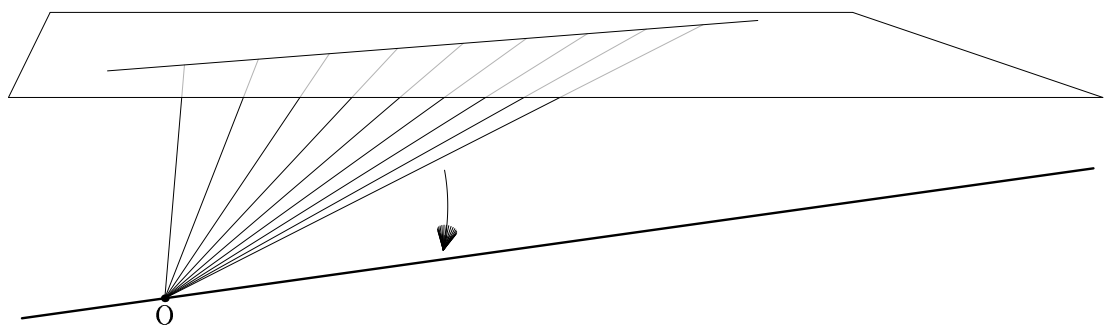
Un package 3D pour MetaPost

Xavier Caruso

31 décembre 2001

Table des matières



`3d.mp` est un ensemble de macros permettant de créer de façon relativement simple des figures en trois dimensions avec MetaPost. En particulier, comme nous allons le voir, il s'occupe tout seul de dessiner différemment les arêtes cachées et permet de changer d'angle de vue par une simple commande.

1 Comment utiliser ce package ?

Pour utiliser les macros définies dans ce package, il suffit d'inclure dans vos fichiers MetaPost le fichier `3d.mp` au moyen de la commande suivante :

```
input 3d.mp;
```

Signalons tout de suite qu'afin d'éviter les risques de conflits, les noms des variables globales utilisées ainsi que ceux des macros définies[1] commencent tous par le préfixe `td`.

2 Deux moyens de se repérer dans l'espace

2.1 Le type `td_coords`

`3d.mp` définit un nouveau type de variable qui porte le nom de `td_coords`[2]. Celui-ci consiste en des triplets de `numeric` que l'on peut additionner et multiplier par des scalaires. Il est possible d'accéder aux composantes d'un `td_coords` via les fonctions `td_x`, `td_y` et `td_z`. Pour ses propres besoins, le package `3d.mp` définit quelques opérations sur les variables de ce nouveau type. Il est peut-être utile de les présenter :

- `td_pdtscal(u,v)` : retourne le produit scalaire de `u` et `v`
- `td_pdtvect(u,v)` : retourne le produit vectoriel de `u` par `v`
- `td_norme(u)` : retourne la norme du vecteur `u`
- `td_unite(u)` : retourne le vecteur unitaire qui a la même direction et le même sens que `u`
- `td_projnorm(u,n)` : retourne le projeté du vecteur `u` sur un plan normal au vecteur `n`

2.2 Repérage des plans

Un plan P de l'espace est repéré par trois variables de type `td_coords` que l'on désignera par la suite par les lettres `i`, `j` et `o` qui correspondent respectivement à deux vecteurs qui forment une base de notre plan P et un point O qui appartient à P.

Un point de P pourra être repéré simplement par la donnée d'une variable de type `pair` une fois connu le plan P. On dispose ainsi d'un second moyen pour repérer les points de l'espace : un point M de l'espace est repéré par trois variables de type `td_coords` et une variable de type `pair`, les trois premières variables servant à déterminer un plan P auquel M appartient et la dernière servant à préciser où se situe M sur ce plan.

Le fichier `3d.mp` possède une macro permettant de déterminer la *position absolue* d'un point M à partir de la donnée des quatre variables décrites précédemment. Il s'agit de la fonction `td_ddtotd` qui prend pour argument dans l'ordre `i`, `j`, `o` puis `u` où `u` est la variable de type `pair` qui sert à repérer la position de M dans la plan.

Il existe ici aussi quelques macros permettant d'effectuer des opérations sur les vecteurs lorsqu'ils sont repérés de la façon précédente. Il s'agit de :

[1] Liste consultable en annexe.

[2] Il s'agit en fait simplement d'une redéfinition du type `color`.

