

Université de Rennes 1  
IRMAR

Documentation of the  
`fig4tex` macro-package  
(version 1.7)

Y. Lafranche and D. Martin



## Contents

1. Introduction
2. A first look
3. Definition of points
  1. Unit and scale
  2. The basic macro
  3. Text argument
  4. Symbolic constants
  5. Macros for vector and point generation
4. Drawing the figure
  1. Principle
  2. Basic drawing macros
  3. Example
5. Writing text on the figure
6. Writing text on an existing figure
7. Setting graphical attributes
  1. Introduction
  2. Line attributes
  3. Using color
8. Drawing macros
  1. Preliminary remarks
  2. Arc
  3. Example
  4. Curve
  5. Filled area
  6. Arrow
  7. Axes
  8. Triangle related macros
  9. Grid
  10. Flow chart
9. Helpers
10. Three-dimensional macros
  1. Introduction
  2. Projection
  3. Macro specifications
  4. 3D examples
11. Using this macro-package
12. List of user macros
13. Index



# Documentation of the fig4tex macro-package

(version 1.7)

**Abstract.** This document describes a set of  $\text{\TeX}$  macros designed to *create* a figure (as a *short* postscript file) at compilation time of the document, and to *write text* on it in a straightforward way, using the  $\text{\TeX}$  fonts, under total control of the user. These macros can also be used to write a legend on an existing figure, given as an encapsulated postscript file created by any other software. The macros have been designed from the user's point of view. They can be helpful to design a figurative drawing as well as to build an accurate geometric construction based on points in 2D or in 3D so that we can even think of this macro-package as a kind of geometric modeller.

## §1. Introduction

Inserting a figure given as a postscript file in a  $\text{\TeX}$  document has now become very common. There exist several sets of macros allowing to do this rather easily. With some other packages, it is also possible to create directly a postscript file. In that case, the  $\text{\TeX}$  macros are more or less a translation of postscript commands.

But, as far as we know, with these kinds of tools, there is no real *link* between the figure and the  $\text{\TeX}$  document, essentially because generally two softwares are needed: one for the  $\text{\TeX}$  document, one for the figure. In particular, a question that arises very often is: how can I write some text or caption using  $\text{\TeX}$ 's facilities (font, boxes, maths formulæ, . . .) at a *precise* point on the figure? An even worse case: if I have to modify my geometry for any good reason or change the observation angles in 3D, shall I have to make plenty of modifications to make everything fit together again?

The set of macros described below gives an answer to these questions. Yes, it is possible to write anything anywhere on a figure. No, provided that we have followed some elementary pieces of advice, any modification made at the top level of the construction automatically propagates to lower levels, allowing to change nearly anything at any time.

The macros have been designed from a *geometrical* point of view (i.e. so that geometric constructions are allowed), in such a way that they are easy to use and give an immediate result (only one compilation is necessary). Moreover, since the result of the process is a  $\text{\TeX}$  box, every usual  $\text{\TeX}$  commands can be applied to give the figure the right position inside the document.

The principle is the following. With the help of adequate macros, the user defines some characteristic points. These points are used to build the figure which is generated as a postscript file. Each of them can also be used as an attach point of a text printed on the figure. Thus, an interesting fact is to have the *same* geometric description both to draw the figure and to locate the text.

In the general case, three steps are needed:

- 1) definition of characteristic points necessary to build the geometrical skeleton of the figure,
- 2) creation of the postscript file corresponding to the geometry,
- 3) if needed, writing text on the figure, generally using the characteristic points, or other ones.

It must be emphasized that a figure already existing as an encapsulated postscript file, i.e. a postscript file specifying a correct bounding box, can be handled as well; step 2 has then to be skipped. A tool macro (`\figscan`) has been designed to determine accurately the bounding box: see the **Writing text on an existing figure** section for further information.

The set of macros consists of internal macros and user macros. Internal macros deal with technical issues like arithmetic and vectorial computation. They are not intended to be used directly from the  $\text{\TeX}$  document. On the contrary, the so-called user macros are higher level macros designed in such a way that most of the technical part of the work is hidden to the user. In order to make them as easy to use as possible, general syntactic rules have been settled:

- a. Except in text arguments, spaces are not significant.
- b. Whenever it is possible, the arguments are separated with commas (e.g. [3,5,7], (1.33, 0.57)). Colon (:), semi-colon (;) and equal sign (=) are also used as separators.
- c. Brackets ([ ]), parentheses (()) and slashes (//) are used to surround groups of arguments that are logically linked together. In most of the cases, brackets are devoted to point numbers, parentheses are devoted to numerical values. T<sub>E</sub>X's braces ({} ) are obviously always available.
- d. Numerical values are always given without any unit specification, except in two cases: one of them is described in the section **Writing text on the figure** (argument `Distance` of `\figwrite*`) and the other one in the section **Arrow** (`length` attribute of `\pssetdefault arrowhead`).

The user macros are divided in two main groups according to the beginning of their name: `\fig` or `\ps`. Macros beginning with `\ps` are commands used to create the postscript file. They are intended to be called during step 2, they have no effect otherwise. Macros beginning with `\fig` can themselves be divided in two classes, so-called mute and non-mute macros. Mute macros are mainly related to the definition of the geometry and can be called at any step. Non-mute macros are the only ones that write something on the figure. Logically, they have to be called during step 3 ; their names begin with `\figwrite`.

In the following sections, we describe the user macros. Then comes a list recalling the syntax of each of them and at last an index, which can be quite helpful to find examples where a given macro is used.

To catch the essential, the reader is advised to read this documentation at least up to the section **Writing text on the figure**, and then jump to any other specific topic.

## §2. A first look

Here is a short example to show how it works. Consider the following file:

```

\input fig4tex.tex
\newbox\demo
% 1. Definition of characteristic points
\figinit{pt}
\figpt 1:(80.5, 120)
\figpt 2:(-10, -10)
\figpt 3:(130, 20)
\figptbary 5:c.g.[1,2,3 ; 1,1,1]
% 2. Creation of the postscript file
\psbeginfig{Fig1.ps}
\psline[1,2,3,1]
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure 1}{
\figinsert{Fig1.ps}
\figwriten 1:(4)
\figwritew 2:(2)
\figwritee 3:(2)
\figsetmark{$\bullet$}\figwrites 5:$G$(4)
}
\centerline{\box\demo}
\bye

```

If we compile this file with T<sub>E</sub>X, then we obtain the figure 1. Some L<sup>A</sup>T<sub>E</sub>X users should read specific instructions in the section **Using this macro-package**. Now, let us comment these statements.

0. The first line loads the macro-package. Then we make `\demo` a private box to be used later.
1. Definition of the three vertices of a triangle. The system is first initialized and the unit to be used for the coordinates is set to `pt`. Each point is associated to a number whose value is chosen by the user. The text  $A_i$  is implicitly joined to point  $i$ . At last, the isobarycenter is defined as point 5, with `c.g.` as joined text.

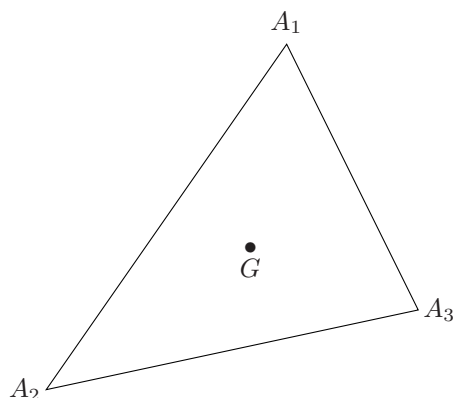


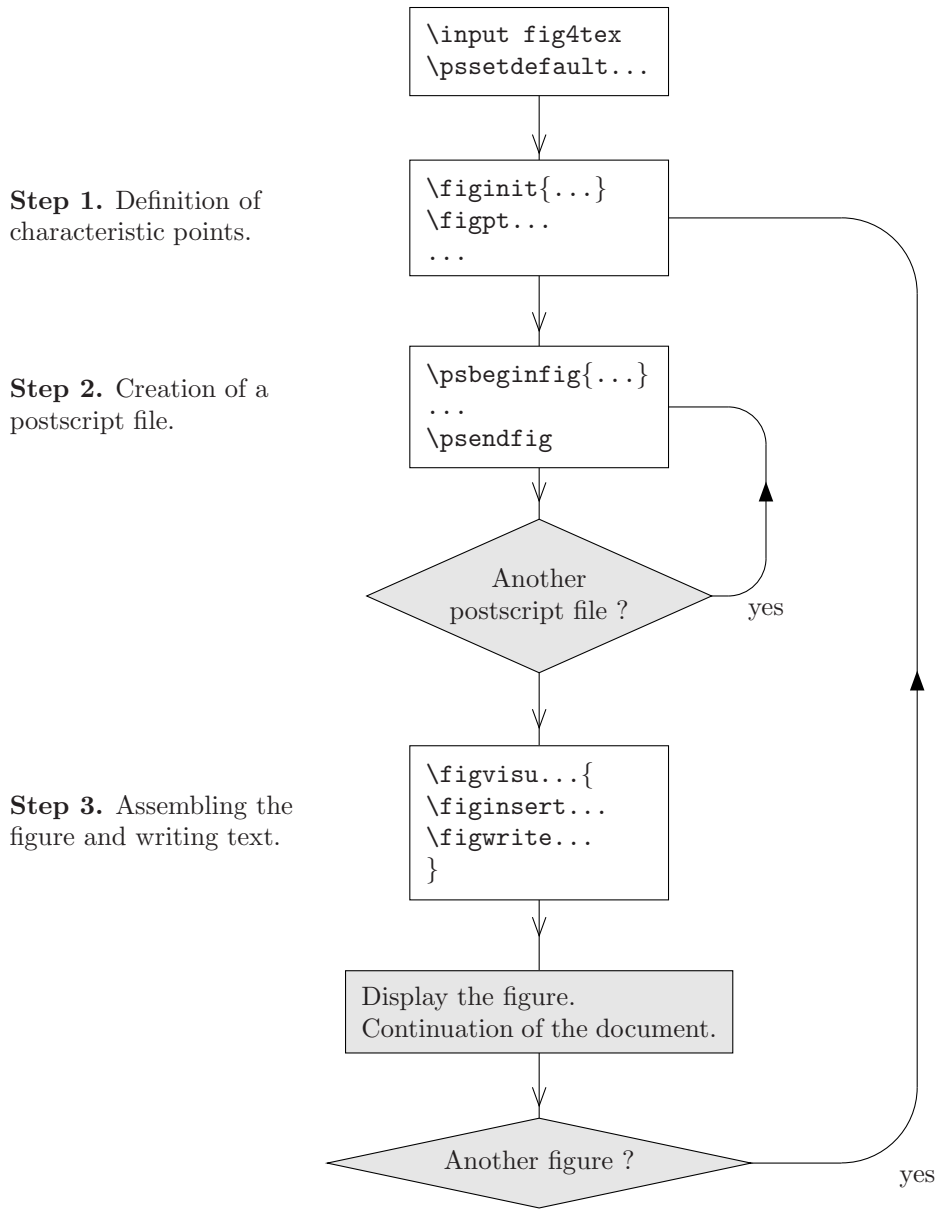
Figure 1

2. Creation of the postscript file that will contain the figure, here the three edges of the triangle, drawn as a single closed broken line. The name of the file, chosen by the user, usually has a `.ps` or a `.eps` suffix.
3. Writing text or comments on the figure. The macro `\figvisu` builds a box whose name (here `\demo`) is given as an argument, as well as the figure caption. The last argument consists of a set of commands that deal with the appearance of the figure. We refer to an existing postscript file, `Fig1.ps`, the one created just before, and “put” it into the box. The text associated to each point is printed at the place chosen by the user, i.e. at 4 pt towards the north for point #1, at 2 pt towards the west for point #2, at 2 pt towards the east for point #3, at 4 pt towards the south for point #5. Note that the text associated to point #5 is modified at this level. By default, the attach point of the text is not marked. However, for the point #5, it is necessary and the symbol `•` (`\bullet`) as been used.

