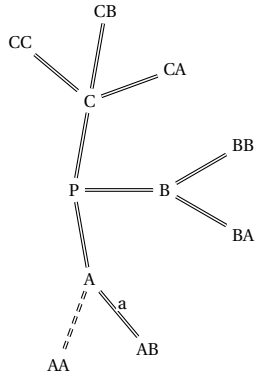
et on donne encore un exemple classique de croissance horizontale.

```
\tikzstyle{every node}=[rectangle,rounded
                corners=2pt,fill=gray!25]
\tikzstyle{level 1}=[sibling distance=14mm]
\tikzstyle{level 2}=[sibling distance=5mm]
\begin{tikzpicture}[grow=right,edge from
parent fork right,level distance=14mm]
\scriptsize
\node{P}
  child{node{A}
    child{node(v){AB}}child{node{AA}}}
  child{node{B}
    child{node{BA}}child{node{BB}}}
  child{node{C}
    child{node{CA}}child{node{CB}}
                 child{node{CC}}};
\end{tikzpicture}
```



Il y a de nombreuses possibilités pour l'option de croissance (`grow`) [28] et pour les options de liaison entre parents et descendants : forme de la liaison elle-même et points d'ancrage de la liaison sur le nœud de départ et sur le nœud d'arrivée. On va seulement présenter un dernier exemple plus orginal que les précédents.

```
\tikzstyle{level 1}=[sibling angle=80]
\tikzstyle{level 2}=[sibling angle=60]
\begin{tikzpicture}[grow cyclic,level distance=12mm]
\tikzstyle{edge from parent}
                    =[draw,double,thin]
\scriptsize
\node{P}
  [counterclockwise from=-80]
  child{node{A}
  [counterclockwise from=-110]
    child{node{AA}
     edge from parent[densely dashed]}
    child{node{AB}edge from parent
      node[fill=white,inner sep=0.5pt,
     outer sep=2pt,right,pos=0.3]{a}}}
  child{node{B}
  [counterclockwise from=-30]
       child{node{BA}}child{node{BB}}}
  child{node{C}
            [counterclockwise from=20]
      child{node{CA}}child{node{CB}}child{node{CC}}};
\end{tikzpicture}
```



Pour l'exempe précédent, on a choisi l'option `grow cyclic` qui place les descendants de même génération sur un cercle autour du parent ; l'option `sibling distance` est alors remplacée par l'option `sibling angle` ; il faut en plus donner le sens du positionnement circulaire des descendants de même génération et l'angle de départ de ce positionnement avec l'option `counterclockwise from`.

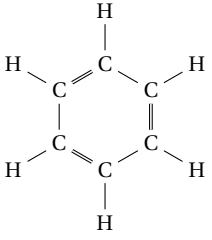
On remarque encore la manière de modifier les liaisons entre les générations par la commande de style :
`\tikzstyle{edge from parent}={double}`
et la modification exceptionnelle de l'une d'entre elle avec l'option :
`edge from parent[densely dashed]`
juste après le matériel du nœud d'arrivée de la liaison concernée. On

remarque enfin que l'on peut ajouter un label aux liaisons entre parents et descendants en plaçant, toujours à la même position, l'option :
`edge from parent node[fill=white,... right,pos=0.3]`
Dans le cas présent, il fallait « recouvrir » les options du style `every node` que l'on ne voulait pas conserver. S'il y avait eu beaucoup de tels labels à placer, il aurait fallu, bien entendu, définir un style spécifique pour les labels ou écrire une macro pour abréger la saisie.

On va terminer cette section en montrant comment l'option `grow` utilisée en tant qu'option de la commande `child` permet de composer des formules chimiques.