- `td_projdir(i,j,o,m)` donne les coordonnées dans le plan défini par `i`, `j` et `o` du projeté orthogonal du point de position absolu `m` dans ce plan
- `td_pcentdir(i,j,o,c,m)` donne les coordonnées dans le plan défini par `i`, `j` et `o` de l'image du point de position absolue `m` par la projection de centre le point défini par `c` sur le plan évoqué précédemment

3 Fonctions de base

3.1 Segments

L'appel de la fonction `td_ligneabs(a,b)` permet de tracer un segment reliant la point de position absolue `a` à celui de position absolue `b`. Pour illustrer cela regardons le code suivant qui dessine les douze arêtes d'un cube :

```
td_beginfig(1);
  td_ech:=3cm;
  td_coords A,B,C,D,E,F,G,H;
  A:=(0,0,0); B:=(0,1,0); C:=(1,1,0); D:=(1,0,0);
  E:=(0,0,1); F:=(0,1,1); G:=(1,1,1); H:=(1,0,1);
  td_ligneabs(A,B); td_ligneabs(B,C); td_ligneabs(C,D); td_ligneabs(D,A);
  td_ligneabs(E,F); td_ligneabs(F,G); td_ligneabs(G,H); td_ligneabs(H,E);
  td_ligneabs(A,E); td_ligneabs(B,F); td_ligneabs(C,G); td_ligneabs(D,H);
td_endfig;
```

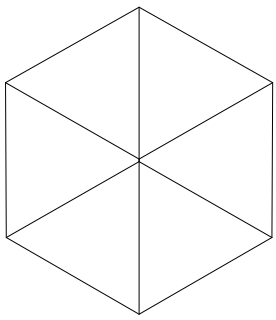


Figure 1

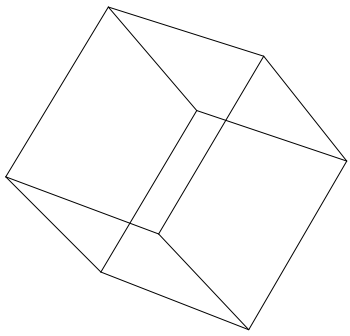


Figure 2

Le résultat produit est celui de la figure 1. Il est nécessaire de faire quelques commentaires. Tout d'abord l'emploi des macros habituelles `beginfig` et `endfig` a été remplacé par `td_beginfig` et `td_endfig`. Elles fonctionnent de la même manière à l'exception qu'elles font certaines choses en plus comme la mise à jour de certaines variables globales dont se servent les autres macros du package. Il est également important de noter que les macros appelées pour faire des dessins comme `td_ligneabs` ne dessinent en fait pas réellement sur le fichier PostScript final mais plutôt dans deux variables globales de type `picture` répondant au nom de `td_aretes` et `td_aretescachees` et c'est seulement à l'appel de la macro `td_endfig` que celles-ci sont effectivement dessinées. Il est toutefois possible de récupérer le dessin déjà constitué par l'intermédiaire de la macro `td_image` qui retourne ce-dit dessin.

Une autre remarque est l'apparition de la variable globale `td_ech`. Il s'agit simplement d'une variable de type `numeric` qui représente la longueur des vecteurs unitaires.

On remarquera que malheureusement il est nécessaire de déclarer toutes les variables de type `td_coords` qui apparaissent. En effet, si cela n'est pas fait, MetaPost va croire bêtement qu'il s'agit de variables de type `pair` et va produire une erreur.

Il est assez souvent commode d'utiliser le raccourci

```
td_ligneabs(A,B,C,D,A);
```

qui est équivalent à

```
td_ligneabs(A,B); td_ligneabs(B,C); td_ligneabs(C,D); td_ligneabs(D,A);
```

Ainsi le code que nous avons fourni peut-être un peu raccourci, devenant ainsi même plus lisible.

Il existe une autre façon de tracer des segments qui utilise le repérage relatif des points dans l'espace. Elle se concrétise via la commande `td_ligne(i,j,o,m1,m2,...)` qui permet de tracer les segments M_iM_{i+1} où les `m.i` sont des variables de type `pair` qui correspondent aux coordonnées des points M_i dans le plan repéré par `i`, `j` et `o`. Il est également possible d'utiliser la macro `td_ligne_(m1,m2,...)` qui utilise les variables globales `td_i_enc`, `td_j_enc` et `td_o_enc` pour savoir dans quel plan les segments doivent être tracés. Par défaut, `td_i_enc` est initialisé à `(1,0,0)`, `td_j_enc` à `(0,1,0)` et `td_o_enc` à `(0,0,0)`, mais bien entendu il est possible de les changer. Toutefois, un appel à `td_beginfig` rendra à ces variables leur valeur d'origine.

3.2 Changer d'angle de vue

Le résultat de la figure 1 n'est pas extraordinaire. En effet, l'on n'arrive pas à distinguer grand chose tant les arêtes sont les unes sur les autres. Pour palier à cela, il faudrait regarder le cube depuis un autre endroit. Ceci peut se faire à l'aide de la commande `td_anglevue(oeil,direction,angle)`. `oeil` est une variable de type `td_coords` qui correspond à la position de l'œil qui regarde la figure. `direction` est également une variable de type `td_coords` qui correspond à un vecteur. La direction de celui-ci fournit la direction du regard alors que sa norme fournit le facteur d'échelle avec lequel l'œil est capable de voir. Quant à `angle`, il s'agit d'une variable de type `numeric` qui correpond à l'angle (exprimé en degré) duquel est tourné l'œil. Un angle de 0 degré correspond à un œil dont le bas est dirigé vers les z négatifs.

