11

Dessins en PostScript et géométrie analytique

1 Utiliser les coordonnées pour dessiner

De quoi s'agit-il ?

Réaliser quelques dessins en PostScript.

Enjeux

Mettre en œuvre de manière plaisante les premiers rudiments de géométrie analytique en découvrant le PostScript.

La géométrie analytique prend place dans la construction de l'idée de linéarité d'une part par la correspondance linéaire, sur chaque axe, entre distances (munies d'un signe) à l'origine et abscisses, et d'autre part par la représentation des droites et plans à l'aide d'équations linéaires et affines, voir sections 5 et 6 du chapitre 16.

De quoi a-t-on besoin ?

Un éditeur de texte et un interpréteur PostScript[1].

Prérequis. – Repérages : pour repérer un point sur une feuille de papier, il faut par exemple deux directions (deux axes), une origine, une orientation et une unité de mesure sur chacune des deux directions, ainsi qu'une convention quant à l'ordre des informations qui sont données.

Comment s'y prendre ?

Le PostScript permet de représenter des points, des lignes droites ou des courbes à partir des coordonnées de points. Il possède deux axes (invisibles). Au départ, le premier est horizontal et coïncide avec le bord inférieur de la feuille de papier. L'autre est vertical et coïncide avec le bord gauche de la feuille. L'origine des deux axes est donc le coin inférieur gauche

[1] On écrit les commandes du PostScript avec n'importe quel éditeur de texte, par exemple `Alpha` sur Macintosh ou `NotePad.exe` (le Bloc-notes) sous Windows.

Les commandes sont ensuite interprétées à l'aide d'un interpréteur PostScript. Sur Macintosh on utilisera `MacGS` ou `GSview` que l'on trouve sur le site internet :

`<http://www.cs.wisc.edu/~ghost/macos/index.htm>`.

Pour Windows, on trouve `GSview` sur le site internet :

`<http://www.cs.wisc.edu/~ghost/gsview/index.html>`.

Ces programmes sont gratuits, sauf l'éditeur `Alpha` qui est un shareware. Il est à noter que les imprimantes PostScript possèdent un interpréteur PostScript incorporé.

de la feuille. Le sens des axes est habituel : sur l'axe horizontal, le sens positif va de gauche à droite et sur l'axe vertical, il va de bas en haut. Ce qui est moins habituel, c'est l'unité de longueur sur chacun des axes : elle vaut exactement $\frac{1}{72}$ pouce. C'est une très petite unité qui permet de dessiner sans devoir utiliser trop de chiffres après la virgule, ou plutôt après le point, car en PostScript la virgule est remplacée par un point. Dans cette unité, la largeur d'une feuille A4 vaut 612 et la hauteur 792. Autrement dit, le coin supérieur droit de la feuille possède par défaut (612,792) comme coordonnées[2].

Les premières commandes. – Pour le PostScript, un dessin est un ensemble de lignes droites et de courbes qui sont soit tracées, soit remplies (avec une certaine couleur). L'ensemble des lignes droites et courbes est appelé *chemin*, en anglais *path*. La première chose à faire pour commencer un dessin ou une partie de dessin est de dire que l'on commence un nouveau chemin en écrivant : `newpath`.

L'idée de départ est assez simple : un morceau de ligne droite (un segment) va d'un point à un autre. Sur cette base, une ligne brisée, en un ou plusieurs morceaux, se définit grâce aux instructions suivantes :

– `moveto` : on (dé)place le *point courant* à l'endroit spécifié ; cela correspond au fait d'aller placer son crayon en un point de la feuille (sans tracer quoi que ce soit).

– `lineto` : on déplace le point courant à l'endroit spécifié tout en définissant une partie de chemin qui devra être tracé ; c'est un segment qui va du point où l'on se trouvait au point indiqué.

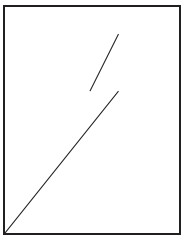
– `rmoveto` : c'est la même chose que `moveto`, mais ce sont les coordonnées du déplacement qui sont indiquées et non celles du point d'arrivée ; c'est un déplacement relatif.

– `rlineto` : c'est la même chose que `lineto`, mais ce sont les coordonnées du déplacement qui sont indiquées et non celles du point d'arrivée.

Lorsque le chemin a été défini au moyen de ces commandes, il n'apparaît pas encore sur le dessin. Il faut encore le tracer, ce qui se fait au moyen de la commande `stroke`.

Voici un premier dessin.

```
newpath
0 0 moveto
400 500 lineto
0 200 rmoveto
-100 -200 rlineto
stroke
```



On peut observer que

– les coordonnées des points sont indiquées *avant* l'instruction qui leur correspond ; par exemple

[2] Il y a une différence entre les dimensions réelles de la feuille et la partie qui peut en être imprimée. Il y a en effet une bordure inaccessible à l'imprimante, dont les dimensions varient d'une imprimante à l'autre.

```
0 0 moveto
```

signifie : déplacer le point courant en $(0, 0)$;

– on ne peut tracer de segment ou se déplacer de manière relative que si l'on se trouve déjà quelque part, c'est-à-dire s'il existe un point courant ; après un `newpath`, il n'y a pas de point courant et il faut donc commencer le chemin par un `moveto` avant d'utiliser `lineto`, `rlineto` ou `rmoveto`.

Modifier le système d'axes. – On peut modifier le système d'axes de plusieurs manières différentes. En voici deux qui sont utiles pour les dessins de cette activité.

1. Pour déplacer les axes, sans les changer de direction, de sens, ni d'unité, on utilise l'instruction `translate` qui en change uniquement l'origine. Par exemple `100 100 translate` déplace l'origine au point $(100, 100)$.
2. Pour faire tourner le système d'axes, on utilise l'instruction `rotate`. Par exemple, `45 rotate` fait tourner le système d'axes de 45 degrés, autour de l'origine, dans le sens opposé à celui des aiguilles d'une montre.

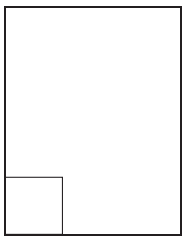
Les commandes `translate` et `rotate` sont toujours relatives au système d'axes en vigueur au moment où on les écrit.

Un carré

Dessiner un carré dont le côté mesure 200 unités et dont le coin inférieur gauche coïncide avec le coin inférieur gauche de la feuille.

On travaille avec des coordonnées relatives pour devoir modifier les instructions le moins possible si on en change le point de départ.

```
newpath
0 0 moveto
200 0 rlineto
0 200 rlineto
-200 0 rlineto
0 -200 rlineto
stroke
```



Le carré se trouve en bas et à gauche de la feuille. On n'en voit bien que deux côtés, ce qui peut être gênant.

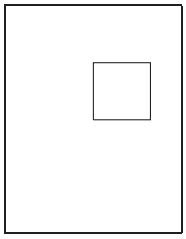
Dessiner le même carré mais avec son coin inférieur gauche au centre de la feuille.

Voici deux possibilités pour déplacer le carré vers le centre de la feuille :

1. Placer le coin inférieur gauche au centre.
2. Déplacer les axes.

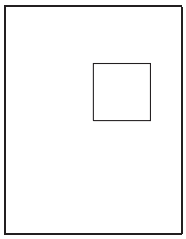
Placer le coin inférieur gauche du carré au centre de la feuille, c'est-à-dire en $(306, 396)$, se fait en remplaçant `0 0 moveto` au début par `306 396 moveto`.

```
newpath
306 396 moveto
200 0 rlineto
0 200 rlineto
-200 0 rlineto
0 -200 rlineto
stroke
```



En déplaçant l'origine du système d'axes au moyen de `translate`, le coin inférieur gauche du carré reste $(0, 0)$; ce dernier point n'est toutefois plus le coin inférieur gauche de la feuille mais le point dont les coordonnées ont été transmises à l'opérateur `translate`.

```
306 396 translate
newpath
0 0 moveto
200 0 rlineto
0 200 rlineto
-200 0 rlineto
0 -200 rlineto
stroke
```



Faire tourner le carré

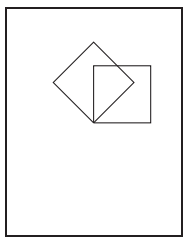
Dessiner le même carré « sur une pointe ».

Il s'agit par exemple de faire tourner le carré de 45° (sens trigonométrique) autour du sommet inférieur gauche. Une manière de le faire est de calculer les coordonnées de chacun des sommets du carré dans cette position. Une autre manière, beaucoup plus facile, consiste à faire tourner les axes de 45° au moyen de l'instruction `rotate` et à dessiner le carré dans ce nouveau système d'axes : si on garde les mêmes coordonnées, le carré suit le mouvement.

On voit ici l'intérêt d'avoir déplacé l'origine du système d'axes au milieu de la feuille : faire tourner le système d'axes se fait autour de son origine. Si l'on gardait l'origine située au coin inférieur gauche de la feuille et que l'on faisait tourner le système d'axes, le carré, en suivant ce mouvement, sortirait de la feuille.

```
306 396 translate
newpath
0 0 moveto
200 0 rlineto
0 200 rlineto
-200 0 rlineto
0 -200 rlineto
stroke
```

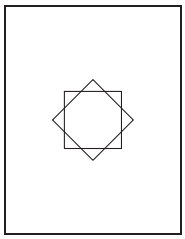
```
45 rotate
newpath
0 0 moveto
200 0 rlineto
0 200 rlineto
-200 0 rlineto
0 -200 rlineto
stroke
```



Dessiner les mêmes carrés de telle manière que le deuxième soit obtenu par une rotation de 45° autour du centre du premier et non pas autour de son sommet inférieur gauche.

Il suffit que le centre du carré coïncide avec l'origine du système d'axes. Pour cela on place son sommet inférieur gauche en $(-100, -100)$.

```
306 396 translate
newpath
-100 -100 moveto
200 0 rlineto
0 200 rlineto
-200 0 rlineto
0 -200 rlineto
stroke
45 rotate
newpath
-100 -100 moveto
200 0 rlineto
0 200 rlineto
-200 0 rlineto
0 -200 rlineto
stroke
```

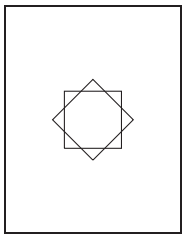


Les instructions pour dessiner les deux carrés sont identiques ! On peut « mémoriser » ces instructions pour ne devoir les écrire qu'une seule fois. Ceci sera utile lorsque l'on voudra dessiner plus de deux fois un tel carré... Pour mémoriser une suite d'instructions, il faut choisir un nom, par exemple `carre`. Ce nom ne peut pas comporter de caractère accentué. Pour la définition on fait précéder ce nom du caractère `/`. En écrivant

```
/carre { ... } def
```

on mémorise la suite d'instructions entre les accolades dans `carre`. Lorsque l'on écrira ensuite `carre`, cela aura exactement le même effet que d'écrire directement cette suite d'instructions.

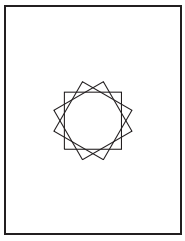
```
/carre {
  newpath
  -100 -100 moveto
  200 0 rlineto
  0 200 rlineto
  -200 0 rlineto
  0 -200 rlineto
  stroke
} def
306 396 translate
carre
45 rotate
carre
```



Dessiner trois ou quatre carrés identiques ayant même centre et placés régulièrement autour de ce centre.

La solution s'inspire directement de ce qui précède. La voici par exemple pour trois carrés :

```
/carre {
  newpath
  -100 -100 moveto
  200 0 rlineto
  0 200 rlineto
  -200 0 rlineto
  0 -200 rlineto
  stroke
} def
306 396 translate
carre
30 rotate
carre
30 rotate
carre
```



2 Parallélisme

De quoi s'agit-il ?

Traiter, en PostScript, de questions concernant les parallélogrammes et les cubes.