Remarks:

- All the specifications, graphical and textual ones, are gathered into *one* `TEX` file.
- Textual specifications refer to attach points that have been previously defined. It must be emphasized that the definition of a point can take place anywhere in the process, that means even during step 2 or 3. For example, the point #5 could have been defined just before `\figwrites 5:$G$(4)`. Each attach point is referred to by its number.
- A point can be redefined at any time, the new coordinates are then valid until another redefinition occurs. This means that, except for the points defined during step 1 which are usually kept unmodified until the completion of the figure, many temporary points referring to the same number can be “defined-and-used” as many times as needed.
- By default, once created, the postscript file is reused as is during any further compilation of the source file. Indeed, the macros called between `\psbeginfig` and `\psendfig` are executed only during the creation of a *new* file. To re-create the postscript file, one can simply delete it and compile the source file again. This default behaviour can be modified: see the **Principle** and **Setting graphical attributes** sections for further information.
- It is possible to create several postscript files from the same source file. Each of them is given a name which is supplied first as an argument to the macro `\psbeginfig` for the creation and then to the macro `\figinsert` to use it. It must be emphasized that the *same* box (here `\demo`) can be used for several figures. For example, this documentation uses only three such boxes. Indeed, two or three boxes must be processed into a single group in order to display several figures side by side.
- Use of symbolic constants is possible and can be quite helpful (see the **Symbolic constants** subsection).
- If used, the `\magnification` command would apply to the whole document including the figure and its legends.
- Except for points numbers, no control is made on the validity of the arguments of the macros.
- In addition to the postscript files explicitly wanted by the user and the classical files produced by `TEX`, this package creates a temporary file whose name is `\jobname.anx`, which is empty at the end of the compilation. In fact, this is true only if a postscript file is created.

fig4tex macro-package  
instructions diagram



**Figure 2**

This figure shows an overview of how to use `fig4tex`. The text of the program that produces it is given in the subsection **Flow chart**.



### §3. Definition of points

#### 1. Unit and scale

First of all, the user has to choose the unit to be used for the figure among those defined in The `TEX`book:

```
pt  TEX point,
pc  pica (1 pc = 12 pt),
in  inch (1 in = 72.27 pt),
bp  big point, postscript unit (72 bp = 1 in),
cm  centimeter (2.54 cm = 1 in),
mm  millimeter (10 mm = 1 cm),
dd  didot point (1157 dd = 1238 pt),
cc  cicero (1 cc = 12 dd),
sp  scaled point (65536 sp = 1 pt).
```

The unit is given as an argument to the macro `\figinit` whose purpose is double: 1) set the unit and the scale factor, 2) initialize everything to the default state. If `\figinit` is not called, the default unit is `pt`. Because of 2), it must be called if several figures are defined in the same document.

Moreover, it is also possible to put a scale factor before the unit. It may be useful for example when the user wants to give every coordinates, say in `cm`, but wants the figure to appear twice bigger on the paper. In this case, he has to set a scale factor of 2 with `cm` as unit by saying `\figinit{2cm}`. The default scale factor is 1, so for example `\figinit{in}` is equivalent to `\figinit{2.54cm}`.

Notice that although 1 pt is nearly equal to 1 bp, it is not a good idea to use one unit instead of the other because the difference is most of the time visible on the figure.

#### 2. The basic macro

The following action is the definition of points. The basic macro is `\figpt` whose prototype is

```
\figpt NewPt :Text(X,Y)
```

The first argument `NewPt` is a positive integer chosen by the user. The theoretical limit to this value is  $2^{31} - 1$  and is imposed by `TEX`. This is not a problem since, in practice, huge numbers are useless. Precisely, because of memory limitations, it is not wise to use big numbers. On the contrary, the user is *advised* to choose small numbers first, but must feel free to define point `#400` or `#1500`, which could be seen as a wise limit. Notice that 3-digit numbers are especially useful to denote points in 3D.

The second argument `Text` is called “joined text” to this point (see below). The last arguments are the coordinates  $(X, Y)$  of the point with respect to the user Cartesian axes. We recall that the unit (see above) must be omitted.

#### 3. Text argument

To each point the user defines, a text is implicitly joined, namely  $A_i$  for point  $i$ . The macro `\figsetptname` sets this default text by saying `\figsetptname{\$A_{#1}\$}`, where `#1` stands for the point number, and it can be used to modify this default. For instance, the command `\figsetptname{\$X^{(#1)}\$}` changes it to  $X^{(i)}$ . To remove the automatic association of text, just say `\figsetptname{}`. Since the effect of this macro is dynamic, from a logical point of view, this macro has to be called during step 3. This allows for example to write the names of some points, then call `\figsetptname{}` which now hides the implicit name of every point.

If the text argument is not empty, it overrides the implicit text. More generally, the last text associated with the point will be printed. This is often done by the non-mute macros (whose name begin with `\figwrite`).

A text argument can be any valid `TEX` material, that is to say in practice:

- nothing. Implicit text is then used.
- free text where spaces are significant. To print one delimiter, like `:` or `(`, just enclose the whole text into braces `{}`.
- a box. Inside a `\vbox`, use `\hfill\break` instead of `\par` and think of `\parindent0pt` if needed.

By default, the position of the point is not displayed. However, it is sometimes needed and the user can choose anything he wants among  $\text{T}_{\text{E}}\text{X}$ 's possibilities to highlight the point location. For this purpose, marks like  $\bullet$ ,  $\circ$ ,  $\cdot$ ,  $\times$ ,  $+$  are most commonly chosen and can be set by the macro `\figsetmark{Mark}`. To get back to the default state, just say `\figsetmark{}`.

In fact, the following marks are not quite vertically centered: `*` (`\ast`), `\bullet` (`\bullet`), `\circ` (`\circ`) and `\diamond` (`\diamond`). So corrected versions have been created, to be used instead if a very accurate result is needed. Their names are `\figAst`, `\figBullet`, `\figCirc`, `\figDiamond`. We can observe the difference on this figure:



Remark: By the way, for the same reason, do not use the macro `\cdot`: simply say `\figsetmark{.}`.

At last, it is possible to print the coordinates of the point on the figure in the unit chosen for the `\figinit` call. To do that, just call the macro `\figcoord{Ndec}` in the definition of the text argument. In the previous example, one could have written during step 1:

```
\figt 3:$A_3\scriptstyle\figcoord{2}$(130, 20)
```

or during step 3:

```
\figwritet 3:$A_3\scriptstyle\figcoord{2}$(2)
```

The argument `Ndec` of `\figcoord` is a positive integer setting the number of decimals to be printed.  $\text{T}_{\text{E}}\text{X}$  provides at most 5 decimals. Missing decimals, for integer values for example, are printed as 0. By default, the printed values are rounded up to the integer (`Ndec = 0`), or the first, second or third decimal, according to the value of the argument `Ndec`. Notice that if `Ndec > 3`, the values are not rounded: we obtain the internal raw value. To suppress this rounding, one can say `\figsetroundcoord{no}`, and then `\figsetroundcoord{yes}` to enable it again for subsequent printed values.

Remark: The two macros `\figsetptname` and `\figcoord` can be combined to print together the point name and its coordinates by saying, for example, `\figsetptname{$A_{\#1}$ \figcoord{1}}`. This will automatically apply for any point, except for those bearing a specific joined text, as just said.

#### 4. Symbolic constants

As already mentioned in the remarks following the first example, a symbolic constant can be used instead of any numerical value. This can be quite useful for example when a dimension has to be passed as an argument to several macros. This is shown in the following lines :

```
\def\Absone{12.3456}
\figt 10:(\Absone, 120)
\figt 11:(\Absone, -10)
```

The points `#10` and `#11` have the same abscissa which is define just before as a  $\text{T}_{\text{E}}\text{X}$  macro.

Moreover, the macro

```
\figgetdist\Value [Pt1,Pt2]
```

computes the euclidian distance between the two points `Pt1` and `Pt2`, and “puts it” in the macro `\Value` whose name is chosen by the user. The macro can then be used as a symbolic numerical constant. A typical example is the computation of the radius of a circle (see the macro `\pscirc`). This feature allows us to build geometrical constructions depending only on the coordinates of characteristic points.

Let  $C$ ,  $P_1$  and  $P_2$  stand for `Center`, `Pt1` and `Pt2`. The companion macro of the previous one is

```
\figgetangle\Value [Center,Pt1,Pt2]
```

which computes the value (in degrees) of the oriented angle  $(\overrightarrow{CP_1}, \overrightarrow{CP_2})$ .

Remark: The name of the postscript file to be created can also be put in a macro definition. The reference to the file is then done via this macro, `\MyPSfile` in the following example:

```

% 2. Creation of the postscript file
\def\MyPSfile{Figure.ps}
\psbeginfig{\MyPSfile}
...
% 3. Writing text on the figure
\figvisu{\demo}{Caption}{
\figinsert{\MyPSfile}
...

```

## 5. Macros for vector and point generation

The name given to each of the macros described below is governed by the following rule:

- . a macro whose name begin with `\figvect` produces a vector,
- . a macro whose name begin with `\figpt` produces one point,
- . a macro whose name begin with `\figpts` produces at least one point.

### Vector creation

The definition of a vector can be done with one of the two macros

```

\figvectC NewVect (X,Y)
\figvectP NewVect [Pt1,Pt2]

```

The first macro defines the vector by its components (X,Y) and the second one by the origin Pt1 and the end point Pt2. The word `NewVect` stands for a number chosen by the user in the *same* set as the one used for the definition of a point. That means that vectors and points are internally stored in the same way. The user must remember if, say #10, is a point or a vector ; no check is performed if a point is used instead of a vector or vice-versa.

The macro

```

\figvectN NewVect [Pt1,Pt2]

```

defines a vector normal to the line (Pt1, Pt2). Precisely, the vector is the image of  $\overrightarrow{P_1P_2}$  by a  $\pi/2$  rotation, where  $P_1$  stands for Pt1 and  $P_2$  stands for Pt2.