Par exemple, la figure 2 est obtenue simplement en rajoutant au début du code présenté précedemment la ligne

```
td_anglevue((-10,-8,-3),(10,7,2),30);
```

Il est important de noter que la fonction `td_anglevue` modifie des variables globales qui sont ensuite réutilisées par les macros de dessin telles `td_ligneabs`. Il est donc fortement conseillé de faire appel à la macro `td_anglevue` (et de même par exemple de définir la valeur de `td_ech`) avant de commencer à dessiner quoi que ce soit. Par défaut la position de l'œil est mise à `(-60,-60,60)`, la direction du regard est à `(50,50,-50)` et l'angle de l'œil est de 0 degré. Cela correspond plus ou moins à une perspective isométrique.

Finalement, il est peut-être mieux de placer l'œil suffisamment loin de la figure pour ne pas trop altérer les droites parallèles, cela pouvant paraître choquant au premier abord.

3.3 Courbes

Il est possible de tracer des courbes pourvu que celles-ci soient contenues dans un plan. Ceci se fait simplement par l'appel de la fonction `td_courbe(i,j,o,p)`. Les `td_coords` `i`, `j` et `o` servent à préciser le plan dans lequel la courbe doit être tracée, alors que le `path` `p` est précisément la courbe à tracer. Là encore, il y a la macro `td_courbe_(p)` qui trace le `path` `p` dans le plan courant défini par les variables globales `td_i_enc`, `td_j_enc` et `td_o_enc`. Par exemple, la commande

```
td_courbe_(fullcircle scaled 5);
```

trace un cercle de diamètre 5 et de centre l'origine du repère dans le plan horizonral (si l'on n'a rien redéfini).

Voici un code permettant de dessiner une figure ressemblant à une sphère :

```
td_beginfig(3);
  td_anglevue((0,50,0),(0,-200,0),85);
  td_precision:=50;
  td_option(withcolor 0.6white);

  for t=0 step 5 until 360:
    td_j_enc:=(0,cosd(t),sind(t));
    td_courbe_(fullcircle);
  endfor
td_endfig;
```

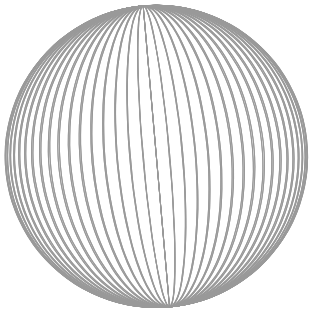


Figure 3

Là encore, il nous faut faire quelques commentaires. Tout d'abord, on remarque l'apparition de la variable globale `td_precision`. Celle-ci indique le nombre de points qui vont être considérés lors du tracé d'une courbe. Par défaut, elle vaut 500 mais ici il a été préférable de la réduire étant donné que le nombre de courbes à tracer n'est pas petit. Une grande précision fait rapidement augmenter le temps de calcul et la taille du fichier produit.

Il y a ensuite l'appel à la macro `td_option`. Ceci permet de définir avec quel style on souhaite faire les tracés ultérieurs. On notera qu'il est inutile de faire des appels à `pickup` ou à `drawoptions`, ceci n'affectant que le comportement de la macro `draw` qui n'est point utilisée dans le package `3d.mp`.

3.4 Flèches

Il existe tout un tas de macros pour dessiner des flèches soit au bout de segments, soit au bout de courbes. La liste exhaustive est la suivante :

- `td_ligneflecheabs`
- `td_lignefleche`
- `td_lignefleche_`
- `td_ligneflecheiabs`
- `td_ligneflechei`
- `td_ligneflechei_`
- `td_ligneflechesabs`
- `td_lignefleches`
- `td_lignefleches_`
- `td_courbefleche`
- `td_courbeflechei`
- `td_courbefleches`

Les fonctions précises de ces macros s'intuitent très facilement. Il suffit pour cela de savoir que `fleche` permet d'ajouter une flèche au bout, `flechei` une flèche au début et `fleches` une au bout et une au début.

Des variables globales permettent de contrôler la taille et l'allure des flèches. Il s'agit de `tdfleche_long` qui renferme la longueur *longitudinale* de la flèche, de `tdfleche_norm` qui renferme la longueur *normale* de la flèche et finalement de `tdfleche_nb` qui donne le nombre de petits segments qui partent du sommet et qui vont former la flèche[3]. Par défaut `tdfleche_long` est initialisé à 0.6, `tdfleche_norm` à 0.2 et `tdfleche_nb` à 25. Un appel à `td_beginfig` reinitialise les valeurs de ces constantes.