Enjeux

Utiliser des vecteurs pour dessiner et prouver.

Sur la place des vecteurs dans le développement de l'idée de linéarité, voir la section 7 du chapitre 16.

De quoi a-t-on besoin ?

Les outils informatiques pour travailler en PostScript (voir page 351).

Prérequis. – Une première initiation au PostScript, par exemple la section 1 de ce chapitre. Une description plus complète de quelques principes et des opérateurs de base se trouvent dans l'annexe 4 à la page 501. Le lecteur est invité à s'y référer lorsque les aspects du PostScript utilisés ici nécessitent un peu plus d'explications.

Comment s'y prendre ?

En PostScript, on met toujours les arguments avant le nom de la fonction. Pour la suite des activités, il est utile de préciser cela : le PostScript, comme d'autres langages informatiques, travaille avec ce qu'on appelle une *pile*. Bornons-nous à dire ici qu'une pile est une mémoire dans laquelle s'entassent les uns au-dessus des autres (ou les uns à côté des autres) les éléments que l'on y met. Chaque opérateur y prend, en commençant par le dessus, les arguments dont il a besoin. Pour plus de détails, le lecteur est invité à consulter la section 1.1 de l'annexe 4 à la page 501.

Pour faciliter le travail d'écriture des dessins, nous proposons d'utiliser des opérateurs qui ne sont pas standard en PostScript, mais qui permettent de travailler directement avec des vecteurs à deux ou à trois dimensions. Ces opérateurs sont décrits dans l'annexe 5 à la page 509. Pour les avoir à sa disposition, il suffit de les copier avant les instructions PostScript du dessin que l'on souhaite réaliser[3].

Les vecteurs s'introduisent entre crochets, avec un (ou plusieurs) espace(s) pour séparer les composantes. Lorsque nous disons « vecteurs », il s'agit de n-uples de réels qui représentent, selon le contexte et l'opérateur PostScript utilisé, tantôt une position, tantôt un déplacement.

Les nouveaux opérateurs définis ont leur nom qui commence par une majuscule. Par exemple `Add` permet d'additionner deux vecteurs. Comme toujours en PostScript, les vecteurs que l'on additionne se placent avant l'opérateur.

Les opérateurs disponibles sont :

`Add` et `Sub` : addition et soustraction de deux vecteurs.

`Mul` : multiplication d'un vecteur par un scalaire. On écrit d'abord le vecteur et ensuite le scalaire.

[3] Ils peuvent être téléchargés à partir du site internet du CREM,
`http://www.profor.be/crem/index.htm`

`Div` : division d'un vecteur par un scalaire. On écrit d'abord le vecteur et ensuite le scalaire.

`Neg` : multiplication du vecteur par -1.

`Moveto`, `RMoveto`, `Lineto` et `Rlineto` : définition de chemins à partir de vecteurs à deux ou trois dimensions. Lorsque les vecteurs ont trois dimensions, il y a implicitement une perspective parallèle qui projette les vecteurs sur le plan du dessin. Il s'agit d'une perspective dont la fuyante est à 30° et dont le rapport vaut un demi. On peut modifier les paramètres de la perspective cavalière.

`Point` : dessine un point (un petit disque noir) à l'endroit mentionné. On peut modifier le rayon du disque en introduisant, par exemple

```
/RayonPoint 3 def
```

Par défaut, ce rayon vaut 5.

2.1 Le quadrilatère passant par les milieux

Comment s'y prendre ?

Soit les points

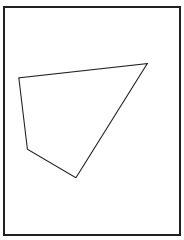
$$A = \begin{pmatrix} 250 \\ 200 \end{pmatrix},\ B = \begin{pmatrix} 500 \\ 600 \end{pmatrix},\ C = \begin{pmatrix} 50 \\ 550 \end{pmatrix} \text{ et } D = \begin{pmatrix} 80 \\ 300 \end{pmatrix}.$$

Dessiner le quadrilatère $ABCD$ et le quadrilatère joignant les milieux des côtés de $ABCD$.
Que peut-on dire de ce dernier quadrilatère ?

Cette activité demande un aller-retour entre l'ordinateur et la feuille de papier : certaines parties du travail demandent en effet de mettre au point, ou de revoir, les outils mathématiques nécessaires à leur réalisation.

Lorsque l'on a recopié les opérateurs permettant de travailler directement avec des vecteurs, on « écrit » le dessin de $ABCD$ en PostScript par exemple de la manière suivante :

```
[...]
/a [250 200] def
/b [500 600] def
/c [50 550] def
/d [80 300] def

newpath
a Moveto b Lineto c Lineto
d Lineto a Lineto stroke
```



Il faut ensuite déterminer le milieu de chaque côté. Prenons le milieu de $[AB]$. Les coordonnées de A correspondent aux composantes du vecteur $\overrightarrow{OA}$. Rappelons qu'en PostScript, l'origine O se trouve au départ dans le coin inférieur gauche de la feuille de dessin. Les coordonnées du point milieu M sont les composantes du vecteurs $\overrightarrow{OM}$ que l'on peut trouver grâce à $\overrightarrow{OA}$ et à $\overrightarrow{AB}$:

$$\overrightarrow{OM} = \overrightarrow{OA} + \frac{1}{2}\overrightarrow{AB}.$$

Si l'on décompose $\overrightarrow{AB}$ en fonction de $\overrightarrow{OA}$ et $\overrightarrow{OB}$, on a :

$$\overrightarrow{OM} = \overrightarrow{OA} + \frac{1}{2}(\overrightarrow{OB} - \overrightarrow{OA}),$$

et l'on trouve finalement :

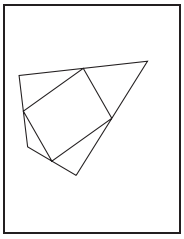
$$\overrightarrow{OM} = \frac{1}{2}(\overrightarrow{OA} + \overrightarrow{OB}).$$

Pour trouver les coordonnées de M, il suffit donc d'additionner celles de A et de B et de diviser le résultat par 2. En PostScript, cela donne `a b Add 2 Div`. Regardons ce calcul en détail[4] :

États successifs de la pile	
`a`	`[250 200]`
`b`	`[250 200]` `[500 600]`
`Add`	`[750 800]`
`2`	`[750 800]` `2`
`Div`	`[375 400]`

On peut donc ajouter maintenant le quadrilatère reliant les milieux des côtés :

```
[...]

                          c d Add 2 Div Lineto
newpath                   d a Add 2 Div Lineto
a b Add 2 Div Moveto      a b Add 2 Div Lineto
b c Add 2 Div Lineto      stroke
```



Le quadrilatère ressemble à s'y méprendre à un parallélogramme. On vérifie que c'en est un grâce aux vecteurs. Appelons M_1 M_2, M_3 et M_4 les quatre milieux (figure 1). Vérifier que $M_1M_2M_3M_4$ est un parallélogramme revient à vérifier, par exemple, que $\overrightarrow{M_1M_2} = \overrightarrow{M_4M_3}$.

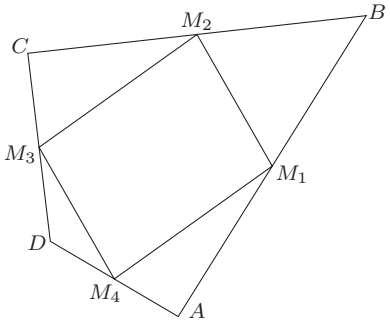


Fig. 1

[4] Dans beaucoup d'ouvrages, la pile est dessinée verticalement. Dans les représentations qui suivent, les piles sont dessinées horizontalement pour gagner de la place. Cela n'a bien sûr aucune importance. Ce qu'il faut garder en tête, c'est que *les éléments de la pile « sortent » du côté où ils « entrent »*.

Calculons les coordonnées de ces quatre milieux :

$$\overrightarrow{OM_1} = \tfrac{1}{2}(\overrightarrow{OA} + \overrightarrow{OB}) = \begin{pmatrix} 375 \\ 400 \end{pmatrix};$$
$$\overrightarrow{OM_2} = \tfrac{1}{2}(\overrightarrow{OB} + \overrightarrow{OC}) = \begin{pmatrix} 275 \\ 575 \end{pmatrix};$$
$$\overrightarrow{OM_3} = \tfrac{1}{2}(\overrightarrow{OC} + \overrightarrow{OD}) = \begin{pmatrix} 65 \\ 425 \end{pmatrix};$$
$$\overrightarrow{OM_4} = \tfrac{1}{2}(\overrightarrow{OD} + \overrightarrow{OA}) = \begin{pmatrix} 165 \\ 250 \end{pmatrix}.$$

La conclusion vient du calcul suivant :

$$\overrightarrow{M_1M_2} = \overrightarrow{OM_2} - \overrightarrow{OM_1} = \begin{pmatrix} 275 \\ 575 \end{pmatrix} - \begin{pmatrix} 375 \\ 400 \end{pmatrix} = \begin{pmatrix} -100 \\ 175 \end{pmatrix};$$
$$\overrightarrow{M_4M_3} = \overrightarrow{OM_3} - \overrightarrow{OM_4} = \begin{pmatrix} 65 \\ 425 \end{pmatrix} - \begin{pmatrix} 165 \\ 250 \end{pmatrix} = \begin{pmatrix} -100 \\ 175 \end{pmatrix}.$$

Obtient-on un parallélogramme quel que soit le quadrilatère $ABCD$ de départ, ou bien les coordonnées ont-elles été choisies de manière à donner ce résultat ?

Les élèves peuvent essayer avec d'autres coordonnées et constater que la propriété se répète. On peut le prouver en se ramenant, comme on l'a fait dans l'exemple précédent, aux coordonnées, mais le calcul algébrique qui en résulte est assez lourd. Faire le calcul directement avec les vecteurs est beaucoup plus simple. On veut donc vérifier que

$$\overrightarrow{M_1M_2} = \overrightarrow{M_4M_3}.$$

Calculons ces deux vecteurs en fonction de A, B, C et D :

$$\overrightarrow{M_1M_2} = \overrightarrow{OM_2} - \overrightarrow{OM_1} = \frac{\overrightarrow{OB}+\overrightarrow{OC}}{2} - \frac{\overrightarrow{OA}+\overrightarrow{OB}}{2} = \frac{\overrightarrow{OC}-\overrightarrow{OA}}{2} = \frac{\overrightarrow{AC}}{2};$$
$$\overrightarrow{M_4M_3} = \overrightarrow{OM_3} - \overrightarrow{OM_4} = \frac{\overrightarrow{OC}+\overrightarrow{OD}}{2} - \frac{\overrightarrow{OD}+\overrightarrow{OA}}{2} = \frac{\overrightarrow{OC}-\overrightarrow{OA}}{2} = \frac{\overrightarrow{AC}}{2}.$$

2.2 Les parallélogrammes par trois points

Comment s'y prendre ?

Soit les points

$$A = \begin{pmatrix} 150 \\ 180 \end{pmatrix},\ B = \begin{pmatrix} 250 \\ 110 \end{pmatrix} \text{ et } C = \begin{pmatrix} 180 \\ 280 \end{pmatrix}.$$

Dessiner tous les parallélogrammes qui ont ces points pour sommets.

Soit un point D tel que $ABCD$ est un parallélogramme. Si les sommets sont dans cet ordre-là (figure 2 à la page suivante), alors on doit avoir

$$\overrightarrow{AB} = \overrightarrow{DC}.$$

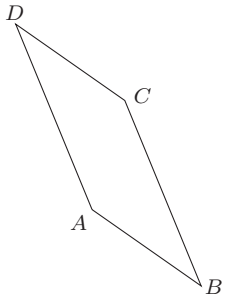


Fig. 2

On peut trouver les coordonnées de D à partir de l'égalité

$$\overrightarrow{OD} \;=\; \overrightarrow{OC} + \overrightarrow{BA}.$$

En traduisant cela en coordonnées, on obtient $D = C + A - B$.