The macro

```

\figvectNV NewVect [Vector]

```

defines a vector normal to the vector `Vector`. Precisely, `NewVect` is the image of `Vector` by a  $\pi/2$  rotation.

The macro

```

\figvectU NewVect [Vector]

```

defines a unitary vector in the direction given by `Vector`, taking into account the unit and the scale factor chosen by the user. Thus, after `\figinit{2cm}`, the length of `NewVect` is 2cm. This macro is helpful when scaling is important ; it can be used together with `\figgetdist`.

The macro `\figvectDBezier` computes the derivative at a point lying on a Bézier curve. It is described at the end of this section.

### Point transformation

We describe here macros that transform one point into another. The simplest one corresponds to the identity transformation. It may happen that it is convenient to copy points, for example to save the result of a geometric transformation to be used later. The macro

```

\figptcopy NewPt :Text/Pt/

```

has been built for this purpose. It makes `NewPt` be a copy of `Pt`, changing the joined text to `Text`.

Now here is a set of macros corresponding to classical geometrical transformations, namely homothety, rotation, symmetry and translation. Roughly speaking, these macros use the following syntactic scheme:

```

\macroname Result = Data /transformation definition/

```

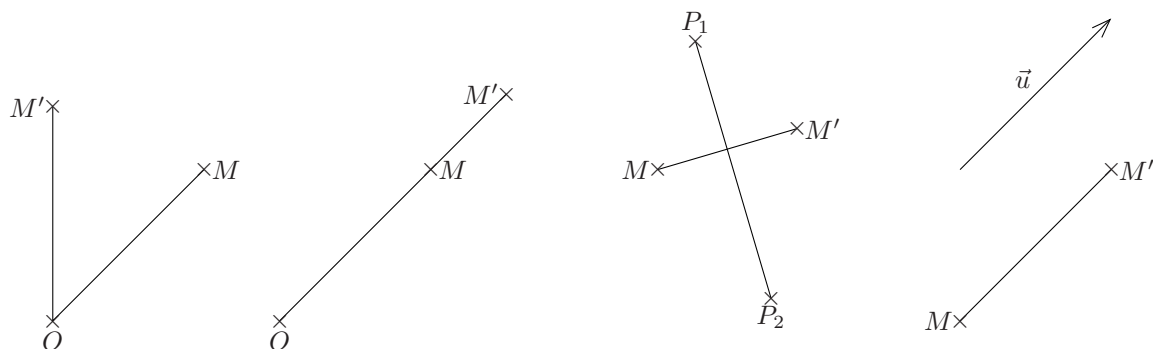


Figure 4 Rotation ( $\pi/4$ ), homothety (ratio 1.5), symmetry /  $[P_1, P_2]$  and translation of vector  $\vec{u}$

```

% 1. Definition of characteristic points
\figinit{cm}
\figvectC 5(3,0)
\figpt 0:$O$(0,0)\figpt 1:$M$(2,2)\figptrot2:$M'$=1/0,45/           % rotation
\figptstra10=0,1/1,5/\figpthom12:$M'$=11/10,1.5/                   % homothety
\figpttra21:$M'$=1/2,5/
\figpttraC23:$P_1$=21/0.5,1.7/\figpttraC24:$P_2$=21/1.5,-1.7/
\figptsym22:$M'$=21/23,24/                                         % symmetry
\figpttra31:$M'$=0/4,5/\figvectC 30(2,2)\figpttra32:$M'$=31/1,30/ % translation
\figpttraC34:=31/0,2/\figpttra35:=34/1,30/                         % vector u
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psline[1,0,2]              % rotation
\psline[10,11,12]          % homothety
\psline[21,22]\psline[23,24] % symmetry
\psline[31,32]\psarrow[34,35] % translation
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}\Rotation ( $\pi/4$ ), homothety (ratio 1.5),
symmetry /  $[P_1, P_2]$  and translation of vector  $\vec{u}$ {
\figinsert{\MyPSfile}\figsetmark{\$ \times \$}
\figwrites 0:(0.15)\figwritee 1:(0.1)\figwritew 2:(0.1)           % rotation
\figwrites 10:$O$(0.15)\figwritee 11:$M$(0.1)\figwritew 12:(0.1) % homothety
\figwritew 21:(0.1)\figwritee 22:(0.1)                            % symmetry
\figwriten 23:(0.1)\figwrites 24:(0.1)                            % symmetry
\figwritew 31:(0.1)\figwritee 32:(0.1)                            % translation
\figsetmark{\}\figptbary36:$\vec{u}$[34,35;1,1]\figwritenw 36:(0.1) % translation
}
\centerline{\box\demo}

```

Their prototypes along with their definition are given below. They compute the image called `NewPt` of a given point called `Pt`. A text can be joined to each `NewPt`.

```

\figpthom NewPt :Text= Pt /Center, Ratio/
  refers to the homothety of center Center and of ratio Ratio,
\figptrot NewPt :Text= Pt /Center, Angle/
  refers to the rotation defined by the center Center and the angle Angle given in degrees,
\figptsym NewPt :Text= Pt /LinePt1, LinePt2/
  refers to the orthogonal symmetry with respect to the line defined by the points LinePt1 and LinePt2,
\figpttra NewPt :Text= Pt /Lambda, Vector/
  refers to the translation of vector Lambda * Vector, where Lambda is any real value.

```

A “light” version of the translation macro exists. It is especially useful to build a path joining several points whose relative distances are known. The components (X,Y) of the vector are directly given as arguments. Its prototype is:

```
\figpttraC NewPt :Text= Pt /X,Y/
```

The figure 4 shows the image  $M'$  of a point  $M$  computed by these macros. It is followed by the text of the program that produces it.

Remark: Previously, `\MyPSfile` has been defined as the name of the postscript file and the box `\demo` has been made private. Also, the figures are automatically numbered by the `\newcount` register `\Figno`. This applies to all the examples given further.

The previous transformations are sufficient in most cases. However, one may want to use another transformation that cannot be written as a combination of them. For this purpose, the more general macro

```
\figptmap NewPt :Text= Pt /InvPt/A11,A12 ; A21,A22/
```

allows to define any linear affine mapping. More precisely, if  $I$  and  $A$  respectively stand for an invariant point `InvPt` and the matrix whose coefficients are  $A_{ij}$ ,  $1 \leq i \leq 2$ ,  $1 \leq j \leq 2$ , the image  $M'$  of the point  $M$  is given by  $\overrightarrow{IM'} = A \overrightarrow{IM}$ .

Thanks to the translation, this macro allows to defined any affine transformation.

Some examples are given on figure 5:  $F_1$  defines an elongation along the  $x$  axis, leaving every point lying on the line  $(I; \vec{y})$  invariant,  $F_2$  defines an elongation along the  $y$  axis, leaving every point lying on the line  $(I; \vec{x})$  invariant, as well as  $F_3$  which defines a “shearing” mapping.

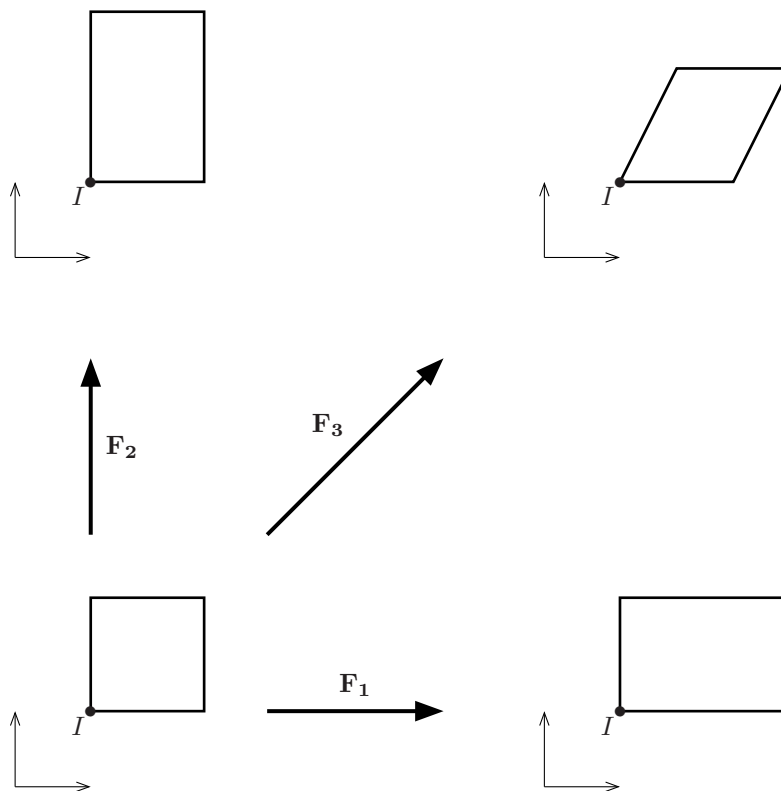


Figure 5

To make some geometrical constructions easier, a “set” version of each of these macros have been written. They compute the image of a set of  $N$  data points called `Pt1`, `Pt2`, ..., `PtN` which can be given in any order. The numbers associated with the result points are successive. Only the first number `NewPt1`, chosen by the user, is given as argument. So, if `NewPt1` is equal to  $k$ , then the points created have numbers

$k, k + 1, \dots, k + N - 1$ . Text eventually previously associated with the result points is lost and moreover no text can be joined to the result points.

Due to the fact that the result points have successive numbers, the choice of the first number must be done carefully. Let  $I_r$  (resp.  $I_d$ ) the set of the result points numbers (resp. the set of the data points numbers). Let  $J = I_r \cap I_d$ .

