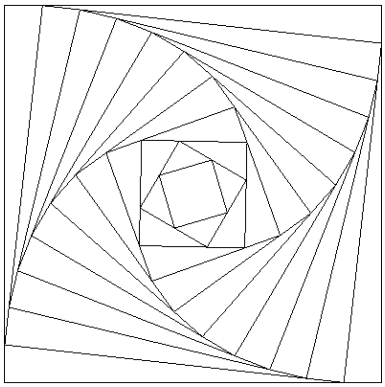
https://clx.asso.fr/spip/?Une-approche-de-metapost

LaTeX

Une approche de metapost

- Besoin d'Aide ? - Education - Transition vers le Libre... -



Date de mise en ligne : vendredi 6 décembre 2002

Avec metapost, insérez des courbes et des graphiques dans vos documents créés avec LaTex. Explications.

A qui s'adresse metapost ?

A l'heure où les logiciels de géométrie dynamique sont pleinement exploitées, en classe, par les élèves, et où bon nombre de documents utilisés par les professeurs sont rédigés avec ces logiciels dits "RIP" (Reconnu d'Intérêt Pédagogique), il est naturel de se poser la question. Metapost est un logiciel de conception de dessins et est donc, à ce titre, inutilisable avec des élèves car non visuel.

Metapost est un langage que j'appellerai de "description géométrique" issu du langage metafont. Ce dernier sert de support à la création de polices utilisables par (La)TeX. Il en reprend la plupart des primitives et en inclut de nouvelles. De ce fait, metapost est naturellement lié à (La)TeX et donc assure une intégration parfaite dans cet environnement de travail.

Un fichier, plusieurs images

Dans un fichier source metapost, on peut définir plusieurs figures. Voici la structure du contenu d'un fichier source :

```
beginfig(1); ... ... endfig;  beginfig(2); ... ... endfig; end
```

En appelant `essaidocclx.mp` ce fichier et en le compilant avec la commande `mpost essaidocclx.mp`, on obtiendra 2 fichiers : `essaidocclx.1` et `essaidocclx.2` que l'on pourra simplement inclure dans un document LaTeX en utilisant par exemple le package `graphicx` :

```
\includegraphics{essaidocclx.1}
```

La lecture des premières lignes du fichier `essaidocclx.1` dans un éditeur nous apprend que le fichier produit est au format PostScript :

```
%!PS %%BoundingBox: -11 -55 203 138  %%Creator: MetaPost
```

Types en metapost

Plusieurs types de données sont définissables et manipulables avec metapost. Toute variable doit être déclarée avec son type. Une variable non déclarée sera considérée comme du type `numeric`.

Les principaux types sont :
– `numeric` : variable numérique signée de -4096 (aussi appelé `-infinity`) à 4096 (appelée `+infinity`) ;
– `pair` : couple de valeurs numériques définissant un point ou un vecteur, ils seront

manipulés comme des affixes (ce qui rend les choses très pratiques) ;
– `path` : chemin qui pourra être une droite, un cercle ou plus généralement une courbe.
D'autres types existent, nous en rencontreront au fur et à mesure de notre besoin.

La géométrie des transformations

Qui dit géométrie dit transformations géométriques, et sur ce terrain metapost est imbattable.
Il permet tout simplement la définition de toutes les transformations affines !
Certaines sont déjà prédéfinies comme opérateurs binaires telles que :
– `shifted` : `(x,y) shifted (a,b)` sera la définition du translaté de vecteur
de coordonnées (a,b) du point (x,y). Bien sûr cet opérateur pourra s'appliquer sur un objet de type path ;
– `rotated` : `(x,y) rotated 35` sera l'image par la rotation d'angle de mesure 35
degrés du point de coordonnées (x,y), là encore cette transformation pourra s'appliquer à d'autres types.

De manière générale, metapost introduit un type `transform` qui permet de définir les
composantes de la matrice de transformation.