Selon l'ordre dans lequel on effectue les opérations, on obtient plusieurs manières d'écrire cela en PostScript. Par exemple, $D = C + (A - B)$ s'écrit

```
/d c a b Sub Add def
```

tandis que $D = (C + A) - B$ s'écrit

```
/d c a Add b Sub def
```

Regardons ces calculs en détail :

`/d c a b Sub Add def`

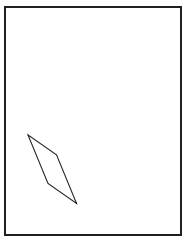
États successifs de la pile				
`/d`	`d`			
`c`	`d`	`[180 280]`		
`a`	`d`	`[180 280]`	`[150 180]`	
`b`	`d`	`[180 280]`	`[150 180]`	`[250 110]`
`Sub`	`d`	`[180 280]`	`[-100 70]`	
`Add`	`d`	`[80 350]`		
`def`				

`/d c a Add b Sub def`

États successifs de la pile				
`/d`	`d`			
`c`	`d`	`[180 280]`		
`a`	`d`	`[180 280]`	`[150 180]`	
`Add`	`d`	`[330 460]`		
`b`	`d`	`[330 460]`	`[250 110]`	
`Sub`	`d`	`[80 350]`		
`def`				

On peut alors écrire la séquence d'instructions qui dessine le parallélogramme $ABCD$.

```
[...]

/a [150 180] def            newpath
/b [250  110] def           a Moveto b Lineto c Lineto
/c [180 280] def            d Lineto a Lineto
/d c a b Sub Add def        stroke
```

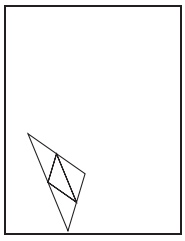


Pour trouver les autres parallélogrammes, il suffit de considérer les autres ordres possibles pour les sommets. Il n'y a que deux autres possibilités.

1. $ACBD$: dans ce cas, $\overrightarrow{OD} = \overrightarrow{OB} + \overrightarrow{CA}$.
2. $ABDC$: dans ce cas, $\overrightarrow{OD} = \overrightarrow{OC} + \overrightarrow{AB}$.

On peut dessiner les trois parallélogrammes sur la même feuille.

```
[...]
/d b a c Sub Add def        /d c b a Sub Add def
newpath                     newpath
a  Moveto c Lineto          a  Moveto b Lineto
b Lineto d Lineto           d Lineto c Lineto
a Lineto                    a  Lineto
stroke                      stroke
```



2.3 Cubes

Soit un cube dont la base inférieure est $ABCD$ et la base supérieure $A'B'C'D'$ (A' se trouvant au dessus de A, ...) avec

$$A = \begin{pmatrix} 100 \\ 200 \\ 0 \end{pmatrix},\ B = \begin{pmatrix} 300 \\ 200 \\ 0 \end{pmatrix},\ D = \begin{pmatrix} 100 \\ 400 \\ 0 \end{pmatrix} \text{ et } A' = \begin{pmatrix} 100 \\ 200 \\ 200 \end{pmatrix}.$$

Dessiner le cube de telle manière que

(a) toutes les arêtes soient visibles (le cube est transparent) ;

(b) le cube soit opaque.

Un schéma peut aider à visualiser la situation (figure 3).

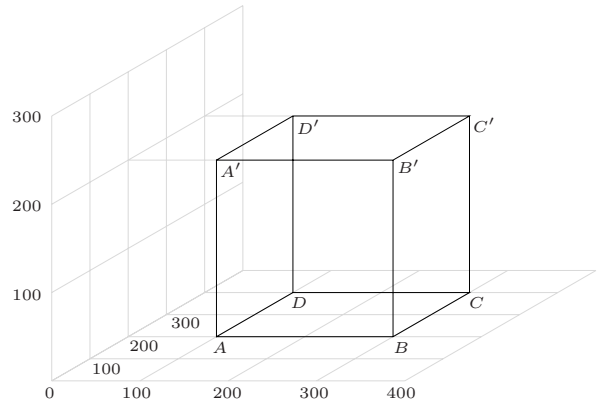


Fig. 3

Pour trouver les coordonnées du point C, on utilise les égalités vectorielles

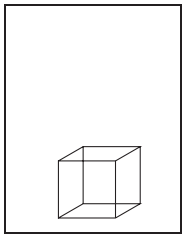
$$\begin{aligned} \overrightarrow{AD} &= \overrightarrow{OD} - \overrightarrow{OA}\,; \\ \overrightarrow{OC} &= \overrightarrow{OB} + \overrightarrow{AD}. \end{aligned}$$

Pour trouver les coordonnées des points B', C' et D', on utilise les égalités vectorielles

$$\begin{aligned} \overrightarrow{AA'} &= \overrightarrow{OA'} - \overrightarrow{OA}\,; \\ \overrightarrow{OB'} &= \overrightarrow{OB} + \overrightarrow{AA'}\,; \\ \overrightarrow{OC'} &= \overrightarrow{OC} + \overrightarrow{AA'}\,; \\ \overrightarrow{OD'} &= \overrightarrow{OD} + \overrightarrow{AA'}. \end{aligned}$$

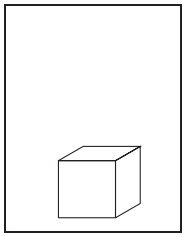
Pour dessiner le cube transparent, on écrit alors le programme PostScript, par exemple, comme ci-après.

```
[...]
/a [100 200 0] def          newpath
/b [300 200 0] def          a Moveto b Lineto c Lineto
/d [100 400 0] def          d Lineto a Lineto
/ap [100 200 200] def       ap Lineto bp Lineto cp Lineto
                            dp Lineto ap Lineto
/c b d a Sub Add def        b Moveto bp Lineto
/bp b ap a Sub Add def      c Moveto cp Lineto
/cp c ap a Sub Add def      d Moveto dp Lineto
/dp d ap a Sub Add def      stroke
```



Pour le cube opaque, il faut uniquement modifier la deuxième partie du programme, ce qui donne

```
[...]
        newpath
        a Moveto b Lineto bp Lineto
        ap Lineto a Lineto
        b Moveto c Lineto cp Lineto
        bp Lineto b Lineto
        ap Moveto bp Lineto cp Lineto
        dp Lineto ap Lineto
        stroke
```



Prolongement possible

Dessiner un octaèdre régulier.

Indication : chaque sommet d'un octaèdre régulier est le centre d'une face d'un cube.

2.4 Sections de cubes

Comment s'y prendre ?

Dessiner en PostScript la section du cube $ABCDA'B'C'D'$ ci-dessus par le plan PQR où

- P se trouve au tiers de l'arête $[AB]$, du côté de A ;
- Q est au milieu de l'arête $[BC]$;
- R est au milieu de l'arête $[CC']$.

Pour ce faire, déterminer des procédures générales pour trouver l'intersection d'une droite avec les différentes faces du cube.

Dans un premier temps, dessinons les points P, Q et R. Les milieux de segments n'ont plus de secret pour nous. Pour le tiers, on a

$$\overrightarrow{OP} = \overrightarrow{OA} + \tfrac{1}{3}\overrightarrow{AB}.$$

Ceci peut s'écrire immédiatement en PostScript

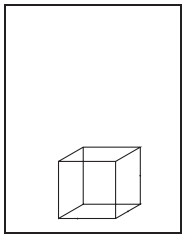
```
/p a b a Sub 3 Div Add def
```

En détail,

États successifs de la pile	
/p	p
a	p | [100 200 0]
b	p | [100 200 0] | [300 200 0]
a	p | [100 200 0] | [300 200 0] | [100 200 0]
Sub	p | [100 200 0] | [200 0 0]
3	p | [100 200 0] | [200 0 0] | 3
Div	p | [100 200 0] | [$\frac{200}{3}$ 0 0]
Add	p | [$\frac{500}{3}$ 200 0]
def	

Pour placer un point (un petit disque noir) aux endroits requis, on utilise l'opérateur (non standard) `Point`. (On peut modifier la grosseur du point en redéfinissant le paramètre `RayonPoint`.)

```
[...]
/p a b a Sub 3 Div Add def
/q b c Add 2 Div def
/r c cp Add 2 Div def

p Point q Point r Point
[...]
```



Pour déterminer les autres sommets de la section, il est nécessaire de trouver des points intermédiaires. Ce sont les intersections de droites contenues dans le plan de section avec une face du cube (ou plus précisément un plan contenant une face). Par exemple, on peut rechercher le point S, intersection de la droite PQ avec le plan contenant la face $DCC'D'$ (figure 4).

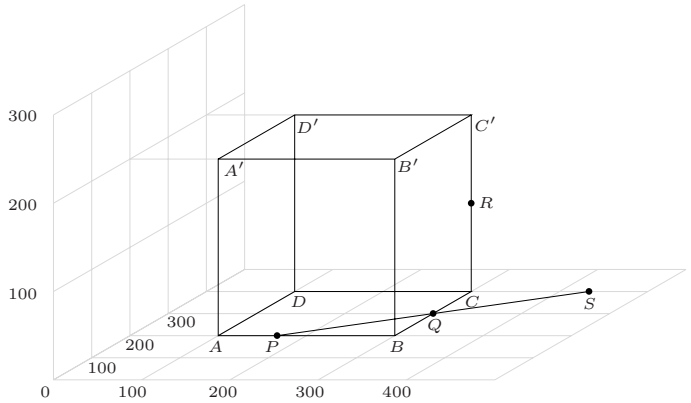


Fig. 4

Le point S se trouve sur la droite PQ. On peut donc écrire

$$\overrightarrow{OS} \quad = \quad \overrightarrow{OP} + \lambda_S \overrightarrow{PQ},$$

où λ_S est un scalaire. Comme le point S se trouve dans la face verticale à l'arrière du cube, on sait que sa deuxième coordonnée y_S vaut 400. Par conséquent,

$$400 \quad = \quad y_P + \lambda_S(y_Q - y_P),$$

ou encore

$$\lambda_S \quad = \quad \frac{400-y_P}{y_Q-y_P} \quad = \quad \frac{400-200}{300-200} \quad = \quad 2.$$

Remarquons qu'une simple considération sur les triangles isométriques QBP et QCS amène également ce résultat.