- If  $I_r = I_d$ , or  $J$  is empty, or `NewPt1` does not belong to  $J$  (which is generally the case in practice), then there is no problem.
- Otherwise, the result given by the macro *may be wrong*, at least partially: this is because, in this case, `NewPt1` belongs to  $J$ , and the data points are taken into account sequentially, beginning from the first element in the list. However, we can take advantage of this fact to compute, in a quite simple way, a sequence of points  $P_{n+1} = f(P_n)$ ,  $n = 1, \dots, N$  by just saying

`\macroname P2 = P1,P2, ... PN /transformation definition/`

Since these macros are extensions of the previous ones `\figptxxx` and can produce more than one result, their names are `\figptsxxx`. The arguments defining the transformation are the same as for the corresponding “unary” version. Here are their prototypes:

```

\figptsrot NewPt1 = Pt1, Pt2, ..., PtN /Center, Angle/
\figptshom NewPt1 = Pt1, Pt2, ..., PtN /Center, Ratio/
\figptssym NewPt1 = Pt1, Pt2, ..., PtN /LinePt1, LinePt2/
\figptstra NewPt1 = Pt1, Pt2, ..., PtN /Lambda, Vector/
\figptsmap NewPt1 = Pt1, Pt2, ..., PtN /InvPt/A11,A12 ; A21,A22/

```

The text of the program that produces the figure 5 is the following:

```

% 1. Definition of characteristic points
\figinit{cm}
\figpt 0:(0,0) % Initial origin
\def\SQEL{1.5} % Square edge length
% Vertices of the initial square
\figpt 1:(1,1)
\figpttraC 2:=1/\SQEL,0/\figpttraC 3:=2/0,\SQEL/\figpttraC 4:=3/-\SQEL,0/
\def\dist{7}\def\TV{50} % Translation distance and name of the vector
% For each case, the initial figure is translated. Then the transformation
% is applied.
% 1st transformation
\figvectC \TV(\dist,0)\figptstra 10=0,1,2,3,4/1,\TV/
\figptsmap 11=11,12,13,14/11/1.5,0; 0,1/ % Check point 11 is invariant !
\figptbary 15:[1,11;4,2]\figptbary 16:[1,11;2,4] % Arrow end points
% 2nd transformation
\figvectC \TV(0,\dist)\figptstra 20=0,1,2,3,4/1,\TV/
\figptsmap 21=21,22,23,24/21/1,0; 0,1.5/
\figptbary 25:[1,21;4,2]\figptbary 26:[1,21;2,4]
% 3rd transformation
\figvectC \TV(\dist,\dist)\figptstra 30=0,1,2,3,4/1,\TV/
\figptsmap 31=31,32,33,34/31/1,0.5; 0,1/
\figptbary 35:[1,31;4,2]\figptbary 36:[1,31;2,4]
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psset arrowhead(length=0.2)
\psaxes 0(1)\psaxes 10(1)\psaxes 20(1)\psaxes 30(1)
\psset(width=1)
\psline[1,2,3,4,1]\psline[11,12,13,14,11]
\psline[21,22,23,24,21]\psline[31,32,33,34,31]

```

```

\psset(width=1.5)\psset arrowhead(fill=yes,length=0.4)
\psarrow[15,16]\psarrow[25,26]\psarrow[35,36]
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPSfile}
% Name of each transformation
\figptbary 0:[15,16;1,1]\figwriten 0:\bf F$_{\hbox{\bf\scriptstyle 1}}$$(0.2)
\figptbary 0:[25,26;1,1]\figwritee 0:\bf F$_{\hbox{\bf\scriptstyle 2}}$$(0.2)
\figptbary 0:[35,36;1,1]\figwritenw 0:\bf F$_{\hbox{\bf\scriptstyle 3}}$$(0.2)
% Mark each invariant point
\figsetmark{\figBullet}\figwritesw 1,11,21,31:$I$(0.1)
}
\centerline{\box\demo}

```

The last macro is

```
\figptorthoprojline NewPt :Text= Pt /LinePt1, LinePt2/
```

that computes the orthogonal projection `NewPt` of `Pt` onto the line defined by the points `LinePt1` and `LinePt2`. Its corresponding “set” version is then:

```
\figptsorthoprojline NewPt1 = Pt1, Pt2, ..., PtN /LinePt1, LinePt2/
```

Remark: Although the work done by this macro can be obtained using `\figptmap`, this form is interesting because it is more practical. This remark also apply to the macros `\figpthom`, `\figptrot` and `\figptsym`.

### Point computation

Once points and vectors are defined, we would like to make some elementary geometrical computation.

First of all, it is very convenient to define a point as the barycenter of other ones. The macro `\figptbary` allows us to do that. Its prototype is

```
\figptbary NewPt :Text [Pt1,... ,PtN ; Coef1,... ,CoefN]
```

The resulting point is the barycenter of a set of  $N$  points whose numbers are given as a comma separated list, `Pt1, ... ,PtN`. Each point bears an *integer* coefficient. The coefficients are given in the corresponding order, also as a comma separated list of  $N$  elements, `Coef1, ... ,CoefN`. The user has to choose the number `NewPt` and can associate a text with it.

Another version of this macro, `\figptbaryR`, exists. In this case, the coefficients assigned to each point are *real*. Both versions are useful, depending on the context. `\figptbaryR` is slightly slower than `\figptbary`, so the latter is preferable if there is no particular reason to use `\figptbaryR`.

As we will see in following sections, it is possible to draw circles and more generally ellipses. So we need a macro to create points lying on an ellipse. The macro

```
\figptell NewPt :Text: Center;XRad,YRad (Angle,Inclination)
```

creates the point `NewPt`, with an associated `Text`, lying on the ellipse defined by its `Center`, its radius `XRad` in the X direction, its radius `YRad` in the Y direction (in local axes) and its `Inclination`, which is the angle between the local axes of the ellipse and the absolute axes corresponding to the paper sheet. The position of the point is set by the parametrization `Angle` measured with respect to the local axes. The coordinates of the point are then classically computed as  $(X_{Rad} \cos \alpha, Y_{Rad} \sin \alpha)$  where  $\alpha$  stands for `Angle`. The two angles, `Angle` and `Inclination`, are to be given in degrees.

Another version of this macro, `\figptellP`, exists. The difference comes from the definition of the ellipse with three points. Its prototype is

```
\figptellP NewPt :Text: Center,PtAxis1,PtAxis2 (Angle)
```

Let  $C, A_1, A_2$  stand for `Center`, `PtAxis1` and `PtAxis2`. The ellipse is defined by its center  $C$  and the end points of its two axes  $A_1$  and  $A_2$ . Thus, the local axes are defined by the orthogonal basis  $(\vec{CA_1}, \vec{CA_2})$ . The macro creates the point `NewPt`, with an associated `Text`, lying on the ellipse at the position set by the

parametrization `Angle` given in degrees, starting from the half-line  $(C, A_1)$  and measured counterclockwise around the vector  $\overrightarrow{CA_1} \times \overrightarrow{CA_2}$ .

Since we handle circles more often than ellipses, the macro

```
\figptcirc NewPt :Text: Center;Radius (Angle)
```

has been written to obtain the result in a more straightforward way. This calling sequence is equivalent to

```
\figptell NewPt :Text: Center;Radius,Radius (Angle,0)
```

Then, we often need to compute the intersection of two lines. This can be done with the macro

```
\figptinterlines NewPt :Text[LinePt1,Vector1; LinePt2,Vector2]
```

It computes the intersection `NewPt` of the line defined by the point `LinePt1` and the vector `Vector1`, and the line defined by the point `LinePt2` and the vector `Vector2`. As usual now, the user has to choose the number of the result point and can associate a text with it.

In the same idea, the macro

```
\figptsintercirc NewPt1 [Center1,Radius1 ; Center2,Radius2]
```

computes the intersection of two circles. More precisely, we consider the two circles defined by their center and radius, here respectively  $(\text{Center1}, \text{Radius1})$  and  $(\text{Center2}, \text{Radius2})$ . Only one result point number is to be given as argument ; if `NewPt1` is equal to  $k$ , then the second point `NewPt2` is equal to  $k + 1$ . They must obviously be different from `Center1` and `Center2`. The points `NewPt1` and `NewPt2` are ordered so that the angle  $(\text{NewPt1}, \text{Center1}, \text{NewPt2})$  is positive. If the two circles do not intersect, then on return `NewPt1` is set to `Center1` and `NewPt2` is set to `Center2`.

The macro

```
\figptsinterlinell NewPt1 [Center,XRad,YRad,Inclination ; LinePt1,LinePt2]
```

computes the intersections of the line defined by the two points `LinePt1` and `LinePt2`, and the ellipse defined by `Center`, `XRad`, `YRad` and `Inclination` in the same way as previously for the macro `\figptell1`. Also, as explained just above, if `NewPt1` is equal to  $k$ , then the second point `NewPt2` is equal to  $k + 1$ . Let  $A_1, A_2$  stand for `LinePt1`, `LinePt2` and  $P_1, P_2$  stand for `NewPt1`, `NewPt2`.  $P_1$  and  $P_2$  must be different from  $A_1$ . On output,  $P_1$  and  $P_2$  are ordered so that the vector  $\overrightarrow{P_1P_2}$  has the same direction as  $\overrightarrow{A_1A_2}$ . If the line does not intersect the ellipse, then on return `NewPt1` is set to `LinePt1` and `NewPt2` is set to `LinePt2`.

A “point” version of this macro exists. Its prototype is:

```
\figptsinterlinellP NewPt1 [Center,PtAxis1,PtAxis2 ; LinePt1,LinePt2]
```

The only difference from the previous one consists in the definition of the ellipse by the three points `Center`, `PtAxis1` and `PtAxis2`, which are explained above where the macro `\figptell1P` is presented.

## Macros related to Bézier curves

Now, here are some more specific macros related to a cubic Bézier arc. Although some of them do not seem quite necessary at first glance, they are used internally especially for 3D computations. Consequently, they have been made public.

Let us briefly recall that a cubic Bézier arc is a parametric curve  $B$  whose parameter value  $t$  is taken in  $[0, 1]$  in order to benefit from some mathematical properties. Each point  $B(t)$  can then be defined as the barycenter of four points, called *control points*, the corresponding weights being the Bernstein cubic polynomials evaluated at  $t$ .

For each of the three following macros, we consider a cubic Bézier curve  $B$  defined by the four control points `Pt1`, `Pt2`, `Pt3` and `Pt4`.

The first macro

```
\figptBezier NewPt :Text: t [Pt1,Pt2,Pt3,Pt4]
```

computes the point  $B(t)$ , namely `NewPt`, lying on  $B$  for the parameter value `t`. We recall that, as a consequence of the definition, for `t = 0`, `NewPt = Pt1`, and for `t = 1`, `NewPt = Pt4`.

The next step is to compute the derivatives  $B'(t)$  and  $B''(t)$ . The macro

```
\figvectDBezier NewVect : n, t [Pt1,Pt2,Pt3,Pt4]
```



computes the vector `NewVect` corresponding to the derivative of order `n` of  $B$  for the parameter value `t`. The order `n` must be equal to 1 or 2.

We are thus ready to introduce the macro

```
\figptcurvcenter NewPt :Text: t [Pt1,Pt2,Pt3,Pt4]
```

which computes the curvature center `NewPt` at the point lying on  $B$  for the parameter value `t`.

The macro

```
\figptscontrol NewPt1 [Pt1,Pt2,Pt3,Pt4]
```

computes the two control points `NewPt1` and `NewPt2` so that the cubic Bézier curve defined by the control points `Pt1`, `NewPt1`, `NewPt2` and `Pt4` interpolates the four points `Pt1`, `Pt2`, `Pt3` and `Pt4` with respective parameter values of 0, 1/3, 2/3 and 1. Applying the rule setting the point numbers already seen, if `NewPt1` is equal to  $k$ , then the second point `NewPt2` is equal to  $k + 1$ .

The macro

```
\figptscontrolcurve NewPt1, \NbArcs [Pt0,Pt1,...,PtN,PtN+1]
```

computes the control points used by the macro `\pscurve` when it is invoked by the corresponding calling sequence `\pscurve [Pt0,Pt1,...,PtN,PtN+1]` whose effect is to draw a curve that consists in  $N - 1$  cubic Bézier arcs (see section **Curve**). If `NewPt1` is equal to  $k$ , then the points created have numbers  $k, k + 1, \dots, k + M$  with  $M = 3(N - 1)$ . `\NbArcs` denotes a **T<sub>E</sub>X** macro whose name is chosen by the user. It can be considered as an output variable whose value is set to  $N - 1$  by `\figptscontrolcurve` so that the calling sequence `\psBezier \NbArcs [NewPt1, NewPt1+1, \dots, NewPt1+M]` draws exactly the same curve, provided that the value of the curve roudness has not been changed. As a consequence, the data points `Pti`,  $i = 1, \dots, N$  are duplicated on output: indeed, the point number  $k + j$  is equal to the point number `Pti` where  $j = 3(i - 1)$ ,  $i = 1, \dots, N$ . The numbers assigned to the points created must be different from any of the data points `Pt0`, `Pt1`,  $\dots$ , `PtN`, `PtN+1`.