On commence avec la formule déployée du benzène qui rappellera à certains des temps heureux.

```
\begin{tikzpicture}[level distance=7mm,inner sep=2pt]
\def\double{edge from parent[double]}
\node{C} %haut
    child[grow=up]{node{H}}
    child[grow=-150]{node{C} %gauche
      child[grow=150]{node{H}}
      child[grow=-90]{node{C} %gauche
         child[grow=-150]{node{H}}
         child[grow=-30]{node{C} %bas
           child[grow=-90]{node{H}}
                    \double}}\double}
    child[grow=-30]{node{C} %droite
      child[grow=30]{node{H}}
      child[grow=-90]{node{C} %droite
         child[grow=-30]{node{H}}
         child[grow=-150]{node{\phantom{C}}}\double}};
\end{tikzpicture}
```

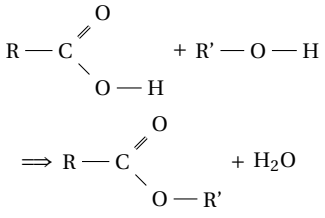


Sur cette formule déployée, on remarque que le C du bas est en quatrième génération par l'intermédiaire de la demi chaîne de gauche mais aussi en quatrième position par l'intermédiaire de la chaîne de droite d'où l'utilisation de `\phantom{C}` pour avoir une liaison parfaitement positionnée et pour éviter une superposition de deux lettres C.

On termine avec une très classique réaction de substitution (page suivante) pour laquelle on a été amené à écrire deux macros pour faciliter la saisie ; on peut peut-être faire mieux ; de plus, on ne peut affirmer qu'elles conviendront dans tous les cas : ce n'est qu'un exemple de ce

qu'il est possible de faire dans le cadre d'une discipline où la perfection est très difficile à obtenir.

```
\def\double{edge from parent[double]}
\def\bfor{\vcenter\bgroup\hbox\bgroup
  \begin{tikzpicture}[level distance=7mm,inner sep=2pt]}
\def\efor{\end{tikzpicture}\egroup\egroup}
$\bfor\node{C}
  child[grow=left]{node{R}}
  child[grow=45]{node{O}\double}
  child[grow=-45]{node{O}
    child[grow=right]{node{H}}};\efor
+\bfor\node{O}child[grow=left]
                              {node{R'}}
  child[grow=right]{node{H}};\efor$
\vskip1mm$\quad\Longrightarrow
\bfor\node{C}
  child[grow=left]{node{R}}
  child[grow=45]{node{O}\double}
  child[grow=-45]{node{O}
    child[grow=right]{node{R'}}};\efor
+\bfor\node{H${}_{2}$O};\efor$
```



2. Mindmaps

Cette dénomination, *mindmap* en anglais, est utilisée pour représenter des ensembles d'éléments en relation hiérarchisée sous la forme d'arbres ayant une structure et surtout un style particulier : les nœuds sont des cercles colorés et les liaisons ressemblent à des liaisons entre les organes des êtres vivants.

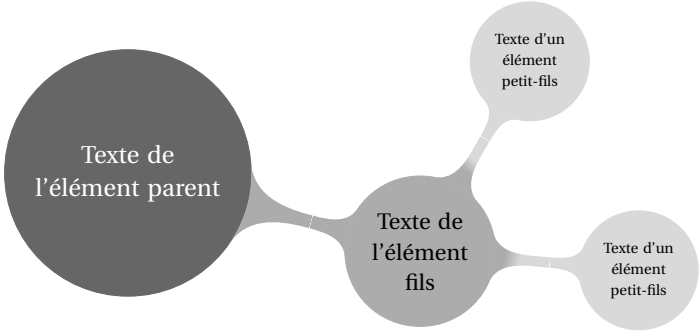
La structure de base de ces mindmaps est un cas bien particulier de la structure des arbres à croissance cyclique : on va retrouver certains aspects de la syntaxe déjà utilisée.

Il y a d'abord trois styles généraux [20] :
`mindmap`, `large mindmap` et `huge mindmap`
dont les dimensions prédéfinies conviennent bien à des impressions sur les formats A4, A3 et A2. Pour réduire les dimensions des exemples, on a été amené à introduire l'option supplémentaire `small mindmap`, sur le modèle de l'option `large mindmap` ; cette option est définie dans le fichier `smallmin.tex` qui ne fait pas partie de la distibution ; ce n'est qu'une commodité pour l'édition de ce petit manuel : il suffit de

pourcenter la commande de chargement de ce fichier et d'enlever `small` devant `mindmap` dans les options de figure et tout va fonctionner… sauf que les exemples vont déborder sur les marges.