En PostScript, les coordonnées du point S se calculent alors de cette manière qui suit

```
/s p q p Sub 2 Mul Add def
```

Pour trouver les coordonnées du point T, intersection de la droite SR avec la face horizontale supérieure du cube (figure 5), on recommence le même genre de calcul. On sait que

$$\overrightarrow{OT} \quad = \quad \overrightarrow{OS} + \lambda_T \overrightarrow{SR},$$

et que la troisième coordonnée z_T de T vaut 200, ce qui entraîne que

$$200 \quad = \quad z_S + \lambda_T(z_R - z_S),$$

ou encore

$$\lambda_T \quad = \quad \frac{200-z_S}{z_R-z_S} \quad = \quad \frac{200-0}{100-0} \quad = \quad 2.$$

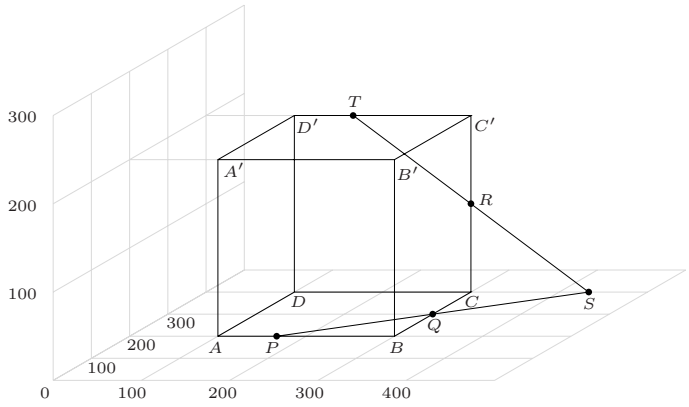


Fig. 5

En PostScript, les coordonnées du point T se calculent alors comme suit

```
/t s r s Sub 2 Mul Add def.
```

Il faut rechercher les coordonnées du point U, intersection de la droite passant par T et parallèle à PS (elle se trouve dans la face supérieure du cube) avec la face gauche du cube (figure 6). On sait qu'il existe un scalaire λ_U tel que

$$\overrightarrow{OU} \quad = \quad \overrightarrow{OT} + \lambda_U \overrightarrow{PQ}.$$

Comme U de trouve dans la face verticale à gauche du cube, on sait aussi que $x_U = 100$. Il en découle que

$$100 \quad = \quad x_T + \lambda_U(x_Q - x_P),$$

ou encore

$$\lambda_U \quad = \quad \frac{100 - x_T}{x_Q - x_P}.$$

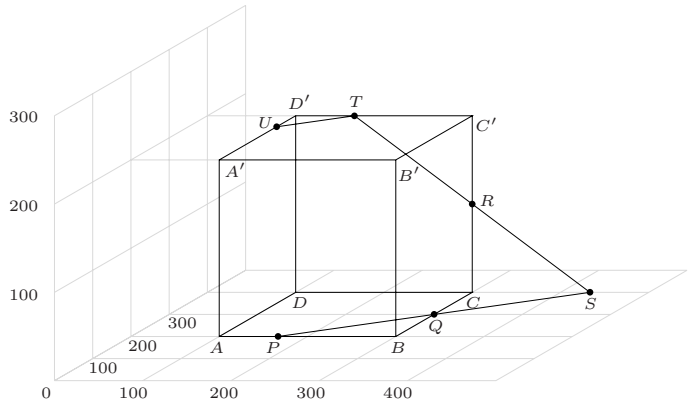


Fig. 6

Les coordonnées de T (et donc x_T) ont été calculées en PostScript. Nous pouvons récupérer x_T au moyen de l'opérateur `get`. Pour avoir la première coordonnée de `t`, il faut écrire

```
t 0 get
```

car, en PostScript, les indices sont numérotés à partir de zéro. De même, nous pouvons utiliser `get` pour obtenir $x_Q - x_P$. Nous pouvons donc poursuivre le programme par

```
/lu 100 t 0 get sub q p Sub 0 get div def
/u t q p Sub lu Mul Add def
```

Regardons le premier de ces calculs en détail[5],

	États successifs de la pile
`/lu`	`lu`
`100`	`lu` | `100`
`t`	`lu` | `100` | $[x_T \ y_T \ z_T]$
`0 get`	`lu` | `100` | x_T
`sub`	`lu` | `100` $-$ x_T
`q p Sub`	`lu` | `100` $-$ x_T | $[x_Q - x_P \ y_Q - y_P \ z_Q - z_P]$
`0 get`	`lu` | `100` $-$ x_T | $x_Q - x_P$
`div`	`lu` | $\frac{100 - x_T}{x_Q - x_P}$
`def`	

[5] Dans la description de la pile, les noms de variables en *italique* indiquent une valeur numérique dans la pile tandis que les noms en `caractères droits` indiquent des noms PostScript.

Il nous reste à trouver l'intersection W de l'arête AA' avec le plan de section. C'est le point d'intersection de la droite passant par U et parallèle à QR avec la face avant du cube (figure 7). Nous avons donc

$$\overrightarrow{OW} = \overrightarrow{OU} + \lambda_W \overrightarrow{QR},$$

et

$$200 = y_U + \lambda_W (y_R - y_Q).$$

Par conséquent,

$$\lambda_W = \frac{200 - y_U}{100}.$$

Nous pouvons donc écrire en PostScript

```
/lw 200 u 1 get sub 100 div def
/w u r q Sub lw Mul Add def
```

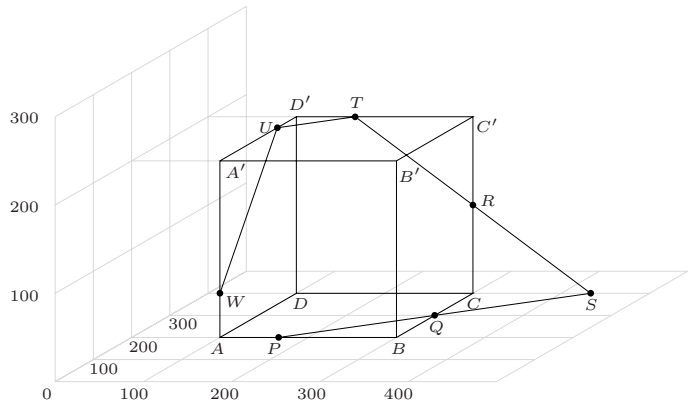


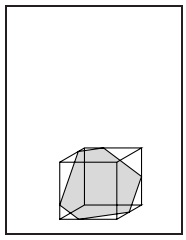
Fig. 7

Nous avons tous les points permettant de dessiner la section. Il reste maintenant à les relier, et éventuellement à dessiner la section en grisé. Ceci se fait au moyen de l'opérateur `setgray` qui prend un argument dans la pile : le niveau de gris. La couleur blanche se définit par `1 setgray` et le noir par `0 setgray`. Les niveaux de gris intermédiaires[6] se définissent en donnant une valeur entre `0` et `1`. Si on utilise ensuite l'opérateur `stroke`, les traits seront dessinés dans le gris choisi. Pour dessiner une surface en gris, il faut utiliser l'opérateur `fill` qui remplit l'intérieur du chemin courant dans la couleur choisie. Comme l'opérateur `fill` rend le dessin opaque, l'ordre dans lequel on place les instructions de dessin a son importance.

[6] Le niveau de gris est en fait le rapport entre le nombre de pixels blancs et le nombre total de pixels dans une surface donnée.

```
0.85 setgray
newpath
p Moveto q Lineto
r Lineto t Lineto
u Lineto w Lineto
p Lineto fill
```

```
0 setgray
newpath
p Moveto q Lineto
r Lineto  t Lineto
u Lineto w Lineto
p Lineto stroke

[ dessin du cube ]
```



Prolongement possible

Les procédures mises en œuvre pour trouver les coordonnées de S, T, U et W sont semblables. La seule vraie différence se trouve dans la composante utilisée pour déterminer la valeur du paramètre (λ_S, λ_T, λ_U ou λ_W). Il s'agit en effet dans tous les cas de trouver l'intersection d'une droite avec un plan dont la principale caractéristique (du point de vue algébrique) est précisément que tous ses points ont une de leurs coordonnées constante.

> Écrire une procédure automatique permettant de déterminer la valeur de λ en fonction des données suivantes :
> – deux points de la droite ;
> – l'indice de la composante constante des points du plan ;
> – la valeur de cette constante.

Un telle procédure demande donc quatre arguments (à prendre dans la pile) :

1. Le premier point de la droite.
2. Le deuxième point de la droite.
3. L'indice de la composante constante des points du plan (1 pour x, 2 pour y et 3 pour z).
4. La valeur de cette constante.

Pour trouver la valeur de λ_S avec une telle procédure, appelée par exemple `lambda` et qui est détaillée ci-après, on aura à introduire

```
p q 2 400 lambda
```

et pour trouver λ_T

```
s r 3 200 lambda
```

Construisons la procédure en suivant pas à pas ce qu'elle doit faire dans le premier cas. Au moment où l'on écrit `lambda`, il y a donc dans la pile

$[x_P\ y_P\ z_P]$	$[x_Q\ y_Q\ z_Q]$	2	400

Pour pouvoir utiliser facilement les données qui se trouvent dans la pile, nous les mettons dans des variables ayant un nouveau nom (leur nom d'origine ne sert à rien car d'une fois à l'autre il sera différent) :

```
/@valeur exch def
```

mettra `400` dans la variable `@valeur`.

Une remarque technique s'impose ici. Pourquoi utiliser le caractère « `@` » ? Il peut être considéré comme une lettre au même titre que les autres caractères de l'alphabet. Il convient relativement bien pour « protéger » les noms de variables. Supposons que nous utilisons le nom `valeur`. Il y a un risque qu'un utilisateur de la macro fasse appel à la macro `lambda` sans savoir (ou en ayant oublié) qu'une variable avec ce nom y est définie. Il n'est donc pas impossible que l'utilisateur définisse une variable avec ce même nom. Dans un tel cas, la valeur de la variable `valeur` sera « écrasée » par celle que lui attribue la macro `lambda`. Celui qui programme la macro doit donc « protéger » les noms. Ceci est une problématique générale en informatique. Elle donne lieu aux notions de variables *locales* et *globales*. Une manière de réaliser une bonne protection à peu de frais est de réserver un caractère comme « `@` » pour ne l'utiliser que dans des noms intermédiaires[7].

Regardons la définition de `@valeur` en détail,

États successifs de la pile					
	$[x_P\ y_P\ z_P]$	$[x_Q\ y_Q\ z_Q]$	2	400	
`/@valeur`	$[x_P\ y_P\ z_P]$	$[x_Q\ y_Q\ z_Q]$	2	400	`@valeur`
`exch`	$[x_P\ y_P\ z_P]$	$[x_Q\ y_Q\ z_Q]$	2	`@valeur`	400
`def`	$[x_P\ y_P\ z_P]$	$[x_Q\ y_Q\ z_Q]$	2		

Nous récupérons ensuite l'indice de la composante utile. Nous avons vu qu'en PostScript les indices des composantes sont numérotés à partir de 0. Nous devons donc retirer 1 pour que l'utilisateur puisse introduire la valeur usuelle (numérotée à partir de 1)

```
1 sub /@indice exch def
```

mettra `1` dans la variable `@indice`. En détail :

États successifs de la pile					
	$[x_P\ y_P\ z_P]$	$[x_Q\ y_Q\ z_Q]$	2		
`1`	$[x_P\ y_P\ z_P]$	$[x_Q\ y_Q\ z_Q]$	2	1	
`sub`	$[x_P\ y_P\ z_P]$	$[x_Q\ y_Q\ z_Q]$	1		
`/@indice`	$[x_P\ y_P\ z_P]$	$[x_Q\ y_Q\ z_Q]$	1	`@indice`	
`exch`	$[x_P\ y_P\ z_P]$	$[x_Q\ y_Q\ z_Q]$	`@indice`	1	
`def`	$[x_P\ y_P\ z_P]$	$[x_Q\ y_Q\ z_Q]$			

Les instructions

```
@indice get /@q exch def
```

mettront y_Q dans la variable `@q`. En détail,

[7] Même si le PostScript ne possède pas de variables locales liées à des opérateurs, il existe une notion de noms locaux : ceux de variables ou opérateurs associés à un dictionnaire. Ceci devient trop technique pour le travail suggéré ici, mais peut faire l'objet d'un travail au cours d'informatique.

États successifs de la pile	
	$[x_P\ y_P\ z_P]$ $[x_Q\ y_Q\ z_Q]$
`@indice`	$[x_P\ y_P\ z_P]$ $[x_Q\ y_Q\ z_Q]$ `1`
`get`	$[x_P\ y_P\ z_P]$ y_Q
`/@q`	$[x_P\ y_P\ z_P]$ y_Q `@q`
`exch`	$[x_P\ y_P\ z_P]$ `@q` y_Q
`def`	$[x_P\ y_P\ z_P]$

Les instructions

```
@indice get /@p exch def
```

mettront y_P dans la variable `@p`.