Remarks:

- this macro uses the current value of the curve roudness, which must logically be the same used by `\pscurve`,
- since the value of `\NbArcs` can be known in advance, specifying this macro could have been avoided, but it helps to check the number of points created and manage them,
- this macro is mainly intended to help to compute a point lying on the curve drawn by `\pscurve`, using `\figptBezier` in a second step.

### Special cases

We mention here some macros that compute points linked to a particular context and are usefull to write something at these points. The macro `\figptendnormal` computes the end point of a vector and is described in the section **Triangle related macros**. The macro `\figptsaxes` computes the end points of axes and is described in the section **Axes**.

## §4. Drawing the figure

### 1. Principle

The first macro to call is `\psbeginfig` whose only argument is the name of the postscript file to be created. It initializes all the settings related to the drawing macros.

To command the end of the postscript file being created, just say `\psendfig`. Between these two statements, macros that generate the figure are called. Their name begin with `\ps`.

We recall that, by default, if the file already exists, it is not updated. The user has to delete it to get a new version. However, during the tuning time, this can be somewhat cumbersome. So to enforce `TEX` to update the postscript file at each compilation, the user can insert the command `\psset(update=yes)` before step 2. When the file is satisfactory, this command has to be removed, otherwise the file will be unusefully recreated at each compilation.

WARNING ! We must be aware of what we do : clearly, the contents of the file are destroyed. So, if for some reason the filename given as argument to `\psbeginfig` is the one of a precious file while the update mode is active, the file will be destroyed as well . . .

Between the two commands `\psbeginfig` and `\psendfig`, all the points created up to now can be used, but they can be redefined, as well as new points can be created. Notice that drawing macros and computing macros are executed *only* if the postscript file is created or updated. In other words, every drawing or computing macro called between `\psbeginfig` and `\psendfig` is ignored if the postscript file exists and no update is wanted. This ensures future compilations to run faster.

A variety of attributes can be tuned in order to obtain the expected graphical result. They include general settings, for example color, line with and style, but also attributes related to a specific graphic object, like the opening angle of an arrow-head. They are all set by the macro `\psset`.

Each of these attributes have a default value defined by the authors. It may happen however that some of these choices do not meet a general agreement: for example, one may prefer to have the postscript files always updated by default, or to use a particular color, line width or arrow-head opening angle by default. For this reason, the macro `\pssetdefault` has been created in order to customize the way this macro-package is used. These macros are described in the section **Setting graphical attributes**.

### 2. Basic drawing macros

Then, the first thing we need to draw is a line. The macro

```
\psline[Pt1,Pt2,...,PtN]
```

draws a broken line joining every point from `Pt1` to `PtN` in this order. To close the path, just let the last point number `PtN` be equal to the first one `Pt1`. In this case, the meeting of the two segments at `Pt1` is properly handled, according to the line join attribute (see section **Line attributes**).

Two other versions of this macro exists. Their prototypes are

```
\pslineC(X1 Y1, X2 Y2,..., XN YN)
\pslineF{Filename}
```

For the first macro, the coordinates of the points defining the line are given as argument as a comma separated list. The coordinates of each point are separated from each other by at least one blank space. The coordinates may appear on several lines, and even each group of coordinates on one line at a time.

For the second macro, the coordinates of the points are read from a text file whose name `Filename` is passed as argument to the macro. Each line of the file must contain the coordinates of one point according to the format `x y` in 2D, or `x y z` in 3D. Notice that two coordinates are separated by a white space.

In both cases, the line is closed if the group of coordinates of the last point is exactly identical to that of the first point.

Circles are also among the geometrical entities which are the most often drawn. This is the reason why a specific macro has been built. It draws a circle defined by its center and its radius, and its prototype is

```
\pscirc Center (Radius)
```

### 3. Example

Now, let us examine an example to illustrate some of the macros presented so far. First, draw a circle centered on the origin  $O$  passing through a given point  $M$ . Note the use of the macro `\figgetdist` to compute the radius. Then, create the point  $P$ , intersection of the circle and the X axis, and the point  $Q$  such that  $\overrightarrow{PQ} = 1.5 \overrightarrow{OP}$ . At last, draw the circle with center  $Q$  and radius  $PQ$ .

We can see the result on the figure 6. The text of the program that produces it follows. Notice the scale and the unit. Moreover, since  $M$  is the only true data, if we modify its coordinates, we just have to recompile this program to get the corresponding result (see the previous subsection called **Principle** about the update process).

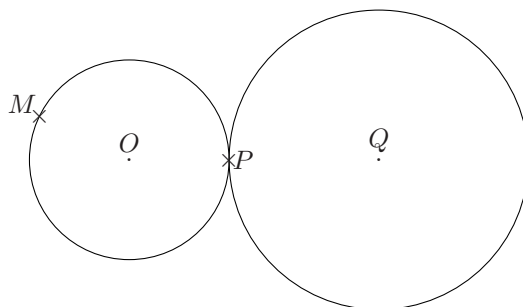


Figure 6

```
% 1. Definition of characteristic points
\figinit{0.5in}
\figpt 1:$O$(0,0)\figpt 2:$M$(-0.9354, 0.45467)
\figgetdist\DistOM[1,2]\figvectC 10(\DistOM,0)
\figpttra 3:$P$=1/1,10/\figpttra 4:$Q$=3/1.5,10/
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\pscirc 1(\DistOM)
\figgetdist\DistPQ[3,4]\pscirc 4(\DistPQ)
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPSfile}
\figsetmark{${\times}}\figwritenw 2:(0.04)\figwritee 3:(0.04)
\figsetmark{.}\figwriten 1,4:(0.08)}
\centerline{\box\demo}
```

## §5. Writing text on the figure

Since the postscript file is now prepared, it remains to make it appear at the right place on the page along with the appropriate legend. To do that, we strongly recommend to use the macro `\figvisu`, although it is possible to do without it, but probably more painfully. Its prototype is:

```
\figvisu{Vbox}{Caption}{Commands}
```

As we have already seen in the previous examples, the first argument is the name of a box register that the user has to make private by the `\newbox` command. This box is filled according to the third argument and is the result of this macro. Note that the macro produces a *vertical* box.

The second argument is the figure caption. It can be void, but can also be, for example, a box containing several lines.

The third argument is a list of commands that make the legend and the drawing fit together. In this part, extra blank lines or `\par`, are forbidden. Moreover, any user provided macro should end with `\ignorespaces`.

To show the drawing, one has to insert it in the box with the macro `\figinsert` whose first argument is the name of the postscript file containing the drawing description. This is done by a separate macro rather than a fourth argument to `\figvisu` because it allows to insert several drawings in the same box. The drawings will then overlap since the origin is common. The second argument of `\figinsert` is a scale factor. This argument is *optional*; it is described in the section **Writing text on an existing figure**.

Once this has been done, the last thing to do, before showing the contents of the box, is to write text on the figure. Several macros have been designed for that. We recall that a text is written on the figure by specifying its position with respect to an attach point. This point can be made visible with the macro `\figsetmark` already seen.

The simplest macro is `\figwritep`[Pt1, Pt2, ..., PtN] which writes the point marker set by `\figsetmark` at the locations defined by the points Pt1, Pt2, ..., PtN.

Then come the macros:

```
\figwrite [Pt1, Pt2, ..., PtN]{Text}
\figwritec [Pt1, Pt2, ..., PtN]{Text}
```

The first one writes a `Text` after the points Pt1, Pt2, ..., PtN according to **T****E****X**'s alignment, while the second one writes the text *centered* on these positions. These two macros do not print the point marker.

At first glance, it may look strange that the same text is written at each point. This is the case if the argument `Text` is present and this may actually be useful. However, if an empty text is given, recalling the definition of this argument (given in the **Text argument** subsection), the macro writes the text given when the point was defined, or the implicit text which can be modified by `\figsetptname`.

This rule also applies to the following macros which are the most useful because they allow to put a text at a prescribed position with respect to the attach points. We refer to a compass-card notation to specify the position of the text. To write `Text` at a given `Distance` to the west, the east, the north or the south of the point Pt<sub>*i*</sub>, we can use the macros:

```
\figwritew Pt1, Pt2, ..., PtN :Text(Distance)
\figwritee Pt1, Pt2, ..., PtN :Text(Distance)
\figwriten Pt1, Pt2, ..., PtN :Text(Distance)
\figwrites Pt1, Pt2, ..., PtN :Text(Distance)
```

To write to the north-west, the south-west, the north-east or the south-east of the point Pt<sub>*i*</sub>, we can use the macros:

```
\figwritenw Pt1, Pt2, ..., PtN :Text(Distance)
\figwritesw Pt1, Pt2, ..., PtN :Text(Distance)
\figwritene Pt1, Pt2, ..., PtN :Text(Distance)
\figwritese Pt1, Pt2, ..., PtN :Text(Distance)
```

For the different cases, the figure 7 shows the position of the corresponding text with respect to the attach point marked with the + sign. To avoid hiding the + sign, the text in central position is represented by an empty rectangle.

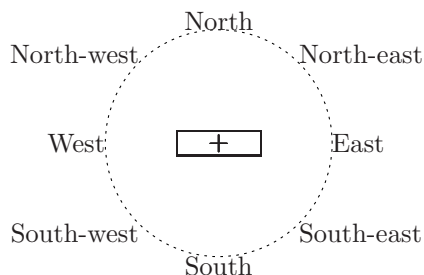


Figure 7

The text of the program that produces this figure is:

```
% 1. Definition of characteristic points
\figinit{cm}
\def\Dist{1.5}\figpt 0:(0, 0)
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psset(dash=5)\pscirc 0(\Dist)
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPSfile}\figsetmark{+}
\figwritew 0:West(\Dist)\figwritee 0:East(\Dist)
\figwriten 0:North(\Dist)\figwrites 0:South(\Dist)
\figwritenw 0:North-west(\Dist)\figwritesw 0:South-west(\Dist)
\figwritene 0:North-east(\Dist)\figwritese 0:South-east(\Dist)
\figwritec[0]{\boxit{1pt}{\phantom{Center}}}
% or equivalently:
\figsetmark{\boxit{1pt}{\phantom{Center}}}\figwritep[0]
}
\centerline{\box\demo}
```

Remarks:

- Here, the 8 first calls to `\figwrite*` write the + sign 8 times at the same place. Similarly, the calls to `\figwritec` and `\figwritep` both write the same rectangle.
- The dashed circle shows the distance between the text and the attach point. For the show, we have chosen a “large” distance, but in practice of course the text is much closer to the attach point.
- Generally, we want the point-text distance to be independent of the scale chosen by the `\figinit` call. Thus, the dimension of the figure can be tuned without acting on this distance. To obtain this result, just specify a unit after the numerical value (see last example). This is one exceptional case where a dimension can be given with a unit specification. (This works in the same way as the `TEX` `\magnification` command: to get a *true* unit, you give a dimension specifying a unit like `truept`).