Quelques petites choses avant de commencer

– Metapost permet de placer des points sur une droite définie par deux points avec une notation de
coordonnées barycentriques ; par exemple, la notation `.5[A,B]`, où A et B sont des objets de
type pair, désigne le milieu du segment [AB], la notation `.7[A,B]` désigne le point M
vérifiant vec(AM)=0,7*vec(AB) ou encore : M=(1-0,7)*A+0,7*B affixement parlant.
– Certains objets de type pair sont prédéfinis, ainsi `z0, z1, z2`, etc. désignent des pairs
dont les coordonnées sont eux-mêmes prédéfinies `x0, x1, x2`, etc. et `y0, y1, y2`, etc.
Cela permet, entre autres, d'éviter d'être obligé de définir des objets de ce type.
– L'instruction `draw` dessine le chemin qu'on lui donne en argument.
– Une droite entre deux objets de type pair A et B est définie par `A--B`, le symbole
`--` indique simplement que l'on va tracer une ligne droite entre les deux points.Pour tracer un
triangle ABC, on utilisera donc `draw A--B--C--A` ou encore `draw A--B--C--cycle`
qui permet de fermer un chemin existant.
– Pour tracer de manière courbe, on utilisera `..` à la place de `--`, bien sûr il existe une infinité de manières de tracer une courbe fermée passant par trois points, par exemple,
metapost fait donc un choix par défaut, on peut le forcer à dessiner un chemin particulier mais ceci dépasse largement le
cadre de notre article.
– Metapost sait travailler avec des inconnues et résoudre des systèmes d'équations linéaires (hé oui, et
rien que pour cela c'est génial !).
– Le mot-clé `whatever` remplace n'importe quelle valeur numérique non définie et dont la
valeur ne nous intéresse pas (cf. exemple 1).
– Le mot-clé `origin` désigne l'objet pair de coordonnées (0,0).
– Comme tout langage, metapost permet de construire des boucles, de comparer des valeurs avec des opérateurs de test, d'élaborer des fonctions, etc. (cf. exemples).
– Une instruction en metapost se conclut par un point-virgule.
– A l'instar de LaTeX, le symbole % est utilisée pour les commentaires.

3 exemples commentés

– exemple 1 : la droite d'Euler

Figure classique de géométrie, on va tracer successivement le centre de gravité G, l'orthocentre H et le centre du cercle circonscrit O d'un triangle ABC.

```
beginfig(1);  pair A,B,C,O,G,H;  u=1cm;  A=origin;B=(5u,0);C=(2u,3.5u);  draw A--B--C--cycle;  % définition de O  (O-0.5[A,B]) rotated 90=whatever*(A-B);  (O-0.5[A,C]) rotated 90=whatever*(A-C);  draw O;  % définition de G  G=2/3[A,0.5[B,C]];  draw G;  % définition de H  (H-A) rotated 90=whatever*(B-C);  (H-B) rotated 90=whatever*(A-C);  draw H; endfig;7 end
```

Nous définissons les coordonnées en centimètres des sommets du triangle, voyez d'ailleurs l'utilisation d'une variable numérique `u` (car non déclarée au préalable) comme unité de mesure.
Si nous regardons la définition de O, elle se fait de manière vectorielle, nous indiquons ici que le vecteur vec(OI) (où I désignerait le milieu de [AB]) est orthogonal au vecteur vec(AB). Bien sûr, cette seule définition de O ne suffit pas et nous devons encore indiquer une autre orthogonalité pour définitivement connaître O.

[https://clx.asso.fr/spip/local/cache-vignettes/L204xH149/figure1-205a1.png]

– exemple 2 : carré tournant du document d'accompagnement du programme de la classe de Première S page 47

On part d'un carré de 10 cm de côté. Sur chaque côté, en tournant dans le même sens, on place un point situé à une distance de 1 cm de chaque sommet du carré. Et on itére.

Nous allons effectuer 10 itérations.

```
beginfig(1);  pair A,B,C,D,tmp;  u=1cm; NbIter=10;  A=origin;B=(10u,0);C=(10u,-10u);D=(0,-10u);  draw A--B--C--D--cycle;  for i=1 upto NbIter :   tmp:=A;   A:=A+u*(B-A)/abs(B-A);   B:=B+u*(C-B)/abs(C-B);   C:=C+u*(D-C)/abs(D-C);   D:=D+u*(tmp-D)/abs(tmp-D);   draw A--B--C--D--cycle;  endfor; endfig; end
```

Remarquons la forme particulière de la boucle `for` qui avec cette syntaxe impose un incrément de 1.
On remarquera aussi que la définition d'une variable se fait avec le signe d'égalité mais sa redéfinition ou sa réaffectation se fait avec le symbole `:=`.
La fonction `abs` s'applique aussi bien à des objets de type `numeric` qu'à des objets de type `pair` et définit... Je vous laisse deviner.

Mais metapost nous offre la fonction `unitvector` ainsi, on pourrait simplifier nos calculs en remplaçant : `A:=A+u*(B-A)/abs(B-A);` par `A:=A+u*unitvector(B-A);`.