Il reste alors à effectuer le calcul

$$\frac{400 - y_P}{y_Q - y_P},$$

c'est-à-dire

```
@valeur @p sub @q @p sub div
```

Reprenant les instructions les unes à la suite des autres, nous avons

```
/@valeur exch def
1 sub /@indice exch def
@indice get /@q exch def
@indice get /@p exch def
@valeur @p sub @q @p sub div
```

Définir l'opérateur `lambda` revient à compléter ces instructions de la manière suivante.

```
/lambda{
/@valeur exch def
1 sub /@indice exch def
@indice get /@q exch def
@indice get /@p exch def
@valeur @p sub @q @p sub div
} def
```

On peut alors définir les point S et T comme ci-dessous.

```
[...]
/lambda{
  /@valeur exch def
  1 sub /@indice exch def
  @indice get /@q exch def
  @indice get /@p exch def
  @valeur @p sub @q @p sub div
} def
```

```
/ls p q 2 400 lambda def
/s p q p Sub ls Mul Add def
/lt s r 3 200 lambda def
/t s r s Sub lt Mul Add def
```

Pour déterminer U, il faut trouver un deuxième point de la droite qui passe par T et qui est parallèle à une direction donnée par un vecteur. Ceci est

très facile (figure 8) : on prend le point U' (`up`) défini par

$$\overrightarrow{OU'} = \overrightarrow{OT} + \overrightarrow{PQ}.$$

Lorsque U sera défini, pour déterminer W, on trouvera W' (`wp`) par

$$\overrightarrow{OW'} = \overrightarrow{OU} + \overrightarrow{QR}.$$

```
/up t q p Sub Add def
/lu t up 1 100 lambda def
/u t up t Sub lu Mul Add def
```

```
/wp u r q Sub Add def
/lw u wp 2 200 lambda def
/w u wp u Sub lw Mul Add def
```

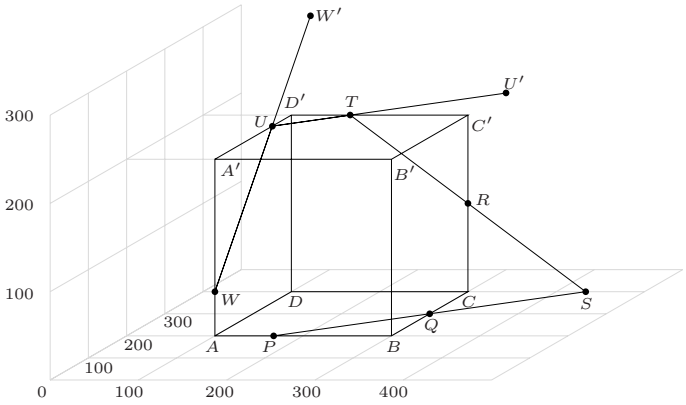


Fig. 8

3 Vu et caché

De quoi s'agit-il ?

Déterminer les parties vues et cachées d'une droite traversant un tétraèdre opaque.

Enjeux

Approfondir la question du vu et du caché.

Travailler la vision dans l'espace.

De quoi a-t-on besoin ?

Les outils informatiques pour travailler en PostScript (voir page 351).

Les opérateurs PostScript (non standard) pour travailler directement avec des vecteurs à deux ou trois composantes (voir page 356).

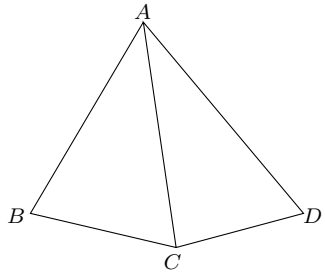
L'opérateur PostScript (non standard) `PPDP` qui permet de déterminer le point de percée d'une droite dans un plan, dont la définition est en annexe (page 511).

Prérequis. – Les sections 1 et 2 de ce chapitre, ainsi que la question concernant le point de percée d'une droite dans un plan, à la page 273 (chapitre 8, section 2.4).

Comment s'y prendre ?

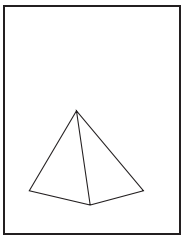
Soit $A = \begin{pmatrix} 200 \\ 120 \\ 300 \end{pmatrix}, B = \begin{pmatrix} 0 \\ 200 \\ 0 \end{pmatrix}, C = \begin{pmatrix} 300 \\ 0 \\ 0 \end{pmatrix}$ et $D = \begin{pmatrix} 400 \\ 200 \\ 0 \end{pmatrix}$.

Dessiner en PostScript le tétraèdre *opaque* $ABCD$.



Il y a une manière très rapide de représenter le tétraèdre (seules les faces ABC et ACD sont visibles) :

```
0 100 translate

                          newpath
/A [200 120 300] def      A Moveto B Lineto
/B [0 200 0] def          C Lineto D Lineto
/C [300 0 0] def          A Lineto C Lineto
/D [400 200 0] def        stroke
```



Même si la représentation du tétraèdre le fait apparaître opaque, le dessin ci-dessus n'est pas opaque. Un dessin est opaque lorsqu'il cache les dessins qui se trouvent en dessous de lui. En PostScript, c'est l'ordre dans lequel sont écrites les instructions de dessin qui détermine ce qui se trouve au-dessus ou en dessous : ce qui est dessiné d'abord se trouve en dessous. Travailler avec des dessins opaques est utile entre autres lorsque l'on veut représenter des parties vues et cachées.

Pour rendre opaque le dessin du tétraèdre, on peut procéder comme suit. L'opérateur `fill` permet de remplir une surface (c'est-à-dire l'intérieur d'un chemin) avec une certaine couleur. On va donc « peindre » le tétraèdre en blanc avant d'en dessiner les arêtes. L'opérateur `setgray` permet de déterminer le *niveau de gris* d'un dessin. Il prend une valeur dans la pile : `1` pour blanc, `0` pour noir, et toutes les valeurs intermédiaires pour les nuances de gris. On remplace alors les cinq dernières lignes ci-dessus par

```
1 setgray newpath A Moveto B Lineto
C Lineto D Lineto A Lineto fill
0 setgray newpath A Moveto B Lineto C Lineto D Lineto
A Lineto C Lineto stroke
```

Ajouter au dessin du tétraèdre opaque $ABCD$ la droite PQ, en ne représentant que ses parties visibles. Quatre positions différentes sont proposées pour P et Q.

Premier cas. – Soit $P = \begin{pmatrix} 20 \\ 30 \\ 50 \end{pmatrix}$ et $Q = \begin{pmatrix} 400 \\ 400 \\ 400 \end{pmatrix}$. Pour dessiner la ligne PQ, on ajoute

```
/P [20 30 50] def
/Q [400 400 400] def

newpath P Moveto Q Lineto stroke
```

On obtient la figure 9. Ceci ne donne aucune indication sur la visibilité de cette droite. On a évidemment l'impression qu'elle se trouve devant le tétraèdre. Mais c'est uniquement parce qu'elle a été dessinée après le tétraèdre. Si on la dessine avant (ce qui se fait simplement en écrivant les dernières instructions avant celles qui dessinent le tétraèdre), on obtient la figure 10. On a alors l'impression que la droite se trouve derrière le tétraèdre. Qu'en est-il exactement ?

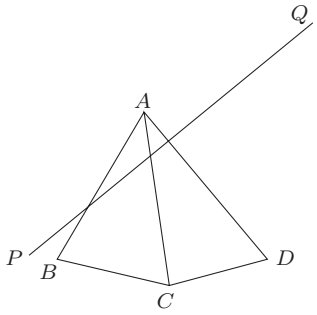


Fig. 9

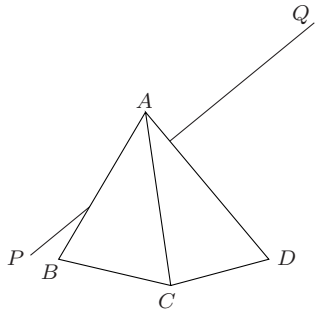


Fig. 10

Utilisons la macro `PPDP` pour déterminer la position du point de percée de la droite PQ dans le plan ABC :

```
A B C P Q PPDP Point.
```

On obtient la figure 11, ce qui montre que la droite PQ rencontre le tétraèdre à l'intérieur de cette face.

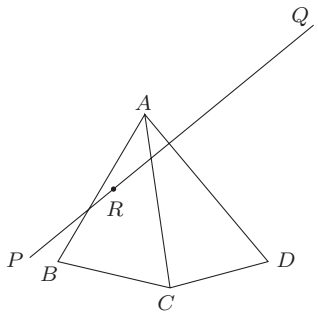


Fig. 11

Il reste deux possibilités pour dessiner la droite. Appelons R le point de percée de PQ dans ABC.

1. On dessine d'abord $[RQ]$, ensuite le tétraèdre et ensuite $[PR]$ (figure 12). On voit l'intérêt de travailler avec un dessin opaque. Si le dessin était

transparent, il aurait fallu déterminer les coordonnées de l'intersection de la droite représentant RQ avec le segment représentant l'arête AD du tétraèdre.

2. On dessine d'abord $[PR]$, ensuite le tétraèdre et ensuite $[RQ]$ (figure 13).

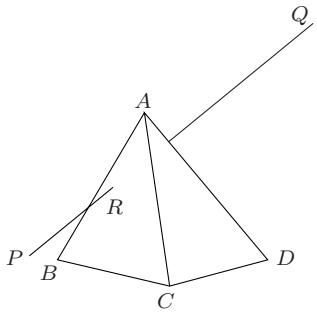


Fig. 12

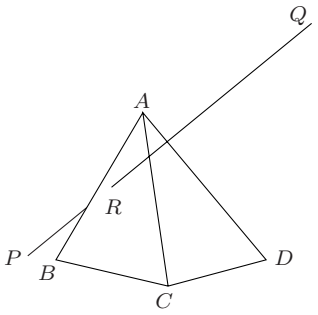


Fig. 13

Pour déterminer laquelle de ces deux représentations est correcte, on recherche l'intersection de la droite PQ avec le plan ACD :

```
A C D P Q PPDP Point.
```

La figure 14 montre que le point de percée se trouve cette fois en dehors du tétraèdre. Il ressort de la position du point S dans cette figure, que le deuxième point de rencontre de la droite et du tétraèdre se trouve dans la face « arrière ». Ceci n'est compatible qu'avec la figure 12, qui est par conséquent la bonne représentation.

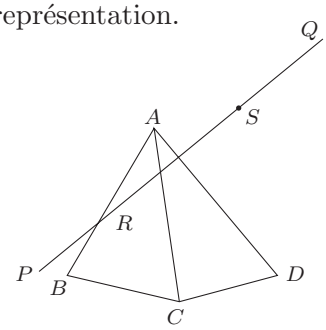


Fig. 14

Deuxième cas. – Soit $P = \begin{pmatrix} 20 \\ 30 \\ 50 \end{pmatrix}$ et $Q = \begin{pmatrix} 400 \\ 50 \\ 400 \end{pmatrix}$. On représente sur un seul dessin le tétraèdre, la droite PQ et les points de percée de PQ dans les faces ABC et ACD. La figure 15 semble montrer que la situation est la même que dans le cas précédent. Pourtant, lorsque l'on dessine la même chose, c'est-à-dire le segment $[RQ]$, le tétraèdre et ensuite le segment $[PR]$, on obtient la figure 16, qui n'est pas la même. En fait, les positions des représentations des points de percée ont pratiquement permuté. Pour s'en rendre compte, on dessine à nouveau les deux points de percée sur

deux dessins différents[8]. Dessinons le point de percée R de PQ dans ABC (figure 17), ce qui permet d'identifier les deux points de la figure 15. Les positions des points de percée indiquent que la droite PQ ne rencontre pas le tétraèdre et qu'elle se trouve devant lui. La figure 16 est donc la représentation correcte.