Another set of macros have been designed in order to take into account the depth of the box containing the `Text` argument. In particular, this feature allows to control the vertical position of the *baseline* of the text to be written. The prototypes of theses macros are :

```
\figwritebw Pt1, Pt2, ..., PtN :Text(Distance)
\figwritebe Pt1, Pt2, ..., PtN :Text(Distance)
\figwritebn Pt1, Pt2, ..., PtN :Text(Distance)
\figwritebs Pt1, Pt2, ..., PtN :Text(Distance)
```

For the “west” and “east” macro, `Distance` measures the length between `Pti` and the end (resp. the beginning) of the `Text` argument, while the baseline of the `Text` is set to `Pti`’s ordinate.

For the “north” and “south” macro, the Text argument is horizontally centered and Distance measures the length between Pti and the baseline of the Text.

Theses macros have to be compared with the macros `\figwriteX`. We can see the difference on figure 8 which is followed by the program that produces it.

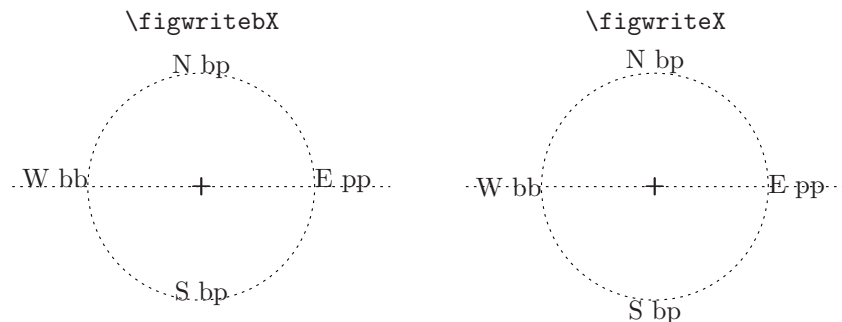


Figure 8

```

% 1. Definition of characteristic points
\figinit{cm}
\def\Dist{1.5}\def\DDist{2}
\figpt 0:(0,0)\figpt 1:(-2.5,0)\figpt 2:(2.5,0)
\figvectC 20(6,0)\figptstra 10=0,1,2/1,20/
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psset(dash=5)
\pscirc 0(\Dist)\psline[1,2]
\pscirc 10(\Dist)\psline[11,12]
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPSfile}
\figsetmark{+}
\figwritebw 0:W bb(\Dist)\figwritebe 0:E pp(\Dist)
\figwritebn 0:N bp(\Dist)\figwritebs 0:S bp(\Dist)
\figwriten 0:\CtrlSq{figwritebX}(\DDist)
%
\figwritew 10:W bb(\Dist)\figwriteee 10:E pp(\Dist)
\figwriten 10:N bp(\Dist)\figwrites 10:S bp(\Dist)
\figwriten 10:\CtrlSq{figwriteX}(\DDist)
}
\centerline{\box\demo}

```

These “baseline” macros are useful in particular when the alignment of the text is important. This is the case for example when we have to write close to a horizontal line. On figure 9, we can observe that the `\figwritebX` macros lead to the best result. On the contrary, the right hand side of the figure does not look so nice since the different texts are not horizontally aligned.



Figure 9

All the previous macros are sufficient most of the time, but for particular cases, they are not. That is why two general macros have been designed. Their prototypes are:

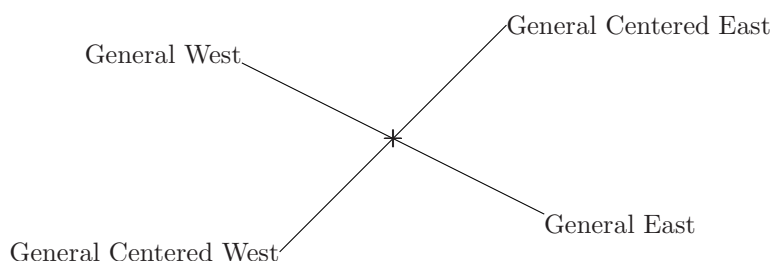
```
\figwritew Pt1, Pt2, ..., PtN :Text(DistanceX,DistanceY)
\figwritee Pt1, Pt2, ..., PtN :Text(DistanceX,DistanceY)
```

They write a `Text` placed, with respect to the point `Pti`, at a horizontal distance `DistanceX` (respectively west or east) and a vertical distance `DistanceY` from the bottom of the bounding box of the `Text` argument if `DistanceY > 0`, from the top if `DistanceY < 0`. If `DistanceY = 0`, the `Text` argument is vertically centered with respect to `Pti`.

The last two macros are internally used by some of the previous ones. They allow a very fine tuning that cannot be achieved otherwise. Their prototypes are:

```
\figwritegcw Pt1, Pt2, ..., PtN :Text(DistanceX,DistanceY)
\figwritegce Pt1, Pt2, ..., PtN :Text(DistanceX,DistanceY)
```

They do the same job as the two previous ones except that `DistanceY` measures the vertical distance between the point and the mid-height of the text. This is shown on the figure 10 where the vectors (`DistanceX,DistanceY`) are drawn from the attach point.



**Figure 10**

The attach point is the origin (point #0), marked with the + sign. The unit is cm. Notice that in the first call the unit is specified. The corresponding calls are:

```
\figsetmark{+}
\figwritew 0:General West(2,10mm)
\figwritegcw 0:General Centered West(1.5,-1.5)
\figwritegce 0:General Centered East(1.5,1.5)
\figwritee 0:General East(2,-1)
```

## §6. Writing text on an existing figure

To write something on an existing figure, any unit can be chosen. For the demonstration, we consider the triangle drawn in the file `Fig1.ps` used at the beginning. If we want to write something at the right of the triangle, we can choose `cm` as unit and write the following commands:

```
\figinit{cm}
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}{
\figpt 1:(4,3)
\figwrite [1]{\vbox{\hbox{Something at the right}\hbox{of the triangle}}}
\figinsert{Fig1.ps}
}
\centerline{\box\demo}
```

Thus, we get the result shown on the figure 11.

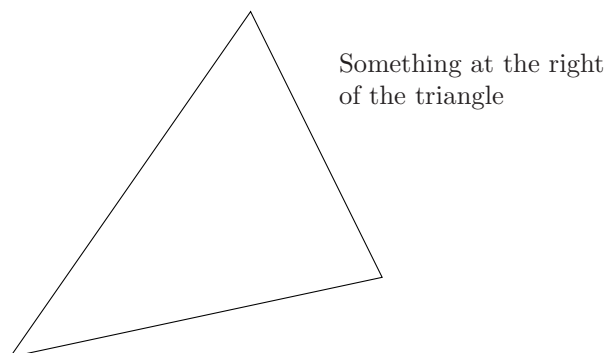


Figure 11

Notice that the macro `\figinsert` needs an encapsulated postscript file to work correctly, because it needs the bounding box of the figure. If the file does not contain this information, a message is printed on the screen and in the log file at compilation time, and a small default bounding box is used. In this case, the best thing to do is to provide one by editing the file and adding such a line in the header (after the first line which is a control line):

```
%%BoundingBox: Xmin Ymin Xmax Ymax
```

where `Xmin` and `Xmax` generally range between 0 and 540, and `Ymin` and `Ymax` range between 0 and 720 (these values correspond to the whole page). It must be noticed that some graphical softwares do not set a correct bounding box. Obviously, the position of the figure on the page depends on these values, so it is recommended to set a bounding box as close as possible to the real figure. This can be done after a few trials with the help of the macro `\figscan` presented below or using `ghostview`. For example, the fifth line of the header of the file `Fig1.ps` is:

```
%%BoundingBox: -9.96262 -9.96262 129.514 119.55139
```

The values of `Xmin` and `Ymin` are negative. This is not an error and is due to the coordinates of the points, but the drawing must be translated to appear entirely on the page, which is done by the macro `\figvisu`.

In the previous example, the coordinates of the attach point of the text need not to be set very accurately and we used integers. However, it sometimes happens that one wants to refer to a particular point on the figure. For that reason, we will have to give the position in postscript units, so the unit to use must be `bp`. Except in the rare case where the user explicitly knows the postscript coordinates of the point, the problem is precisely to obtain easily these coordinates.

One solution may be to use `ghostview`, a postscript previewer available on most operating systems. When the postscript file is processed by this software, the drawing appears on the screen and the position of the mouse is shown cross-shaped when moved on it. At the same time, the postscript coordinates of the position of the mouse is given at the top left corner of the window, varying in real time with each motion. We can take advantage of this to pick the postscript coordinates of any point of the drawing and put them



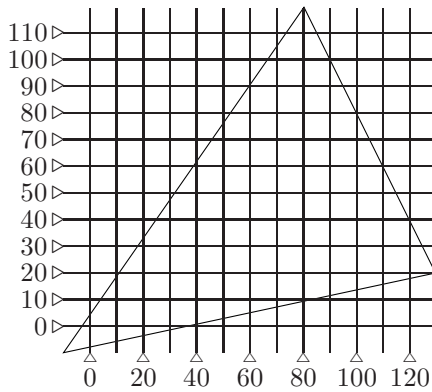


Figure 12

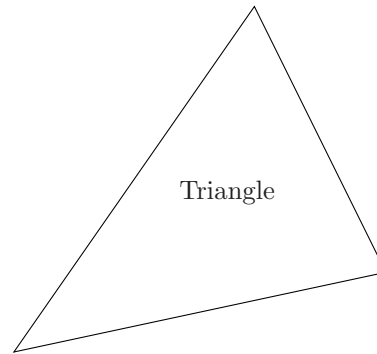


Figure 13

into the  $\text{\TeX}$  document to define an attach point as shown above. Note however that this feature is not available with every `ghostview` implementation.

A second solution is to use the macro `\figscan FileName(HX,HY)` which draws a rectangular grid with horizontal steps `HX` and vertical steps `HY`. Numerical values corresponding to the size of the figure described in the file `FileName` are printed, at most every 20 bp horizontally and every 10 bp vertically to avoid overprinting. The values of the steps are given in postscript units (`bp`) and supposed to be integers, because the first value is rounded and no decimals are printed. This macro is intended to be used temporarily together with the macro `\figinsert` to set the coordinates of the attach point or adjust the bounding box of the figure.

For example, if we want to write the word “Triangle” inside the triangle, we can do the following. First, determine the coordinates of the attach point. We compile a file containing the commands:

```
\def\MyPsfile{Fig1.ps}
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPsfile}
\figscan \MyPsfile (10,10)}
\centerline{\box\demo}
```

Then we use a  $\text{\TeX}$  previewer to look at the result shown on the figure 12.