Sans attendre, voici le premier exemple pour préciser par l'image la description précédente :

```
\input{smallmin.tex}
\begin{tikzpicture}[small mindmap,line width=0.4pt]
\tikzstyle{root concept}+=[concept,
                           concept color=darkgray!80,text=white]
\tikzstyle{level 1 concept}+=[concept color=gray!65]
\tikzstyle{level 2 concept}+=[set style={{every child}
                                    =[concept color=gray!25]}]
\node[root concept]{Texte\\de l'élément\\parent}
    child[grow=-15]{node[concept]{Texte\\de l'élément\\fils}
      child[grow=-5]{node[concept]{AA}}
      child[grow=60]{node[concept]{AB}}};
\end{tikzpicture}
```



Avant d'examiner en détail la conception de ce premier exemple, on donne la liste des styles spécifiques dont on dispose pour composer les éléments :

`concept`
pour composer tous les éléments de l'arborescence ;

`extra concept`
pour composer des éléments isolés et extérieurs à l'arborescence mais en relation avec un ou plusieurs éléments de l'arborescence :

`root concept`
contient des options supplémentaires pour composer l'élément parent ;

`level i concept`, $i = 1, 2, 3, \ldots 4$
contient des options supplémentaires pour composer les éléments des niveaux de descendants.
On peut choisit la couleur des éléments avec l'option :
`concept color=`(couleur)

Les deux derniers styles ci-dessus peuvent être modifiés ou redéfinis par les commandes adéquates ; considérons l'exemple suivant où l'on veut des couleurs différentes pour les générations ainsi que des couleurs différentes pour le contenu des éléments.

On reprend l'exemple précédent sur lequel on remarquera les points suivants avec attention :

1. Pour compendre le rôle que peut jouer l'option `line width=`*x* on prendra la valeur 2 mm et on observera le départ de la liaison du parent vers le fils : l'évasement de la liaison s'appuie sur le contour du nœud et, si le trait du contour est épais, elle s'appuie sur le cercle central du contour et non pas sur l'extérieur du contour.

Pour éviter ce problème, on ajoute, au niveau de l'option `line width`, l'option `outer sep` en donnant la demi-épaisseur choisie pour le contour du nœud. On avait déjà rencontré ce problème à la section 3 du chapitre 5, où la pointe des flèches des pins atteignait le trait blanc du contour du nœud, c'est-à-dire atteignait le milieu du trait de ce contour.

2. On a donné la couleur aux nœuds fils et petit-fils de deux manières différentes : cela entraîne que la liaison n'est pas colorée de la même manière, la coloration varie de façon progressive de la couleur du fils vers la couleur du petit-fils alors qu'elle est uniforme du parent vers le fils et a la couleur du fils ; on entre là dans des finesses du code (voir [20.3.2] pour l'explication de la différence constatée).

3. Il ne faut pas placer l'option de fonte en avant du texte ; pour comprendre, mettre `tiny` devant « Texte … petit-fils » et l'enlever de la défition du style `level 2 concept` : l'interligne n'est plus adapté à la taille des caractères : encore une finesse de syntaxe de TEX répercutée en LATEX et donc que l'on retrouve avec Ti*k*Z.