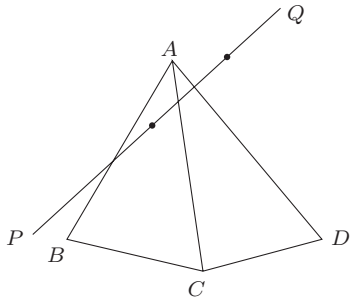


Fig. 15

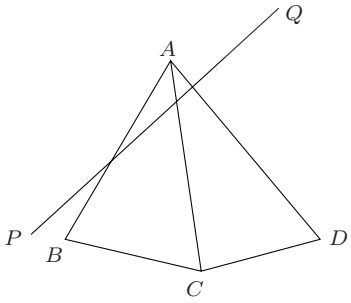


Fig. 16

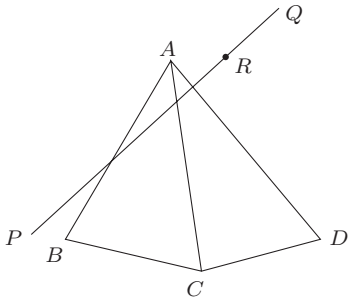


Fig. 17

Troisième cas. – Soit $P = \begin{pmatrix} 20 \\ 40 \\ 50 \end{pmatrix}$ et $Q = \begin{pmatrix} 400 \\ 180 \\ 400 \end{pmatrix}$. La figure 18 donne la représentation du point de percée R de PQ dans ABC. On complète cette représentation avec le deuxième point de percée (figure 19). On voit que la droite PQ rencontre les deux faces. La représentation des parties vues de la droite est donnée par la figure 20. Elle s'obtient en dessinant simplement les segments $[PR]$ et $[SQ]$.

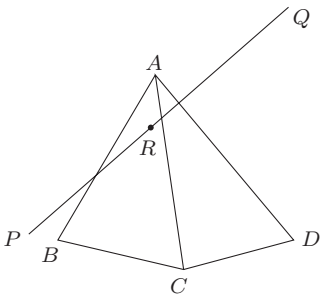


Fig. 18

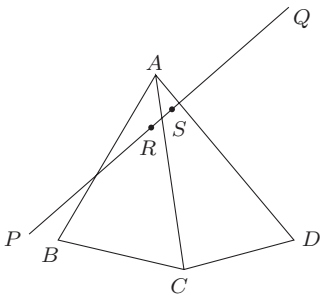


Fig. 19

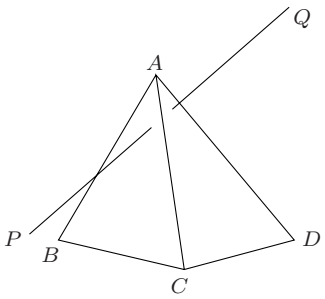


Fig. 20

Quatrième cas. – Soit $P = \begin{pmatrix} -50 \\ 200 \\ 0 \end{pmatrix}$ et $Q = \begin{pmatrix} 400 \\ 50 \\ 400 \end{pmatrix}$. La figure 21 donne la représentation du premier point de percée R dans le plan de la face ABC. On complète cette représentation avec le deuxième point de percée (figure 22). On voit que la droite PQ rencontre la face ACD mais pas la face ABC. La représentation est donnée par la figure 23.

[8] Lorsque le dessin est fait en PostScript, les deux points ne sont pas distincts l'un de l'autre et il faut donc trouver un moyen de les distinguer. Au lieu de le faire en produisant deux dessins différents, on peut dessiner deux points dont les apparences sont différentes.

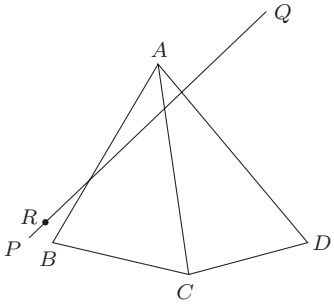


Fig. 21

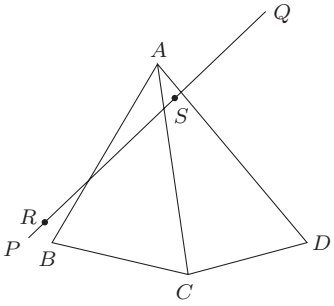


Fig. 22

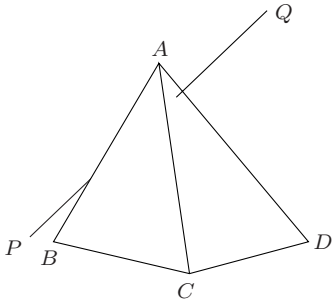


Fig. 23

Prolongement possible

Dans les situations précédentes, nous avons vu qu'il pouvait y avoir plusieurs possibilités pour dessiner les parties vues et cachées pour un même dessin. La position des points de percée de PQ dans les plans des faces visibles du tétraèdre permet de choisir celle qui convient. Deux questions assez différentes peuvent prolonger cette problématique.

Une même représentation pour plusieurs situations spatiales différentes. – Reprenons le dernier cas. En gardant les mêmes positions apparentes de tous les points, on peut dessiner la figure 24. Cette figure peut-elle représenter une situation réelle ?

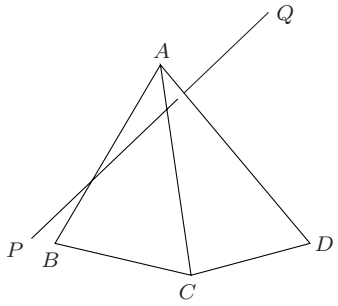


Fig. 24

Trouver des points P et Q tels que la représentation des parties vues et cachées de la droite PQ soit exactement celle de la figure 24.

Pour trouver de tels points « expérimentalement », il faut pouvoir modifier les positions des points P et Q (dans l'espace) sans modifier la position de leur représentation. Résoudre la question ci-dessus passe donc par une question intermédiaire, à savoir

Soit un point $A = \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix}$. Trouver tous les points $B = \begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix}$ ayant la même représentation en perspective que A.

Il faut examiner en détail comment une telle perspective est réalisée concrètement. C'est l'occasion de regarder de plus près l'opérateur PostScript

`Perspective` qui est utilisé avec les macros permettant le travail avec les vecteurs. Les coordonnées de la représentation d'un point $A = \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix}$ sont obtenues par le calcul

$$x_A e_1 + y_A e_2 + z_A e_3,$$

où les e_i sont les coordonnées des images des points

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \text{ et } \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

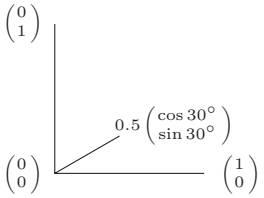


Fig. 25

Pour la représentation avec une fuyante à 30° et de rapport un demi, on a les coordonnées suivantes (cf. figure 25)

$$e_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \; e_2 = \begin{pmatrix} 0.5\cos 30^\circ \\ 0.5\sin 30^\circ \end{pmatrix} \text{ et } e_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Les points A et B ont donc même représentation si et seulement si

$$\left\{ \begin{array}{ccccccc} x_B & + & 0.5\cos 30^\circ y_B & = & x_A & + & 0.5\cos 30^\circ y_A \\ 0.5\sin 30^\circ y_B & + & z_B & = & 0.5\sin 30^\circ y_A & + & z_A. \end{array} \right.$$

Supposons que l'on fixe une valeur pour y_B. On trouve alors que

$$\left\{ \begin{array}{rcccl} x_B & = & x_A & + & 0.5\cos 30^\circ (y_A - y_B) \\ z_B & = & z_A & + & 0.5\sin 30^\circ (y_A - y_B). \end{array} \right.$$

On peut encore écrire ceci d'une autre manière, à savoir

$$\left\{ \begin{array}{rcccl} x_B & = & x_A & + & (y_B - y_A)(-0.5\cos 30^\circ) \\ y_B & = & y_A & + & (y_B - y_A) \\ z_B & = & z_A & + & (y_B - y_A)(-0.5\sin 30^\circ), \end{array} \right.$$

ou encore

$$\begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix} = \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix} + (y_B - y_A) \begin{pmatrix} -0.5\cos 30^\circ \\ 1 \\ -0.5\sin 30^\circ \end{pmatrix}.$$

La forme que l'on obtient n'est pas tellement étonnante. Les points B qui ont même représentation que A sont les points de la droite passant par A et parallèle à la direction de projection, et ce que l'on vient d'obtenir est bien l'équation vectorielle ou paramétrique d'une droite

$$\overrightarrow{OB} = \overrightarrow{OA} + \lambda \overrightarrow{v},$$

avec

$$\lambda = y_B - y_A \;\text{ et }\; \overrightarrow{v} = \begin{pmatrix} -0.5\cos 30^\circ \\ 1 \\ -0.5\sin 30^\circ \end{pmatrix}.$$

Ceci permet de programmer facilement un opérateur pour donner un autre point à partir du paramètre $y_B - y_A$, qui donne l'écart entre la nouvelle et l'ancienne valeur de la deuxième coordonnée. Pour l'utiliser, il faut placer dans la pile les coordonnées (sous forme vectorielle) du point de départ et la valeur de $y_B - y_A$. On a

```
/AutrePoint {
  /@d exch def [30 cos -2 30 sin] -0.5 Mul @d Mul Add
} def
```

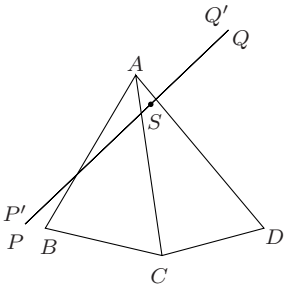


Fig. 26

Nous pouvons revenir maintenant à la figure 24. Nous allons faire varier un segment $[P'Q']$ de manière qu'il ait même projection (représentation en perspective) que le segment $[PQ]$ et que son point de percée dans la face ACD reste S (figure 26). Pour que le vu et caché corresponde à la figure 24, il faudra que le point de percée R' de $[P'Q']$ dans le plan de la face ABC se situe à gauche de S, mais à droite de la face ABC.

Si S reste fixe et que Q se déplace en Q', la position de P' est entièrement fixée : P' est aligné sur Q' et S ; sa projection coïncide avec celle de P. Considérons le plan contenant les droites PQ et $P'Q'$ qui se coupent en S. Puisque tous ces points ont même projection (représentation en perspective) les droites de projection sont contenues dans ce plan : ce sont les droites PP' et QQ' qui sont parallèles. Une telle situation est représentée à la figure 27.

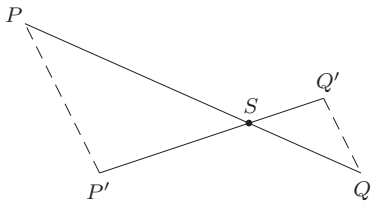


Fig. 27

Grâce au théorème de Thalès, nous savons que si μ est un scalaire tel que $\overrightarrow{SP} = \mu\overrightarrow{QS}$, alors on doit avoir $\overrightarrow{SP'} = \mu\overrightarrow{Q'S}$. Pour trouver la valeur de μ, on peut faire le rapport entre les différences des premières composantes

$$\mu \quad = \quad \frac{x_P - x_S}{x_S - x_Q},$$

ce que l'on traduit en PostScript par

```
/mu P S Sub 0 get S Q Sub 0 get div def.
```

On peut alors déterminer le point P' en fonction du point Q'. Dans ce qui suit, nous notons `Pp` le point P' et `Qp` le point Q'. Voici un exemple où la valeur 5 a été choisie pour $y_{Q'} - y_Q$, ce qui correspond donc à une augmentation de 5 pour la deuxième coordonnée de Q.

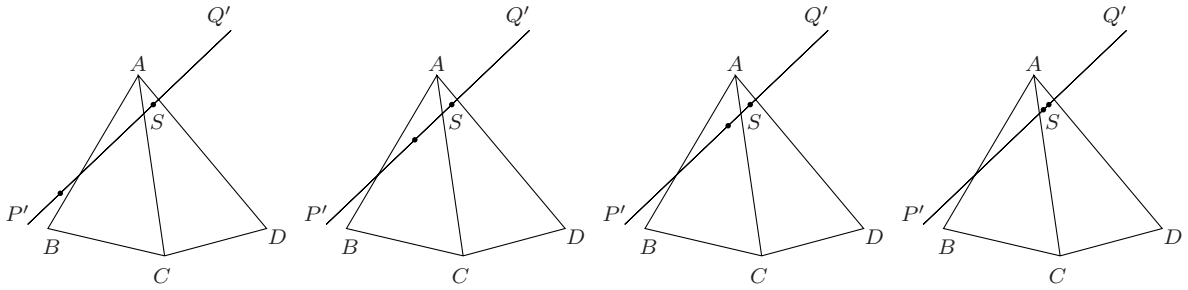
```
/Qp Q 5 AutrePoint def

/Pp S S Qp Sub mu Mul Add def
```

Le dernier calcul correspond à

$$P' \quad = \quad S + \mu(S - Q').$$

La valeur 5 choisie signifie donc que le point Q' « s'éloigne » puisque y augmente. Comme S reste fixe, le point P' « se rapproche ». Lorsqu'on augmente cette valeur, ce point se rapproche encore plus et le point de percée R' de $P'Q'$ dans la face ABC se déplace vers la droite.