If the  $\text{\TeX}$  previewer does not show the postscript files, it is necessary to create the postscript version and use a postscript previewer such as `ghostview` or `gs`. As a last resort, it is always possible to print it on a paper sheet.

The grid and the drawing are printed together. We can observe that horizontally, only one value over two is printed. We are now able to fix the coordinates of the point: we choose `(70,50)`. Then, we set the unit to `bp`, we remove the call to `\figscan`, we create the point and center the text on it, which leads to the following commands. The result is shown on the figure 13.

```
\figinit{bp}
\def\MyPsfile{Fig1.ps}
\figvisu{\demotwo}{\bf Figure \the\Figno}{
\figinsert{\MyPsfile}
\figpt 1:(70,50)\figwritec[1]{Triangle}}
```

Another useful feature is to control the scale of the figure to be inserted in the page. This can be done with the second argument of the macro `\figinsert` whose full prototype is:

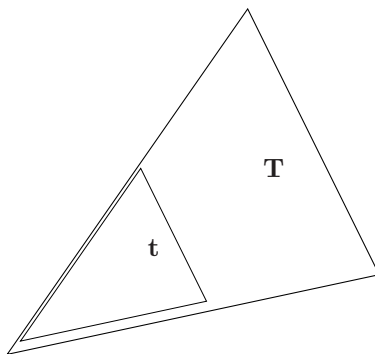
```
\figinsert{FileName, ScaleFactor}
```

The user has to provide the extra argument `ScaleFactor` which is a positive real number whose default value is 1, so that `\figinsert{FileName,1}` is equivalent to `\figinsert{FileName}`.

Remarks:

- Although this macro works with postscript files created by the `fig4tex` macro-package, it is essentially intended to be used with files created by other softwares. We recall that `fig4tex` controls the scale with the macro `\figinit`.
- When several scale factors are specified, the macro `\figscan` works correctly only if it appears *immediately after* the corresponding call to `\figinsert`.

An example of its use is given on figure 14.



**Figure 14**

The same postscript file, `Fig1.ps`, is used twice. We obtain two nested triangles and we observe that they are drawn with respect to the same origin. We choose 0.5 as scale factor and we write the letter **T** on the big triangle, the letter **t** at the same relative position on the small one.

The program that produces it is :

```
\figinit{bp}
\def\MyPsfile{Fig1.ps}
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPsfile}
\figt 1:(90,60)\figwritec[1]{\bf T}
\figinsert{\MyPsfile, 0.5}
\figt 0:(0,0)\figpthom 1:=1/0,0.5/\figwritec[1]{\bf t}
}
\centerline{\box\demo}
```

## §7. Setting graphical attributes

### 1. Introduction

Three macros are involved to set the graphical attributes. The first two ones have both the same prototype and at least one space is necessary before the keyword argument:

```
\psset      keyword (attribute1=value1, attribute2=value2,...)
\pssetdefault keyword (attribute1=value1, attribute2=value2,...)
\psreset{keyword1, keyword2,...}
```

Every attribute has a *current* value which is initialized to its corresponding *default* value as explained below. The macro `\psset` can temporarily change it to another current value, which is used until next change. For example, one can change the line width by saying: `\psset(width=1)`.

To get back to the default value, one can use `\psset` (called with the name of the default value) or `\psreset` which resets to their default value all the attributes related to a particular keyword, for example `\psset(width=\defaultwidth)` or `\psreset{first}`.

Moreover, the macro `\pssetdefault` allows the user to change the default values. Indeed, the choice made by the authors may not fit the needs of the user and this macro provides a way to customize the macro-package without having to modify the macro file. This macro is intended to be called just after the macro-package has been imported. The default values are changed up to the end of the document or next call to `\pssetdefault`, and are taken into account by every macro that refers to the default values, namely `\figinit`, `\psbeginfig` and `\psreset`. For example, the following lines:

```
\input fig4tex.tex
\pssetdefault(update = yes, width=1)
\pssetdefault arrowhead(angle=10)
```

now impose that, by default, the postscript files are updated, the line width is 1 bp and the arrow-head half-angle is 10 degrees.

The keyword argument refers to attributes that are logically grouped together. The list of available keywords is the following:

- `arrowhead` which deals with the arrow-head attributes,
- `curve` which deals with the attributes of a curve drawn by `\pscurve`,
- `first` which deals with the first-level (or primary) attributes,
- `flowchart` which deals with the flow chart attributes,
- `mesh` which deals with the attributes of a mesh drawn by `\psmesh`,
- `second` which deals with the second-level (or secondary) attributes,
- `third` which deals with the third-level (or ternary) attributes.

Then, here follow the lists of the attributes relative to each keyword. Each attribute is given along with the possible values it can take and the name of its default value in parentheses:

- `no` keyword or `keyword = first`
  - . `color` = color definition (`\defaultcolor`)
  - . `dash` = index or dash pattern (`\defaultdash`)
  - . `fillmode` = `yes/no` (`\defaultfill`)
  - . `join` = `miter (V)`, `round (U)` or `bevel (\_)` (`\defaultjoin`)
  - . `update` = `yes/no` (`\defaultupdate`)
  - . `width` = dimension in PostScript units (`\defaultwidth`)
- `keyword = arrowhead`
  - . `angle` = real in degrees (`\defaultarrowheadangle`)
  - . `fillmode` = `yes/no` (`\defaultarrowheadfill`)
  - . `length` = real in user coordinates (`\defaultarrowheadlength`)
  - . `out` = `yes/no` (`\defaultarrowheadout`)
  - . `ratio` = real in  $[0,1]$  (`\defaultarrowheadratio`)

- keyword = `curve`
  - . `roundness` = real usually in  $[0,0.5]$  (`\defaultroundness`)
- keyword = `flowchart`
  - . `arrowposition` = real in  $[0,1]$  (`\defaultfcarrowposition`)
  - . `arrowrefpt` = `start/end` (`\defaultfcarrowrefpt`)
  - . `line` = `polygon/curve` (`\defaultfcline`)
  - . `padding` = real in user coordinates (see `xpadding` and `ypadding`)
  - . `radius` = positive real in user coordinates (`\defaultfcradius`)
  - . `shape` = `rectangle`, `ellipse` or `lozenge` (`\defaultfcshape`)
  - . `thickness` = real in user coordinates (`\defaultfcthickness`)
  - . `xpadding` = real in user coordinates (`\defaultfcxpadding`)
  - . `ypadding` = real in user coordinates (`\defaultfcypadding`)
- keyword = `mesh`
  - . `diag` = integer in  $\{-1,0,1\}$  (`\defaultmeshdiag`)
- keyword = `second`
  - . `color` = color definition (`\defaultsecondcolor`)
  - . `dash` = index or dash pattern (`\defaultseconddash`)
  - . `width` = dimension in PostScript units (`\defaultsecondwidth`)
- keyword = `third`
  - . `color` = color definition (`\defaultthirdcolor`)

Remark: The current values of the attributes can be obtained at compilation time with the help of the macro `\figshowsettings` (see sections **Helpers** and **3D examples**). In this documentation file, the macro `\pssetdefault` is not called, so the default values used for the attributes in all the examples are the “factory” default values.

Except the `update` one, the attributes related to the keywords `first`, `second` and `third` have an action on the way a line is drawn. They are described in the next subsections. However, the `fillmode` attribute is presented later in the **Filled area** subsection, after some graphical macros have been introduced, in order to show its effect in a more demonstrative way. The `update` attribute has been described in the **Principle** subsection.

The secondary and ternary attributes are subsets of the primary ones. They are used by some high level macros and are mentioned when needed.

The other keywords correspond to particular graphical objects ; the associated macros and attributes are described in further sections devoted to those particular cases.

## 2. Line attributes

### • Line width

By default, the line width is set to 0.4 bp. The macro `\psset(width=NewWidth)` modifies the line width by using the new value `NewWidth`. The unit used here is always the postscript point and the argument must be given without any unit specification. For example, you get

—————	with <code>\psset(width=\defaultwidth)</code>
=====	with <code>\psset(width=1)</code>
—————	with <code>\psset(width=1.5)</code>
—————	with <code>\psset(width=2)</code>







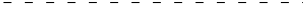


Remark: If the filling mode is on (see **Filled area** subsection), the width of a straight line cannot be changed ; think to check this in case of trouble.

### • Line style




The default line style is solid. The macro `\psset(dash=...)` modifies the line style. Two calling sequences are available:

```
\psset(dash=Index)
\psset(dash=Pattern)
```

The first form refers to the value *Index* which is an integer taking its value between 1 to 10. The solid line corresponds to the value 1, which can be set by saying `\psset{dash=\defaultdash}`. The different line styles available are shown below, using the default line width.

	with <code>\psset{dash=\defaultdash}</code>
	with <code>\psset{dash=2}</code>
	with <code>\psset{dash=3}</code>
	with <code>\psset{dash=4}</code>
	with <code>\psset{dash=5}</code>
	with <code>\psset{dash=6}</code>
	with <code>\psset{dash=7}</code>
	with <code>\psset{dash=8}</code>
	with <code>\psset{dash=9}</code>
	with <code>\psset{dash=10}</code>

The second form refers to a pattern which is a sequence of integers separated with white spaces. The general form is `Dash1 Space1 ... Dashn Spacen Offset` where each value is a dimension given in postscript points (bp), *Dash<sub>i</sub>* corresponding to the length of a dash, *Space<sub>i</sub>* to the length between two dashes. The last value *Offset* is optional. It corresponds to the length by which the whole pattern is shifted towards the beginning of the line. Its default value is 0. It allows to adjust the position of the pattern. For example, on the previous figure, we may want to “center” the pattern over the second segment. Since `\psset{dash=2}` is equivalent to `\psset{dash=6 2}`, we obtain a better look using the pattern form of the macro as shown below:

	with <code>\psset{dash=2}</code>
	with <code>\psset{dash=6 2}</code> (same as previous)
	with <code>\psset{dash=6 2 2}</code>

• Line join

This attribute establishes the shape to be put at the corners of a polygonal line. Its effect is particularly visible with wide lines or when several lines meet at a point. In this case, best rendering is often obtained with `join=round`. The three possible values are shown on the following figure. The default value is `miter`.



### 3. Using color

Highlighting some part of a figure can be achieved by changing the line style, but also, often more efficiently, by adding color. This can be done while creating the figure with the macro `\psset{color=Color Definition}`. Once a color is set, every subsequent graphic will be drawn in this color, until another color setting occurs. This apply to lines as well as to filled areas. The default color is black.

The argument *Color Definition* of this macro can be a *gray* shade or a color definition according to the *cmymk* or the *rgb* color model. The gray shade intensity is controlled by a real number taking its value between 0 and 1, with 0 corresponding to black, 1 corresponding to white and intermediate values corresponding to intermediate shades of gray. To specify a color, one can give:

- either numerical values (or coordinates) corresponding to the color definition in the color model (note there are 4 coordinates for *cmymk* and 3 for *rgb*),
- or a color name, which is a macro containing the color definition.