Et voici un joli trièdre direct :

```
td_beginfig(4);
  td_coords O,I,J,K;
  O:=(0,0,0); I:=(1,0,0);
  J:=(0,1,0); K:=(0,0,1);
  td_ligneflechesabs(4I,O,4J);
  td_ligneflecheabs(O,4K);
td_endfig;
```

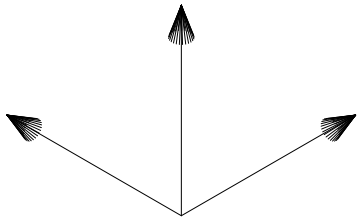


Figure 4

4 Surfaces

Un certain nombre de surfaces *planes* peuvent être définies. L'intêret de définir une surface, plutôt que de tracer les arêtes qui constituent son bord, est que l'ordinateur saura par la suite que la partie qui se trouve à l'intérieur de ces arêtes est composée de matière. Ainsi les tracés qui seront cachés à l'œil par cette matière vont être dessinés de façon différente. Mais voyons tout de suite comment cela fonctionne.

4.1 Surfaces rectangulaires, plans

Le premier type de surface que l'on peut définir sont les surfaces rectangulaires. Ceci se fait grâce à la commande `td_nouveauplan(i,j,o,m,M)`. Les premiers paramètres `i`, `j` et `o` permettent de désigner le plan de travail. `m` correspond aux coordonnées dans le plan précédent du point inférieur gauche du rectangle tandis que `M` correspond aux coordonnées du point supérieur droit. En réalité chaque surface est repérée par un entier qui lui est affecté lors de sa création. Ainsi l'appel à la fonction précédente retourne une valeur de type `numeric` qui est le numéro de la surface créée. On n'a souvent que faire de cette valeur mais afin d'éviter une erreur de compilation, il est bon d'utiliser la tournure suivante :

```
whatever=td_nouveauplan(i,j,o,m,M);
```

Il existe également la fonction `td_nouveauplan_` qui utilise les valeurs de `td_i_enc`, `td_j_enc` et `td_o_enc` pour se repérer.

Il est bien entendu également possible de dessiner ces surfaces. C'est en fait fort simple : pour dessiner la surface repérée par le numéro `i`, il suffit d'entrer la commande

```
td_surface(i);
```

[3] On remarquera en particulier que le design des flèches en trois dimensions n'a rien à voir avec celui des flèches en deux dimensions... Il ne faut pas mélanger les torchons et les serviettes, voyons !

Un appel à `td_surfaces` dessinera toutes les surfaces définies.

Regardons le code qui suit.

```
td_beginfig(5);
  td_anglevue((60,-30,30),(-30,15,-15),0);
  td_ech:=0.5cm;

  td_coords O,I,J,K;
  O:=(0,0,0); I:=(1,0,0); J:=(0,1,0); K:=(0,0,1);
  whatever=td_nouveauplan(I,J,O,(0,0),(10,10));
  whatever=td_nouveauplan(J,K,O,(0,0),(10,10));
  td_surfaces;
  for i=1 upto 9: td_ligneabs((-8,i,12),(12,5,-5)); endfor
td_endfig;
```

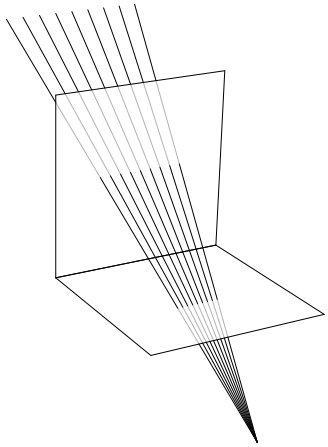


Figure 5

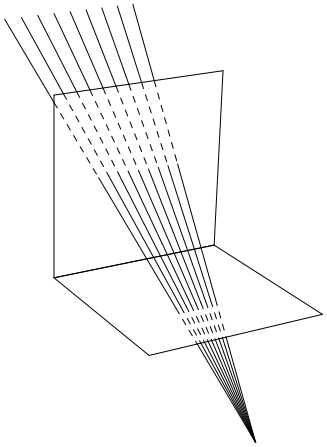


Figure 6

Il produit le dessin de la figure 5. On voit que par défaut les arêtes cachées sont tracées en gris clair. Il est cependant tout à fait possible de modifier ce comportement. Ceci se fait grâce à la commande `td_cachees`. Par exemple la figure 6 a été obtenu en rajoutant simplement au début du code précédent la ligne

```
td_cachees(dashed evenly scaled 0.8);
```

Il est aussi possible de masquer totalement les arêtes cachées grâce à la commande

```
td_cachees(withcolor background);
```

ou encore mieux (car dans ce cas les arêtes ne sont pas du tout dessinées) en utilisant les macros

```
td_masquees;
td_nonmasquees;
```

Il faut noter qu'un appel à `td_beginfig` redonne le style par défaut aux arêtes cachées.

4.2 Surfaces polygonales

Les surfaces polygonales planes se définissent avec `td_nouvellesurf(i,j,o,m1,m2,...)`. Là encore, `i`, `j` et `o` servent à définir le plan dans lequel la surface va vivre tandis que les `m.i` correspondent aux coordonnées dans ce plan des sommets du polygone délimitant notre surface. Là encore, l'appel à cette fonction retourne une valeur qui correspond au numéro de la surface et si l'on ne veut pas le garder l'utilisation de `whatever` est un bon compromis. Là encore, on peut utiliser la macro `td_nouvellesurf_` qui permet de se dispenser de la donnée de `i`, `j` et `o`.