[https://clx.asso.fr/spip/local/cache-vignettes/L396xH399/figure2-02bd9.png]

– exemple 3 : Tracé sur [0 ;4] de la fonction x|-Â»6x^2-x^3

```
beginfig(1);  path courbeA,courbeB;  u=1cm;  % tracé des axes  drawarrow (-.5u,0)--(6.5u,0);  drawarrow (0,-.5u)--(0,6.5u);  % les étiquettes sur les axes  for i=0 step 2 until 6 :   dotlabel.bot(decimal(i),(i*u,0));   dotlabel.lft(decimal(i*6),(0,i*u));  endfor;  % construction de 2 chemins   courbeA = courbeB = origin;  for i=0 upto 12 :   j:=i/2;   v:=(6*(j**2)-(j**3))/6;   courbeA:=courbeA--(j*u,v*u);   courbeB:=courbeB..(j*u,v*u);  endfor  draw courbeA;  draw courbeB withcolor 7red; endfig; end
```

Dans cet exemple, j'ai tracé deux courbes qui passent par les mêmes points. L'une est tracée à l'aide de segments de droite, l'autre par des segments de courbes (celle en rouge). Remarquons ici la médiocrité de la courbe rouge qui ne rend pas vraiment compte de la réalité, en effet les tracés de metapost sont faits par des courbes de Bézier dont il choisit lui-même les points de contrôle (entre deux points réels de la courbe, deux autres points ont été choisis par metapost). Les courbes de Bézier ne sont pas manifestement toujours les meilleures approximations polynômiales (cf. polynômes de Bernstein). On préférera augmenter le nombre de points et tracer de petits segments (finalement comme la calculatrice !).

L'instruction `dotlabel` permet de dessiner un point et d'y mettre une étiquette (ici dépendant de i). Attention l'instruction `dotlabel.bot(i,(i*u,0))` aurait mis la lettre i partout !!

On remarquera le tracé des axes avec une variante de l'instruction `draw` et l'utilisation d'un type non rencontré jusqu'à présent : `color`.

[https://clx.asso.fr/spip/local/cache-vignettes/L281xH279/figure3-a0a44.png]

Macros

metapost autorise la création de macro-commandes, et à l'image de LaTeX permet de les réunir dans un fichier séparée et d'en faire appel avec la commande `input` au début du fichier.

Deux types de macros sont définissables, le type `def` et le type `vardef`. On utilisera de préférence le type `vardef` qui permet d'introduire des noms variables pour les macros. Par exemple, si on veut définir les rotations en donnant leurs angles dans le nom de la macro, on pourra réaliser une unique macro `vardef` qui définit les noms `rotation60`, `rotation120`, etc. De plus, ce type de macros permet avantageusement de définir en son sein des variables locales grâce à l'instruction `save`.

Notons que toute variable déclarée dans une macro (hormis les arguments) est globale si on n'utilise pas l'instruction `save` ; remarquons aussi que les déclarations de type effacent les valeurs des variables qui avaient été éventuellement définies avec le même nom.

– exemple 1 : quelques points remarquables du triangle

En reprenant l'exemple 1 de figure , il peut être intéressant de pouvoir directement fixer ces points qui reviennent fréquemment dans nos figures :

```
vardef centredegravite (expr a,b,c) =  ((a+b+c) scaled 1/3) enddef;  vardef orthocentre (expr a,b,c) =
  save $;   pair $;   ($-a) rotated 90=whatever*(b-c);   ($-b) rotated 90=whatever*(a-c);   $ enddef;
vardef centrecerclecirconscrit (expr a,b,c) =   save $;   pair $;   ($-1/2[a,b]) rotated 90=whatever*(a-b);
  ($-1/2[a,c]) rotated 90=whatever*(a-c);   $ enddef;  vardef cerclecirconscrit (expr a,b,c) =   save $,p;
  pair $;   path p;   $=centrecerclecirconscrit(a,b,c);   p=fullcircle scaled (2*abs($-a)) shifted $;   p
enddef;  vardef centrecercleinscrit (expr a,b,c) =   save $;   pair $;   ($-a)=whatever*(b-a) rotated
(1/2*(angle(c-a)-angle(b-a)));   ($-b)=whatever*(c-b) rotated (1/2*(angle(a-b)-angle(c-b)));   $ enddef;
vardef cercleinscrit (expr a,b,c) =   save I,M,p;   pair I,M;   path p;   I=centrecercleinscrit(a,b,c);
  ((M-I) rotated 90)=whatever*(b-a);   M=whatever[a,b];   p=fullcircle scaled (2*(abs(M-I))) shifted I;   p
enddef;
```

Un exemple d'utilisation en reprenant l'exemple 1 :

```
beginfig(1);  pair A,B,C,O,G,H;  string s;  u=1cm;  A=origin;B=(5u,0);C=(2u,3.5u);  draw A--B--C--cycle;
 O=centrecerclecirconscrit(A,B,C);  G=centredegravite(A,B,C);  H=orthocentre(A,B,C);  pickup pencircle
scaled 2 bp;  for t=O,G,H :   draw t;  endfor endfig; end
```

[https://clx.asso.fr/spip/local/cache-vignettes/L201xH143/figure4-3ec81.png]