La figure 28 montre diverses situations des points de percée pour des valeurs de $y_{Q'} - y_Q$ de plus en plus grandes.

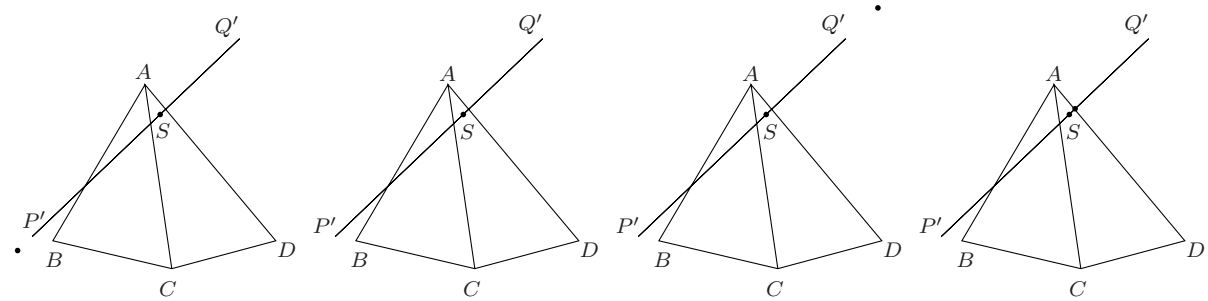


$y_{Q'} - y_Q = 5$ $y_{Q'} - y_Q = 50$ $y_{Q'} - y_Q = 100$ $y_{Q'} - y_Q = 500$

Fig. 28

Une situation qui répond à la question posée est donnée par $y_{Q'} - y_Q = 500$.

On peut terminer ce prolongement en examinant ce qui se passe lorsque l'on donne des valeurs négatives à $y_{Q'} - y_Q$. La figure 29 en montre quelques exemples. Une problématique intéressante est l'analyse de ce qui se passe entre $y_{Q'} - y_Q = -5$ et $y_{Q'} - y_Q = -50$.



$y_{Q'} - y_Q = -5$ $y_{Q'} - y_Q = -10$ $y_{Q'} - y_Q = -50$ $y_{Q'} - y_Q = -500$

Fig. 29

Commentaires

Les logiciels graphiques 3D, qui traitent les représentations en perspective d'objets de l'espace, ont des procédures pour déterminer automatiquement les parties vues et cachées. Il est hors de propos de vouloir « vider » cette question ici. Toutefois il est possible, pour ceux qui ont envie d'en savoir un peu plus, d'avoir une idée relativement générale de cette problématique qui prolonge celle que l'on a traitée dans cette activité, en avançant encore un peu vers l'automatisation des procédures de détection du vu et du caché.

Soit une figure plane opaque. Écrire un programme PostScript qui permet de représenter cette figure ainsi que les parties vues et cachées d'un segment $[PQ]$. Pour simplifier le programme, on suppose que ni P ni Q ne sont dans le plan de la figure.

Il faut déterminer les positions relatives de P et de Q relativement au plan de la figure. Soit R le point de percée de la droite PQ dans ce plan. Il y a quatre possibilités, à savoir

1. P et Q sont devant le plan : il faut dessiner la figure et ensuite le segment $[PQ]$.
2. P et Q sont derrière le plan : il faut dessiner le segment $[PQ]$ et ensuite la figure.
3. P est devant et Q est derrière : il faut dessiner le segment $[RQ]$, ensuite la figure et le segment $[PR]$ pour terminer.
4. P est derrière et Q est devant : il faut dessiner le segment $[PR]$, ensuite la figure et le segment $[RQ]$ pour terminer.

Il faut donc pouvoir déterminer si un point X est devant ou derrière le plan. Considérons le point X' qui se trouve dans le plan de la figure et dont la représentation en perspective coïncide avec celle de X. Si X est devant ce plan, X sera devant X'. Dans le système d'axes utilisé jusqu'ici (voir figure 25), ce sera le cas si la deuxième coordonnée de X est plus petite que celle de X'. Supposons que A, B et C soient trois sommets non alignés de la figure considérée. Le point X' est donné par le point de percée de la droite de projection passant par X dans le plan ABC. Les coordonnées d'un deuxième point de la droite de projection sont données par $\overrightarrow{OX} + \overrightarrow{v}$ où

$$\overrightarrow{v} = \begin{pmatrix} -0.5\cos 30^\circ \\ 1 \\ -0.5\sin 30^\circ \end{pmatrix}$$

donne la direction de projection.

Il y a plusieurs manières de programmer cela en PostScript, notamment en utilisant les opérateurs `if` ou `ifthen` pour réaliser des tests. Ci-dessous, nous présentons une manière de programmer moins classique qui évite ces tests.

Supposons que la figure soit un parallélogramme $ABCD$ tel que

$$A = \begin{pmatrix} 150 \\ 50 \\ 150 \end{pmatrix},\ B = \begin{pmatrix} 300 \\ 400 \\ 250 \end{pmatrix} \text{ et } C = \begin{pmatrix} 400 \\ 200 \\ 400 \end{pmatrix}.$$

La figure en question poura être dessinée par une macro `figure` définie comme suit :

```
/A [150 250 150] def
/B [300 400 250] def
/C [400 200 400] def
/D C A B Sub Add def
/figure {1 setgray newpath
         A Moveto B Lineto C Lineto D Lineto A Lineto fill
         0 setgray newpath
         A Moveto B Lineto C Lineto D Lineto A Lineto stroke
         } def
```

Supposons aussi que les points P et Q soient définis par :

```
/P [100 100 000] def
/Q [400 300 500] def
```

Nous allons maintenant définir la fonction `TestDevantDerriere` qui prend dans la pile les coordonnées du point à tester et qui renvoie dans la pile la valeur 1 ou 0 selon que le point est devant ou derrière le plan ABC. Le principe en est le suivant. Soit X le point à tester. On calcule $y' - y$ où y' est la 2$^{\text{e}}$ coordonnée du point du plan ABC ayant même représentation que X. Cette valeur est positive ou négative selon que X est devant ou derrière le plan ABC. En la divisant par sa valeur absolue, on obtient 1 ou -1. En ajoutant 1, cela fait 2 ou 0. En la divisant par 2, on obtient 1 ou 0. Appelons i_P et i_Q les valeurs ainsi obtenues pour P et Q. Le nombre

$$i = 2 \times i_P + i_Q$$

est entier et varie entre 0 et 3. Il vaut

0 si P et Q sont tous deux derrière le plan ;

1 si P est derrière et Q est devant le plan ;

2 si P est devant et Q est derrière le plan ;

3 si P et Q sont tous deux devant le plan.

Appelons

`DerriereDerriere`,

`DerriereDevant`,

`DevantDerriere` et

`DevantDevant`

les macros à exécuter dans chacun de ces cas. Il s'agit donc d'éxécuter la i^{e} composante de la liste

`[{DerriereDerriere} {DerriereDevant}{DevantDerriere} {DevantDevant}]`.

Pour pouvoir choisir cette i^{e} composante, il faut que i soit considéré par PostScript comme entier. Or, même si sa valeur est entière, il est considéré comme réel (au niveau de sa représentation informatique) et il faut le convertir en entier. C'est ce que fait l'opérateur `cvi`.

Pour exécuter la macro choisie, on utilise l'opérateur `exec`.

Voici les différentes macros :

```
/R A B C P Q PPDP def

/DevantDevant {figure newpath P Moveto Q Lineto stroke} def
/DerriereDerriere {newpath P Moveto Q Lineto stroke figure} def
/DevantDerriere {newpath R Moveto Q Lineto stroke figure
                 newpath P Moveto R Lineto stroke} def
/DerriereDevant {newpath P Moveto R Lineto stroke figure
                 newpath R Moveto Q Lineto stroke} def

/v [30 cos -2 30 sin] -0.5 Mul def

/TestDevantDerriere{/X@ exch def A B C X@ X@ v Add PPDP X@ Sub
                    1 get dup abs div 1 add 2 div cvi
                    } def

/i P TestDevantDerriere 2 mul Q TestDevantDerriere add def

[{DerriereDerriere} {DerriereDevant} {DevantDerriere}
{DevantDevant}] i get exec
```

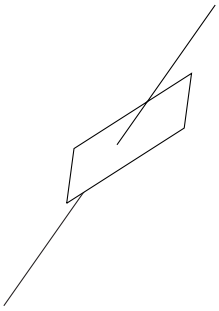


Fig. 30

La figure 30 montre le résultat de cette macro pour les valeurs indiquées.

Annexe IV

Ce qu'il faut savoir du PostScript

1 Calculer

1.1 L'ordre des opérations

L'ordre dans lequel on écrit les opérations n'est pas le même en PostScript que dans l'écriture mathématique courante. Ceci peut paraître un inconvénient dans la mesure où il est nécessaire de modifier des habitudes ancrées depuis l'école primaire. Mais cet inconvénient peut, en fait, être vu comme un avantage. De la même manière que pratiquer une langue étrangère aide à relativiser sa propre culture, pratiquer une autre formalisation du langage mathématique de base permet de relativiser la formalisation usuelle et de faire l'expérience de l'aspect purement conventionnel de ce type de notation.

Le PostScript, de même que d'autres langages informatiques, travaille avec ce qu'on appelle une *pile*. Il s'agit par exemple d'une pile de papiers (et non d'une pile électrique). On ne dépose des papiers qu'au-dessus de la pile. On ne prend en général – il y a des exceptions dans la réalité et aussi dans le PostScript – que ce qui est au-dessus. Lorsque l'on veut faire le calcul

$$3 + 4$$

en PostScript, on place 3 dans la pile, on place 4 et on additionne les deux derniers éléments de la pile

```
3 4 add
```

Lors de cette opération, le 3 et le 4 sont supprimés de la pile, et le résultat du calcul est placé dans la pile. Ce principe reste le même pour toutes les opérations. On place le nombre d'arguments nécessaires dans la pile. On écrit l'opération à faire. Les arguments sont effacés et le résultat est placé dans la pile.

Voici quelques exemples de calcul

(a) $3 + 4 + 5$: `3 4 add 5 add` ou `3 4 5 add add`
La première manière correspond à $(3+4)+5$, la deuxième à $3+(4+5)$.

(b) $6(5 - 9)$: `6 5 9 sub mul`

(c) $-\frac{5+6}{8}$: `5 6 add 8 div neg`

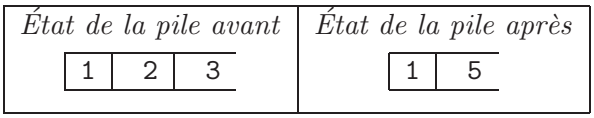
(d) $\frac{9-8}{5-3}$: `9 8 sub 5 3 sub div`

Chaque opérateur PostScript possède un nombre fixe d'arguments. L'opérateur en avale autant qu'il en a besoin. Ces arguments sont donc retirés de la pile l'un après l'autre, toujours en commençant par le dernier.

1.2 Les opérateurs arithmétiques

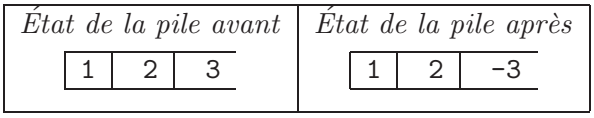
Voici les principaux opérateurs arithmétiques[32].