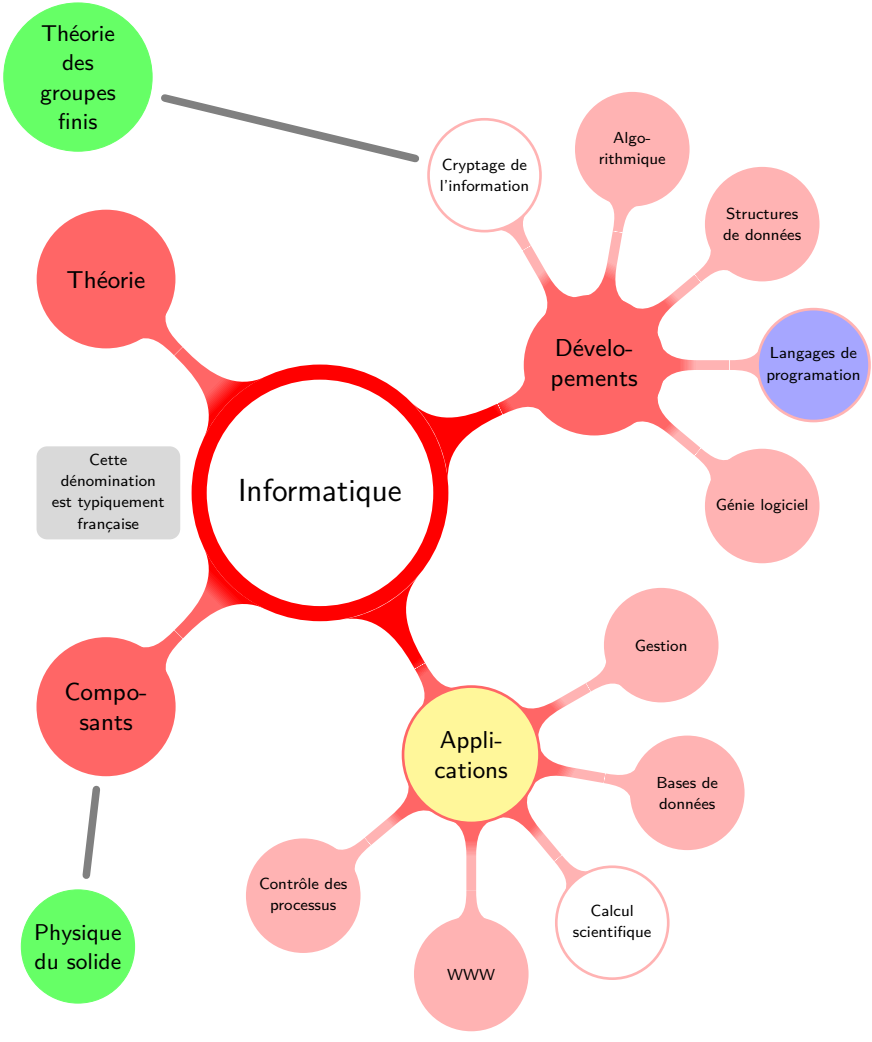
On va terminer par un exemple un peu plus important que le précédent où on a apporté quelques suppléments par rapport aux exemples de la documentation. Par défaut, le contour de l'élément et son intérieur sont colorés avec la même couleur ; on peut colorer l'intérieur ou

ne pas le colorer du tout avec la couleur « `none` » (on rappelle que colorer en blanc et ne pas colorer sont deux opérations différentes). On remarquera que la couleur de l'élément parent est donnée toujours par l'option `concept color` mais cette option est placée en tant qu'option de figure sinon, si cette option était placée dans la définition du style `root concept`, l'évasement de la liaison en direction des fils apparaîtrait en noir. L'explication de ce placement de l'option de couleur serait beaucoup trop longue ; par contre, un coup d'œil (très) attentif à la définition de l'option `concept color` dans le fichier bibliothèque correspondant situé dans `...\tex\generic\libraries\...mindmap...` permet d'en comprendre la raison.

Ces variantes sont destinées à montrer ce que l'on peut faire, ce qui ne veut pas dire qu'elles sont « visuellement compatibles » entre elles : c'est à l'utilisateur des choisir les variantes et les couleurs en gardant une certaine unité et en respectant une certaine logique. À titre de démonstration, on a placé deux éléments extérieurs qui sont en relation avec des éléments de l'arborescence et une annotation de l'élément parent… et on a aussi mis des couleurs !