Le comportement de ces macros n'est pas du tout garanti dans les cas suivants (NDLR : remarquez que la réciproque n'est pas énoncée...) :
- la surface définie a une aire nulle
- la surface définie n'est pas un polygone convexe
- les sommets de ce polygone ne sont pas donnés dans l'ordre

On peut par exemple reprendre l'exemple précédent en tronquant quelque peu le plan vertical :

```
td_beginfig(7);
  td_anglevue((60,-30,30),(-30,15,-15),0);
  td_ech:=0.5cm; td_masquees;
  td_coords O,I,J,K;
  O:=(0,0,0); I:=(1,0,0);
  J:=(0,1,0); K:=(0,0,1);
  whatever=td_nouveauplan(I,J,O,(0,0),(10,10));
  whatever=td_nouvellesurf(J,K,O,
           (0,0),(10,0),(10,10));
  td_surfaces;
  for i=1 upto 9:
    td_ligneabs((-8,i,12),(12,5,-5));
  endfor
td_endfig;
```

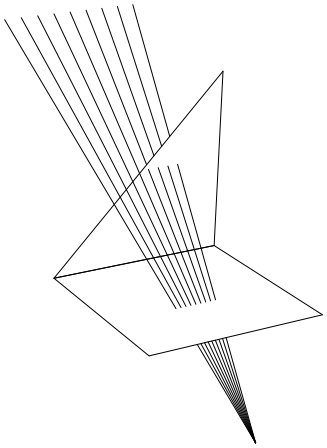


Figure 7

On peut aussi définir des surfaces polygonales en donnant les sommets en position absolue. La syntaxe est alors `td_nouvellesurfabs(m1,m2,...)` où les `m.i` sont ce coup-ci des `td_coords`. Cependant, il n'est pas possible de faire n'importe quoi, la surface définie doit être plane. Si les points `m.i` ne sont pas coplanaires, ceux-ci seront projetés sur le plan défini par les premiers points.

Notez à ce propos qu'il n'existe aucune macro permettant de calculer les `i`, `j` et `o` définissant un certain plan lorsque celui-ci est donné par exemple par un point et une normale ou par trois points non alignés... Mais il ne tient qu'à vous de les créer. On peut par exemple écrire :

```
def normtodir(expr i,j,n) =
  save u; td_coords u; u:=td_unite(n);
  if (td_x(u)=0) and (td_y(u)=0): j:=(0,1,0); else:
    j:=td_unite(td_projnorm((0,0,-1),u));
  fi
  i:=td_unite(td_pdtvect(u,j));
enddef;

def pointstodir(expr i,j,o,a,b) =
  i:=td_unite(a-o);
  j:=td_unite(td_projnorm(b-o,i));
enddef;
```

Je ne sais pas trop s'il est mieux de faire les quelques tests qui s'imposent (comme vérifier que le vecteur `n` est non nul ou que les points `o`, `a` et `b` ne sont pas alignés) avant de faire le calcul proprement dit. Cela permettrait sans aucun doute d'avoir un message d'erreur un peu plus explicite lors de la compilation mais ralentirait sans doute aussi beaucoup le temps nécessaire à celle-là. L'attitude choisie dans toutes les macros du package `3d.mp` est de ne faire aucun test.

Mais revenons à nos moutons.

Donnons un exemple d'utilisation de la macro `td_nouvellesurfabs`. Elle peut servir à tracer de façon simple un cube plein, un résultat à comparer avec la figure 2.

```
td_beginfig(8);
  td_anglevue((-10,-8,-3),(10,7,2),30);
  td_ech:=4cm;
  td_coords A,B,C,D,E,F,G,H;
  A:=(0,0,0); B:=(0,1,0);
  C:=(1,1,0); D:=(1,0,0);
  E:=(0,0,1); F:=(0,1,1);
  G:=(1,1,1); H:=(1,0,1);
  whatever=td_nouvellesurfabs(A,B,C,D);
  whatever=td_nouvellesurfabs(E,F,G,H);
  whatever=td_nouvellesurfabs(A,B,F,E);
  whatever=td_nouvellesurfabs(B,C,G,F);
  whatever=td_nouvellesurfabs(C,D,H,G);
  whatever=td_nouvellesurfabs(D,A,E,H);
  td_surfaces;
td_endfig;
```