– exemple 2 : un quadrillage paramétrable

Pour les deux derniers exemples, le problème est le même, la macro ne doit pas renvoyer un chemin ou un vecteur mais bien toute une figure...
Metapost propose ainsi le type `picture` qui dessine donc l'intégralité d'une figure.
Dans notre cas, l'astuce consiste à l'appel de la macro à :

– 1/ récupérer la figure courante et tout effacer ;
– 2/ dessiner notre quadrillage ou notre cube (cf. exemple 3) ;
– 3/ enregistrer notre figure dans une autre variable ;
– 4/ tout effacer et redessiner la figure avent l'appel de la macro ;
– 5/ la macro renvoie la variable du dessin qu'elle a fait.

Les objets de type `picture` peuvent être transformés et déformés, on voit donc l'intérêt d'un tel type.

La figure courante est récupérable grâce à la variable prédéfinie `currentpicture`.

```
vardef quadrillage (expr larg,long,espace) =  % dessine un réseau espacé larg et long sans unite  % espace
en cm  save fig,qua;  picture fig,qua;   fig=currentpicture;  currentpicture:=nullpicture;   for i=0 upto
larg :   draw (i*espace,0)--(i*espace,long*espace);  endfor;  for i=0 upto long :   draw
(0,i*espace)--(larg*espace,i*espace);  endfor;  qua=currentpicture;    currentpicture:=nullpicture;  draw
fig;  qua enddef;
```

Un exemple d'utilisation :

```
beginfig(1);  picture reseau;  u=.5cm;  reseau=quadrillage(5,3,u);  draw reseau rotated 20 withcolor .5red;
endfig; end
```

[https://clx.asso.fr/spip/local/cache-vignettes/L117xH96/figure5-c9a30.png]

– exemple 3 : Dessiner un cube et le nommer

Ce dernier exemple joue avec les variables globales, puisque vous remarquerez que la commande `save` n'a pas été utilisé avec le tableau de valeurs sommetCube ainsi ce tableau une fois défini sort du cadre de notre macro et donc peut-être réutilisé dans une autre.
L'utilisation du tableau est intéressante à étudier. Les notations `sommetCube[0]` et `sommetCube0` sont identiques, on conservera la première écriture dans des boucles par exemple.

```
vardef cube (expr depart,dimarete) =   save fig,cube,chemin;   pair sommetCube[];   path chemin;   picture
fig,cube;   fig=currentpicture;   currentpicture:=nullpicture;   sommetCube0=depart;
  sommetCube1=sommetCube0 shifted (dimarete,0);   (sommetCube3-sommetCube0)=(sommetCube1-sommetCube0) scaled
.5 rotated 35;   (sommetCube2-sommetCube1)=(sommetCube0-sommetCube1) scaled .5 rotated (-145);     for i:=0
upto 3 :    sommetCube[i+4]=sommetCube[i] shifted (0,dimarete);   endfor   draw
sommetCube0--sommetCube1--sommetCube2--sommetCube6--   sommetCube5--sommetCube4--cycle;   draw
```

```
sommetCube1--sommetCube5;  draw sommetCube0--sommetCube3--sommetCube7 dashed evenly;  draw
sommetCube2--sommetCube3 dashed evenly;  draw sommetCube4--sommetCube7--sommetCube6;  cube=currentpicture;
  currentpicture:=fig;  cube enddef;  vardef nommecube =  label.llft("A", sommetCube0);  label.lrt("B",
sommetCube1);  label.rt("C", sommetCube2);  label.lft("D", sommetCube3);  label.ulft("E", sommetCube4);
  label.ulft("F", sommetCube5);  label.urt("G", sommetCube6);  label.ulft("H", sommetCube7); enddef;
```

Un exemple d'utilisation :

```
beginfig(1);  picture lecube;  pair A,H,F,C;  lecube=cube(origin,5cm);  draw lecube;  nommecube;
 A=sommetCube0;  H=sommetCube7;  F=sommetCube5;  C=sommetCube2;  pickup pencircle scaled 1bp;  draw A--F--C;
 draw H--F;  draw A--H--C--cycle dashed evenly; endfig; end
```

[https://clx.asso.fr/spip/local/cache-vignettes/L308xH281/figure6-859fc.png]

Netographie

– http://melusine.eu.org/syracuse/met...
– http://www.math.jussieu.fr/~zoonek/...
– http://cm.bell-labs.com/who/hobby/M...

Post-scriptum :

Ce n'est, là encore, qu'une approche de metapost.
On pourra dans d'autres articles montrer l'interaction réelle avec (La)TeX, des courbes paramétrées et peut-être aussi la possibilité de faire des mng (images animées).

[Impossible de lire la video]