Ce dernier exemple se trouve sur les deux pages suivantes.

```
\input{smallmin.tex}
\begin{tikzpicture}[small mindmap,line width=0.4pt,
                                                concept color=red]
\sffamily
\tikzstyle{root concept}+=[concept,fill=none,
                                      outer sep=1mm,line width=2mm]
\tikzstyle{level 1 concept}+=[set style={{every child}=[concept,
               concept color=red!60,line width=1pt,draw=red!60]}]
\tikzstyle{level 2 concept}+=[set style={{every child}=[
                                concept color=red!30,font=\tiny]}]
\node[root concept](n5){\Large Informatique}
  child[grow=135]{node[concept]{Théorie}}
  child[grow=-135]{node[concept](n1){Compo\-sants}}
  child[grow=-60]{node[concept,fill=yellow!50]{Appli\-cations}
    child[grow=30]{node[concept]{Gestion}}
    child[grow=-10]{node[concept]{Bases de données}}
    child[grow=-50]{node[concept,fill=none]{Calcul scientifique}}
      child[grow=-90]{node[concept]{WWW}}
      child[grow=-140]{node[concept]{Contrôle des processus}}}
    child[grow=25]{node[concept]{Dévelo\-pements}
      child[grow=120]{node[concept,fill=none](n3){Cryptage
                                                de l'information}}
      child[grow=80]{node[concept]{Algo\-rithmique }}
      child[grow=40]{node[concept]{Structures de données}}
      child[grow=0]{node[concept,fill=blue!35]{Langages
                                                 de programation}}
      child[grow=-40]{node[concept]{Génie logiciel}}};
\node[extra concept,concept color=green!60](n2)at
                                (-32mm,-60mm){Physique du solide};
\node[extra concept,concept color=green!60](n4)
                       at(-32mm,55mm){Théorie des groupes finis};
\draw[concept connection](n1)edge(n2);
\draw[concept connection](n3)edge(n4);
\node[annotation,left,fill=gray!30,text width=15mm,text centered]
       (n5.west){Cette dénomination\\est typiquement\\française};
\end{tikzpicture}
```


Théorie des groupes finis
Cryptage de l'information
Algo-rithmique
Structures de données
Théorie
Dévelo-pements
Langages de programation
Cette dénomination est typiquement française
Informatique
Génie logiciel
Gestion
Compo-sants
Appli-cations
Bases de données
Contrôle des processus
Calcul scientifique
WWW
Physique du solide

Conclusion

On commence cette conclusion en citant la plupart des possibilités disponibles qui n'ont pas été abordées dans ces pages, ou à peine évoquées. On renvoit aux sections et sous-sections des deux références citées :

— réseaux de Petri [GUT 2] [22] ;

— fonds de figure (évoqués superficiellement à la section 2 du chapitre 3 de ce petit manuel) [18] ;

— motifs pour fonds divers [21] ;

— lignes ondulées, en zig-zag, en créneaux, etc. [GUT 2.12] [26] ;

— flèches spéciales [16] ;

— automates [17] ;

— compléments pour le tracé de courbes représentatives de fonctions [23] [24] ;

— autres formes de nœuds et méthode pour en créer encore d'autres [25].

Il n'a été fait aucune allusion jusqu'à présent aux conseils rassemblés par Till Tantau [GUT 3]. Ces conseils peuvent apporter une aide précieuse, ne serait-ce que par les problèmes qu'il font découvrir et les difficultés qu'il y a pour en trouver la meilleure solution.

Le côté pédagogique doit primer et l'excès de couleurs, de fioritures et de fontes nuit certainement à la concentration des apprenants. L'auteur de ces lignes pose franchement la question : que reste-t-il des beaux livres de sciences en couleurs avec de nombreuses sortes de cadres et de nombreuses fontes dans une même page lorsque les bacheliers viennent s'assoir sur les bancs de l'université ?

Pour terminer : il semble bien que Ti*k*Z permet de faire des figures de grande qualité pédagogique dans le sens que l'on devine après avoir lu le paragraphe précédent.

Un seul regret (qui sera peut-être sans objet dans quelque temps) est l'impossibilité de pouvoir avoir accès aux coordonnées des points d'intersection des courbes de Bézier et aux directions des tangentes en ces points comme c'est possible avec Metapost ; ce devrait être possible car le tracé de ces courbes utilise aussi le paramètrage par le « temps de parcours » (cf. la commande `\pgfpointcurveattime` [34.5.2]).

Malgré cette remarque, Ti*k*Z est un outil utile, intéressant et agréable à utiliser tout en étant, du point de vue complexité, quelques crans en dessous de Metapost.

◖ Yves Soulet
ysoulet@cict.fr

Index