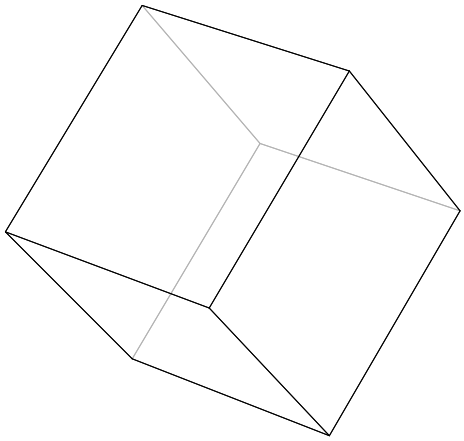
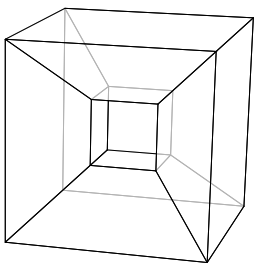


Figure 8

Et même des figures plus compliquées comme un cube troué :



obtenu par le code suivant :

```
td_beginfig(9);
  td_anglevue((20,-60,20),(-10,30,-10),0);
  td_ech:=0.6cm;

  td_coords A,B,C,D,E,F,G,H,Ap,Bp,Cp,Dp,Ep,Fp,Gp,Hp;
  A:=(0,0,0); B:=(0,9,0); C:=(9,9,0); D:=(9,0,0);
  E:=(0,0,9); F:=(0,9,9); G:=(9,9,9); H:=(9,0,9);
  Ap:=(3,3,3); Bp:=(3,6,3); Cp:=(6,6,3); Dp:=(6,3,3);
  Ep:=(3,3,6); Fp:=(3,6,6); Gp:=(6,6,6);  Hp:=(6,3,6);

  whatever=td_nouvellesurfabs(A,B,C,D);
  whatever=td_nouvellesurfabs(E,F,G,H);
  whatever=td_nouvellesurfabs(A,B,F,E);
  whatever=td_nouvellesurfabs(C,D,H,G);
  whatever=td_nouvellesurfabs(Ap,Bp,Cp,Dp);
  whatever=td_nouvellesurfabs(Ep,Fp,Gp,Hp);
  whatever=td_nouvellesurfabs(Ap,Bp,Fp,Ep);
  whatever=td_nouvellesurfabs(Cp,Dp,Hp,Gp);
  whatever=td_nouvellesurfabs(D,A,Ap,Dp);
  whatever=td_nouvellesurfabs(H,E,Ep,Hp);
  whatever=td_nouvellesurfabs(B,C,Cp,Bp);
  whatever=td_nouvellesurfabs(F,G,Gp,Fp);
  whatever=td_nouvellesurfabs(A,E,Ep,Ap);
  whatever=td_nouvellesurfabs(D,H,Hp,Dp);
  whatever=td_nouvellesurfabs(B,F,Fp,Bp);
  whatever=td_nouvellesurfabs(C,G,Gp,Cp);
  td_surfaces;
td_endfig;
```

4.3 Surfaces définies par une courbe

Finalement, il est possible de définir une surface par une courbe plane. La syntaxe pour faire cela est `td_nouvellesurfc(i,j,o,p)` où `i`, `j` et `o` sont comme d'habitude les `td_coords` qui servent à repérer le plan de travail et où `p` est la variable de type `path` définissant la courbe dans ce plan. Là encore, on peut utiliser la fonction `td_nouvellesurfc_(p)`. Là encore, le comportement de la macro n'est pas défini si la surface délimitée par la courbe est d'aire nulle ou n'est pas convexe.

Signalons qu'en fait une telle surface n'est autre qu'une surface polygonale avec un grand nombre de côtés. ce nombre est précisé dans la variable globale `tdsurf_precision` et vaut par défaut 100.

Un autre comportement particulier de ce genre de surfaces est à décrire. En effet, elles ne sont par défaut pas tracées par l'appel de la fonction `td_surfaces`. Ceci simplement parce qu'un polygône à 100 côtés c'est certes une bonne approximation de notre surface mais peut-être en fait pas suffisamment précise pour donner lieu à une jolie courbe régulière. En fait, le fait qu'un contour de surface soit tracé ou non par la commande `td_surfaces` est déterminé par le tableau global de booléens `tdsurf_dessin[]`. Par défaut une surface créée par l'intermédiaire de `td_nouvellesurfc` renseigne ce tableau en mettant un `false` à l'indice adéquat, tandis que les appels à `td_nouvellesurf` et `td_nouveauplan` positionne un `true` dans ce tableau. Ainsi un moyen

de faire en sorte que `td_surfaces` dessine effectivement les surfaces définies à partir de courbe est de les déclarer par la commande

```
tdsurf_dessin(td_nouvellesurfc(i,j,o,p)):=true;
```

mais il est sans doute préférable de tracer la courbe par un moyen alternatif.