`add` Additionne les deux éléments supérieurs de la pile en les avalant et place le résultat au sommet de la pile.



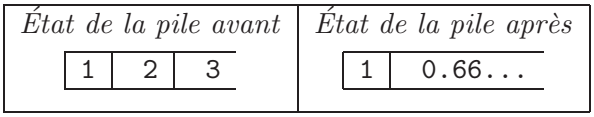
`sub` Effectue la soustraction des deux éléments supérieurs de la pile en les avalant et place le résultat au sommet de la pile.

`neg` Change le signe de l'élément supérieur de la pile.



`mul` Multiplie les deux éléments supérieurs de la pile en les avalant et place le résultat au sommet de la pile.

`div` Effectue la division des deux éléments supérieurs de la pile en les avalant et place le résultat au sommet de la pile.

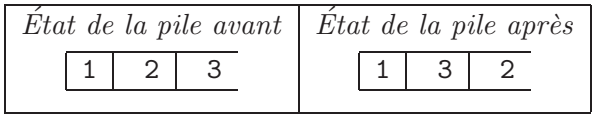


1.3 Gestion de la pile

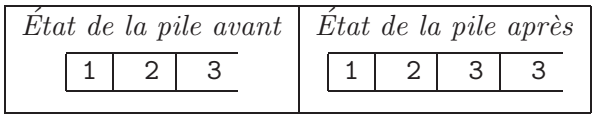
Certains opérateurs sont destinés à la gestion de la pile. Il y a, par exemple, l'opérateur `exch` qui échange les deux derniers éléments de la pile et l'opérateur `dup` qui duplique le dernier élément de la pile.

[32] Concernant la représentation des piles, voir la note 4 à la page 358.

`exch` Échange les deux derniers éléments de la pile.



`dup` Duplique le dernier élément de la pile.



2 Opérateurs pour le dessin

2.1 Définir des chemins

Un dessin est délimité par un chemin, en anglais *path*. Un chemin est constitué d'un point de départ et d'une suite de lignes, droites ou courbes. Une fois le chemin décrit, on peut le tracer (`stroke`) ou le remplir (`fill`). L'endroit du chemin où l'on est arrivé est appelé *point courant*.

`newpath` Définit un nouveau chemin. Après cet opérateur, le point courant n'est plus défini.

`moveto` Détermine le point de départ d'un chemin. Il prend deux arguments dans la pile : ce sont les coordonnées du point où commence le chemin.

`lineto` Ajoute un segment de droite au chemin. Le point de départ est le point courant. Il prend deux arguments dans la pile : ce sont les coordonnées de l'extrémité de ce segment.

`closepath` Termine le chemin en ajoutant un segment entre le point courant et le point de départ du chemin. Il ne prend pas d'arguments dans la pile.

`arc` Ajoute au chemin courant un arc de cercle. Il prend cinq arguments dans la pile : les deux coordonnées du centre, le rayon, l'angle de départ (par rapport à l'horizontale) et l'angle de fin.

2.2 Dessiner

`stroke` Trace un chemin dans la couleur courante. Il ne prend pas d'arguments dans la pile. Par défaut, la couleur est noire.

`fill` Remplit un chemin avec la couleur courante. Il ne prend pas d'arguments dans la pile. Par défaut, la couleur est noire.

`setgray` Définit un niveau de gris comme couleur courante à utiliser pour le tracé des chemins ou leur remplissage. Il prend un argument numérique dans la pile. Le noir est donné par le niveau `0`, le blanc par `1`, et les niveaux de gris intermédiaires par une valeur entre 0 et 1. On peut considérer le niveau de gris comme le rapport entre le nombre de pixels blancs et le nombre total de pixels d'une surface. Par défaut, le niveau de gris est mis à `0`.

2.3 Modifier le système d'axes

Au départ, l'origine du système de coordonnées est toujours le coin inférieur gauche de la feuille et les deux axes sont les bords de la feuille. L'unité sur les deux axes vaut $\frac{1}{72}$ pouce. Tout cela peut être modifié. Les modifications successives s'enchaînent les unes aux autres.

`translate` Déplace l'origine du système de coordonnées. Il prend deux arguments dans la pile : le déplacement horizontal et le déplacement vertical.

`rotate` Fait tourner les axes. Il prend un argument dans la pile : l'angle de rotation en degrés. Lorsque l'angle est positif, le sens de rotation est le sens trigonométrique.

`scale` Met à l'échelle les unités sur chacun des axes du système de coordonnées. Il prend deux arguments dans la pile : les facteurs d'échelle sur chacun des deux axes (ces facteurs peuvent être différents).

3 Définir des variables et de nouveaux opérateurs

3.1 Définir des variables

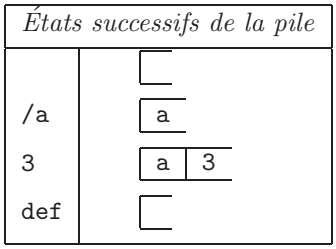
Le contenu de chaque niveau de la pile peut non seulement être une valeur numérique mais aussi un nom, ce qui permet de définir des variables.

Pour mettre un nom dans la pile il faut écrire le nom précédé de `/`. Par exemple, `/Nom` met `Nom` au dessus de la pile. Par contre, lorsque l'on écrit simplement `Nom`, c'est exactement la même chose que d'écrire le contenu de `Nom`.

Si l'on souhaite attribuer la valeur 3 à la variable `a`, il faut introduire :

```
/a 3 def.
```

Regardons les différentes étapes :



Après cela, c'est la variable `a` qui contient la valeur 3, et écrire `a`, c'est écrire 3, c'est-à-dire mettre 3 dans la pile :

État de la pile avant	*État de la pile après*
	3

On peut bien sûr définir une variable comme résultat d'un calcul :

```
/b a dup mul def
```

met le carré de `a` dans `b`. Regardons cela en détail :

États successifs de la pile	
`/b`	b
`a`	b 3
`dup`	b 3 3
`mul`	b 9
`def`	

Il est très utile de pouvoir mettre dans une variable une valeur qui est déjà dans la pile. On peut le faire en utilisant l'opérateur `exch` qui permet de mettre les arguments dans le bon ordre pour l'utilisation de l'opérateur `def`. Par exemple :

`a dup mul /b exch def` :

États successifs de la pile	
`a`	3
`dup`	3 3
`mul`	9
`/b`	9 b
`exch`	b 9
`def`	

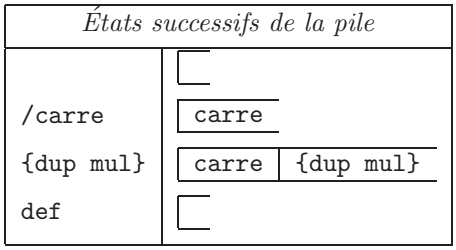
3.2 Définir de nouveaux opérateurs

Lorsque l'on écrit une suite d'instructions (nombres, opérateurs ou noms de variables) entre des accolades, ce n'est pas le résultat de cette suite d'instructions qui est placé dans la pile, mais bien la suite d'instructions elle-même. De cette manière il est possible de définir de nouveaux opérateurs. Il suffit de mémoriser une telle suite d'opérations dans un nom de variable. Lorsque l'on introduira par la suite le nom de cette variable, ce sera exactement comme si l'on écrivait cette suite d'instructions, qui sera donc exécutée à ce moment.

Voici par exemple une fonction permettant de définir le carré d'un nombre :

```
/carre {dup mul} def.
```

Le détail donne :



Lorsque par la suite on introduit par exemple

```
5 carre,
```

cela revient à introduire

```
5 dup mul.
```

4 Les listes

Voici quelques opérateurs qui sont utilisés soit directement, soit dans les macros permettant de travailler avec des vecteurs.

`length` Donne le nombre d'éléments d'une liste.

État de la pile avant	*État de la pile après*
`[1 2 3 4]`	`4`

`get` Met dans la pile un élément d'une liste (array). Il prend deux arguments dans la pile : une liste et un indice. Le premier indice est donné par `0`.

État de la pile avant	*État de la pile après*
`[1 2 3 4]` `2`	`3`

`array` Crée une liste vide. Cet opérateur prend un argument : la longueur de la liste à créer.

État de la pile avant	*État de la pile après*
`2`	`[· ·]`

`astore` Place des éléments de la pile dans une liste.

Par exemple,

```
4 3 2 1 4 array astore :
```

États successifs de la pile	
	-6 | 8
4 3 2 1 4	-6 | 8 | 4 | 3 | 2 | 1 | 4
array	-6 | 8 | 4 | 3 | 2 | 1 | [· · · ·]
astore	-6 | 8 | [4 3 2 1]

`aload` Place les éléments d'une liste dans la pile et recopie la liste dans la pile. Par exemple,

```
[1 2 3] aload :
```

États successifs de la pile	
	-5
[1 2 3]	-5 | [1 2 3]
aload	-5 | 1 | 2 | 3 | [1 2 3]

5 Opérateurs de contrôle

`repeat` Permet de répéter un certain nombre de fois des instructions. Elle demande deux arguments. Le premier est un nombre entier ; c'est le nombre de fois qu'il faut exécuter les instructions. Le deuxième est la suite des instructions à répéter. Ils doivent être mis entre accolades :

```
n { ... } repeat
```

`forall` Permet d'appliquer une suite d'instructions à tous les éléments d'une liste.

```
[. . . .] { ... } forall
```

Annexe V

Macros PostScript pour les vecteurs

```
/Add {/@a2 exch def dup /@a1 exch def
   length /@l exch def /@i 0 def
   @l {@a1 @i get @a2 @i get add /@i @i 1 add def} repeat
   @l array astore} def

/Sub {/@a2 exch def dup /@a1 exch def
   length /@l exch def /@i 0 def
   @l {@a1 @i get @a2 @i get sub /@i @i 1 add def} repeat
   @l array astore} def

/Mul {/@a exch def
   dup length /@l exch def
   {@a mul} forall
   @l array astore} def

/Div {/@a exch def
   dup length /@l exch def
   {@a div} forall
   @l array astore} def

/Neg {-1 exch Mul} def

/e1 [1 0] def
/e2 [30 cos 30 sin] 0.5 Mul def
/e3 [0 1] def

/Perspective {/@v exch def
   e1 @v 0 get Mul
   e2 @v 1 get Mul Add
   e3 @v 2 get Mul Add} def

/Coordonnees {dup length 3 eq {Perspective} if aload pop} def

/Moveto {Coordonnees moveto} def
/RMoveto {Coordonnees rmoveto} def
/Lineto {Coordonnees lineto} def
/RLineto {Coordonnees rlineto} def
```

```
/RayonPoint 5 def
/Point {gsave
    newpath
    Coordonnees RayonPoint 0 360 arc fill
    grestore} def
```

Annexe VI

Point de percée d'une droite dans un plan

La macro PPDP (Point de Percée d'une Droite dans un Plan) est donnée ici sans commentaires. Ceux-ci seront ajoutés dans une version ultérieure.

```
/PPDP {/E@ppdp exch def /D@ppdp exch def /C@ppdp exch def
       /B@ppdp exch def /A@ppdp exch def
       /AB@ppdp B@ppdp A@ppdp Sub def
       /AC@ppdp C@ppdp A@ppdp Sub def
       /fX {A@ppdp Sub /AX@ppdp exch def
            AX@ppdp 0 get AB@ppdp AC@ppdp detyz mul
            AX@ppdp 1 get AB@ppdp AC@ppdp detzx mul add
            AX@ppdp 2 get AB@ppdp AC@ppdp detxy mul add
            } def
       /fD@ppdp D@ppdp fX def
       /fE@ppdp E@ppdp fX def
       D@ppdp fE@ppdp Mul E@ppdp fD@ppdp Mul Sub
       fE@ppdp fD@ppdp sub Div
       } def
```