5 Autres fonctions

5.1 Projections orthogonale et centrale

On a déjà vu qu'il était défini des macros permettant de calculer l'image d'un vecteur dans une projection orthogonale ou une projection centrale. L'équivalent de ces macros existe également pour des courbes. Il s'agit de `td_projc(i,j,o,i',j',o',p)` et de `td_pcentc(i,j,o,i',j',o',c,p)`. `i`, `j` et `o` servent à repérer le plan dans lequel se trouve la courbe `p` avant d'être projeté. `i'`, `j'` et `o'` définissent le plan de projection, et dans le cas d'une projection centrale `c` est une variable de type `td_coords` donnant la position absolue du centre de la projection.

```
td_beginfig(10);
  td_anglevue((10,-100,30),(-2,60,-15),0);
  td_ech:=0.2cm;

  path p,q;
  td_coords i,j,u,A,O;
  i:=(1,0,0); j:=(0,1,0); u:=(1,0,1); A:=(45,5,-20); O:=(0,0,0);

  whatever=td_nouveauplan(i,j,O,(0,-5),(47,15));
  whatever=td_nouveauplan(u,j,O,(0,-5),(25,15));
  td_surfaces;

  p:=fullcircle scaled 5 shifted (30,5); td_courbe(i,j,O,p);
  q:=td_pcentc(i,j,O,u,j,O,A,p); td_courbe(u,j,O,q);

  for z=0 upto 40:
    td_ligneabs(A,td_ddtotd(u,j,O,point z/40*length(q) of q));
  endfor
td_endfig;
```

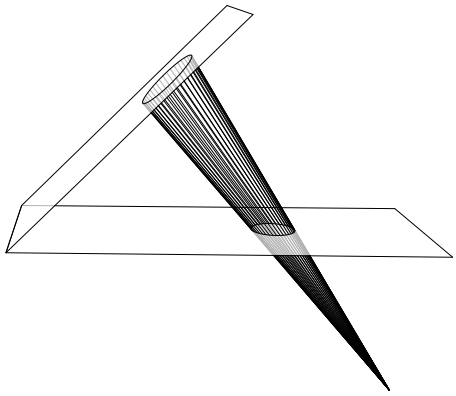


Figure 10

5.2 Insertion d'une image

Il est possible d'insérer une variable de type `picture` n'importe où sur la dessin. La commande pour faire cela est `td_insere(p,o)` où `o` est la position absolue du point qui va correspondre au centre de l'image `p` à insérer. L'image insérée ne subira aucune modification, elle sera juste posée à l'endroit demandé.

Une application de cela est sûrement la nomination des points ou des droites ou des plans d'une figure. En effet, placer la lettre "O" à l'origine se fait simplement par la commande suivante

```
td_insere(thelabel.bot("O",(0,0)),(0,0,0))
```

qui peut être raccourcie en

```
td_label.bot("O",(0,0,0));
```

Les macros `td_labels`, `td_dotlabel` et `td_dotlabels` sont également définies. Par exemple la figure 11 s'obtient simplement en rajoutant la ligne

```
td_dotlabels.rt(A);
```

à la fin du code précédent.

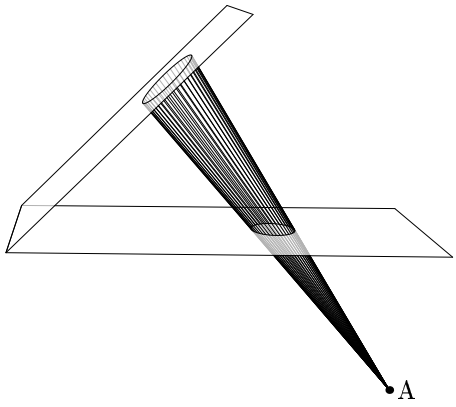


Figure 11

Ainsi ce sont bien les variables globales `laboff`, `labxf`, `labyf` et `dotlabeldiam` ou encore `defaultfont` et `defaultscale` qui sont utilisées et par conséquent ce sont bien elles qu'il faut mettre à jour. D'ailleurs ce serait peut-être bon de le faire ici.

6 Bogues

Il y en a sûrement plein... à chaque fois que je compile quelque chose j'en trouve un nouveau. N'hésitez surtout pas à me les signaler à <`caruso@clipper.ens.fr`>. Il y a aussi quelques manques : tout d'abord, je ne suis pas satisfait de la façon dont est gérée l'annotation. Ensuite, il serait bien de pouvoir travailler avec des surfaces pas forcément planes, et faire des opérations sur icelles comme l'intersection. Finalement (enfin non sûrement pas), les macros servant à tracer les courbes pourraient ne pas se contenter de prélever quelques points sur celles-ci mais se préoccuper en outre des tangentes en ces points, permettant ainsi d'avoir de plus belles courbes avec moins de calcul. Mais bon, complète qui voudra.

Sinon, également quelques trucs grotesques comme les noms de macros et des variables qui sont en français à l'exception de `td_beginfig`, `td_endfig`, `td_label` et ses dérivés, mais je n'ai pas envie de corriger ça maintenant.

Annexes

Liste des variables globales

Nom de la variable	Type
td_aretes	picture
td_aretescachees	picture
td_bas	td_coords
td_direction	td_coords
td_ech	numeric
td_i	td_coords
td_i_enc	td_coords
td_j	td_coords
td_j_enc	td_coords
td_masq	boolean
td_o_enc	td_coords
td_oeil	td_coords
td_options	text
td_pointilles	text
td_precision	td_coords
td_zero	numeric
tdfleche_long	numeric
tdfleche_nb	numeric
tdfleche_norm	numeric
tdsurf_dessin[]	boolean
tdsurf_i[]	td_coords
tdsurf_indice	numeric
tdsurf_j[]	td_coords
tdsurf_nb[]	numeric
tdsurf_o[]	td_coords
tdsurf_precision	numeric
tdsurf_sommets[][]	td_coords

Liste des macros

Nom de la macro	Type des arguments	Valeur de retour
td_anglevue	(td_coords,td_coords,numeric)	
td_beginfig	(numeric)	
td_cachees	(text)	
td_courbe	(td_coords,td_coords,td_coords,path)	
td_courbe_	(path)	
td_courbefleche	(td_coords,td_coords,td_coords,path)	
td_courbeflechei	(td_coords,td_coords,td_coords,path)	
td_courbefleches	(td_coords,td_coords,td_coords,path)	
td_ddtotd	(td_coords,td_coords,td_coords,pair)	td_coords
td_det	(pair,pair)	numeric
td_dotlabel	(string,td_coords)	
td_dotlabels	(string,...)	
td_endfig		
td_fleche	(td_coords,td_coords)	
td_image		picture

Nom de la macro	Type des arguments	Valeur de retour
td_init		
td_insere	(picture,td_coords)	
td_label	(string,td_coords)	
td_labels	(string,...)	
td_ligne	(td_coords,td_coords,td_coords,pair,...)	
td_ligne_	(pair,...)	
td_ligneabs	(td_coords,...)	
td_ligneabs__	(td_coords,td_coords)	
td_lignefleche	(td_coords,td_coords,td_coords,pair,...)	
td_lignefleche_	(pair,...)	
td_ligneflechei	(td_coords,td_coords,td_coords,pair...)	
td_ligneflechei_	(pair,...)	
td_ligneflecheiabs	(td_coords,...)	
td_lignefleches	(td_coords,td_coords,td_coords,pair,...)	
td_lignefleches_	(pair,...)	
td_ligneflechesabs	(td_coords,...)	
td_masquees		
td_memesigne	(numeric,numeric)	boolean
td_nonmasquees		
td_norme	(td_coords)	numeric
td_nouveauplan	(td_coords,td_coords,td_coords, td_coords,td_coords)	numeric
td_nouveauplan_	(td_coords,td_coords)	numeric
td_nouvellesurf	(td_coords,td_coords,td_coords,text)	numeric
td_nouvellesurf_	(td_coords,...)	numeric
td_nouvellesurfabs	(text)	numeric
td_nouvellesurfc	(td_coords,td_coords,td_coords,path)	numeric
td_nouvellesurfc_	(path)	numeric
td_option	(text)	
td_pcentc	(td_coords,td_coords,td_coords,td_coords, td_coords,td_coords,td_coords,path)	path
td_pcentdir	(td_coords,td_coords,td_coords, td_coords,td_coords)	pair
td_pdtscal	(td_coords,td_coords)	numeric
td_pdtvect	(td_coords,td_coords)	td_coords
td_plandroite	(td_coords,td_coords,td_coords, td_coords,td_coords)	numeric
td_projc	(td_coords,td_coords,td_coords, td_coords,td_coords,td_coords,path)	path
td_projdir	(td_coords,td_coords,td_coords)	pair
td_projnorm	(td_coords,td_coords)	td_coords
td_surface	(numeric)	
td_surfaces		
td_trace	(path)	
td_tracecache	(path)	
td_unite	(td_coords)	td_coords
td_visible	(td_coords,numeric)	boolean
td_x	(td_coords)	numeric
td_y	(td_coords)	numeric
td_z	(td_coords)	numeric

Code source du dessin de la page de garde

```
td_beginfig(0);
  td_ech:=0.7cm;
  td_anglevue((0,-50,20),(0,25,-5),0);

  td_coords i,j,k,O,P,A,B;
  i:=(1,0,0); j:=(0,1,0); k:=(0,0,1); P:=(0,0,10); O:=(0,0,0);

  whatever=td_nouveauplan(i,j,P,(-5,-8),(30,8));
  td_surfaces;

  A:=td_ddtotd(i,j,P,(-2,-4));
  B:=td_ddtotd(i,j,P,(25,6));
  td_ligneabs(A,B);

  for t=1 upto 9: td_ligneabs(O,A+(t/10)*(B-A)); endfor

  tdfleche_long:=1;
  tdfleche_norm:=0.4;
  td_courbefleche (td_unite(B-A),td_unite(td_projnorm(O-A,B-A)),O,
               subpath (7.52,7.95) of fullcircle scaled 25);

  td_dotlabels.bot(O);

  td_option(withpen pencircle scaled 1);
  td_ligneabs(-0.2*(B-A),(1.8)*(B-A));
td_endfig;
```