For example, one obtains the same result by saying `\psset(color=\Redrgb)` or `\psset(color=1 0 0)`, provided the macro `\Redrgb` has been defined. Note that the coordinates are separated with spaces and range between 0 and 1. The basic colors have been predefined in each color model, along with some other ones :

- in *cmymk* : `\Blackcmyk, \Whitecmyk, \Cyanmyk, \Magentacmyk, \Yellowcmyk, \Redcmyk, \Greencmyk, \Bluecmyk, \Graycmyk, \BrickRedcmyk, \Brownmyk, \ForestGreencmyk, \Goldenrodmyk, \Marooncmyk, \Orangecmyk, \Purplecmyk, \RoyalBluecmyk, \Violetcmyk,`
- in *rgb* : `\Blackrgb, \Whitergb, \Redrgb, \Greenrgb, \Bluergb, \Cyanrgb, \Magentargb, \Yellowrgb, \Grayrgb, \Chocolatergb, \DarkGoldenrodrgb, \DarkOrangergb, \Firebrickrgb, \ForestGreenrgb, \Goldrgb, \HotPinkrgb, \Maroonrgb, \Pinkrgb, \RoyalBluergb.`

Remark: The color coordinates are taken from different sources and the associated names have been preserved. So a same color may render differently in the two color models: this is the case for `ForestGreen` for example.

It is easy to enrich this small list. To define a color name, one has to associate its coordinates with the name of a macro. For example, orange is defined in *cmymk* model by saying `\def\Orangecmyk{0 0.61 0.87 0}` .

Finally, as already mentionned in the previous **Introduction** subsection, some high level macros use secondary or ternary settings, including color. For example, setting a secondary color in *gray* tone, *cmymk* or *rgb* color code can be achieved by saying `\psset second(color=Color Definition)` which works the same way as `\psset(color=Color Definition)`.

Remark: In previous versions of this macro package, the specification of the color was made using several macros: `\pssetgray`, `\pssetcmyk`, `\pssetrgb`, `\pssetsecondgray`, `\pssetsecondcmyk`, `\pssetsecondrgb`. They are still available, although they are becoming useless since the new macro `\psset` is more practical.

## §8. Drawing macros

### 1. Preliminary remarks

We now present some macros we think useful in a variety of situations listed in the following sections. However, when we want to design a new figure, the available macros may not solve directly the problem and it may be necessary to write a new macro. In this case, we advise you to build it, as far as possible, using user macros. But sometimes, this will imply to use internal macros: the best is to start from the structure of an existing macro that looks similar to the new one.

Every system has limits of use and we must be aware of what can be expected from this macro-package. We recall that its main purpose is to give the ability to write some text on a figure, easily and under total control of the user. Whenever complicated computations are necessary, they must be done elsewhere and used here, or the figure must be entirely created by another software and then imported ; the macro `\figscan` has been created for this purpose.

### 2. Arc

We already saw the macro `\pscirc` which draws an entire circle. Here are macros that deal with a portion of arc. There are several versions of them so that one can choose the simplest solution depending on the available data.

If data consists mainly in angles, one can use the macros whose prototypes are:

```
\psarccirc Center ; Radius (Ang1,Ang2)  
\psarcell Center ; XRad,YRad (Ang1,Ang2, Inclination)
```

The first one draws a circular arc, the second one draws an arc of ellipse. Although a circle is a particular case of ellipse, the macro `\psarccirc` has been built because a circular arc is needed more often than an arc of ellipse, so a shorter list of arguments makes its use more straightforward.

Let us detail the arguments of the macros. `Ang1`, `Ang2` and `Inclination` are angles to be given in degrees. `Ang1` and `Ang2` are measured counterclockwise from the local  $X$  axis, also called major axis.

The circular arc is the part of the circle of center `Center` and radius `Radius` delimited by the angles `Ang1` and `Ang2`.

Except the limiting angles, the arguments of `\psarcell` have the same definitions as those of the macro `\figptell`. The arc of ellipse is the part of the ellipse defined by the center `Center`, the two radii `XRad` and `YRad`, delimited by the parametrization angles `Ang1` and `Ang2`. The ellipse is rotated by `Inclination` on the paper sheet.

If data consists mainly in points, these macros are not very handy. This is the case for example when the arc we would like to draw is limited by two crossing lines. We know the intersecting point (that can be computed by `\figptinterlines`) and one other point on each line. So “point versions” have been created whose prototypes are:

```
\psarccircP Center ; Radius [Pt1,Pt2]  
\psarcellPP Center,PtAxis1,PtAxis2 [Pt1,Pt2]
```

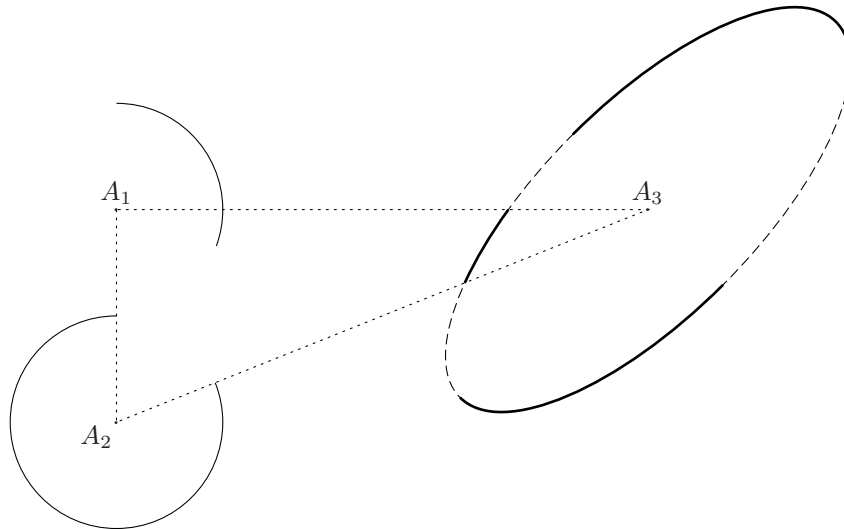
The first macro draws a circular arc of center `Center` and of radius `Radius` limited by the two half-lines  $(Center, Pt1)$  and  $(Center, Pt2)$ . The arc is drawn from `Pt1` towards `Pt2` turning counterclockwise around `Center`.

Let  $C, A_1, A_2$  stand for `Center`, `PtAxis1` and `PtAxis2`. These arguments have the same definition as those of the macro `\figptellP`:  $C$  is the center of the ellipse,  $A_1$  is the end point of the major axis and  $A_2$  is the end point of the minor axis. The macro `\psarcellPP` draws an arc of ellipse of center  $C$  limited by the two half-lines  $(Center, Pt1)$  and  $(Center, Pt2)$ . The arc is drawn counterclockwise around the vector  $\vec{CA_1} \times \vec{CA_2}$ .

Another version of the latter one exists. Its prototype is:

```
\psarcellPA Center,PtAxis1,PtAxis2 (Ang1, Ang2)
```

It behaves exactly like `\psarcellPP` except that it allows to specify the portion of the arc using the parametrization angles `Ang1` and `Ang2`.



**Figure 19**

Figure 19 shows an example of their use. The text of the program that produces it follows. Notice that the points #4 and #5 are created only to have the opportunity to call the “point” versions of `\psarccll`.

```
% 1. Definition of characteristic points
\figinit{pt}
\def\haxis{100}\def\vaxis{40}
\def\startangle{-20}\def\stopangle{90}\def\inclin{45}
\figt 1:(-10,10)\figttraC 2:=1/0,-80/\figttraC 3:=1/200,0/
\figtcell 4:: 3 ; \haxis,\vaxis( 0, \inclin) % End point of major axis
\figtcell 5:: 3 ; \haxis,\vaxis(90, \inclin) % End point of minor axis
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psarccirc 1;\vaxis(\startangle,\stopangle)\psarccircP 2;\vaxis[1,3]
\psset(dash=3)\psarccll 3 ; \haxis,\vaxis(0,360, \inclin)
\psset(dash=\defaultdash,width=1)
\psarccll 3 ; \haxis,\vaxis(\startangle,\stopangle, \inclin)
\psarccllPP 3,4,5[1,2]\psarccllPA 3,4,5(-180,-90)
\psset(width=\defaultwidth,dash=5)\psline[1,2,3,1]
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPSfile}
\figsetmark{.}\figwriten 1,3:(2)\figwritesw 2:(2)
}
\centerline{\box\demo}
```

### 3. Example

Here is a more elaborated example involving a small geometric construction. The data consist of an ellipse of center  $C$  and a point  $O$  lying on its major axis. The aim is to draw the tangent lines to the ellipse passing through  $O$ .

Thus, we have to compute the two points  $I$  and  $J$  lying on the ellipse such that  $(O, I)$  and  $(O, J)$  are tangent to the ellipse.

The two radii of the ellipse are denoted  $R_x$  and  $R_y$ , and are such that  $R_x > R_y$ . Let  $A$  be the middle point of  $[OC]$  and  $\rho = \|\vec{AC}\|$ . The two circles  $(A; \rho)$  and  $(C; R_x)$  intersect at points  $I'$  and  $J'$ . The lines



$(O, I')$  and  $(O, J')$  are the tangent lines to the circle  $(C; R_x)$  passing through  $O$ . Moreover, if  $B$  denotes a point lying on  $(O, C)$  with  $x_B > x_C$ , the angles  $\theta_I = (\overrightarrow{CB}, \overrightarrow{CI'})$  and  $\theta_J = (\overrightarrow{CB}, \overrightarrow{CJ'})$  are the parametrization angles of the points  $I$  and  $J$  which solves the problem.

The corresponding figure is given below, along with the program that produces it. Some additional points are computed in order to draw the lines.

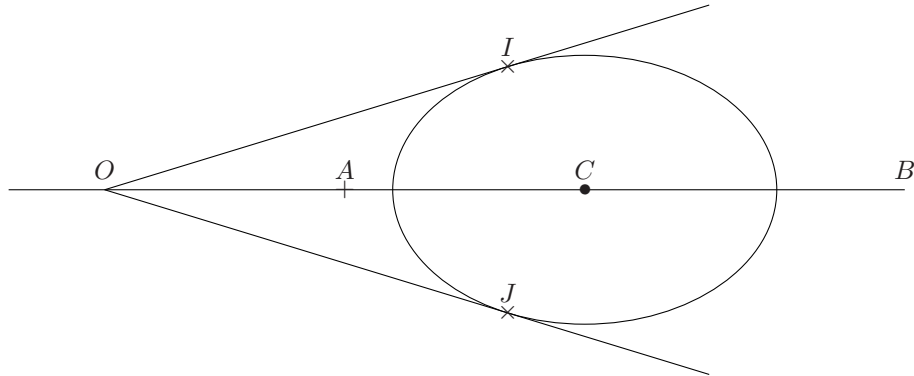


Figure 20

```
% 1. Definition of characteristic points
\figinit{in}
% Data
\figpt 0:$O$(-0.5,0)\figpt 1:$C$(2,0)
\def\Rx{1}\def\Ry{0.7}
% Computations
\figptbary2:$A$[0,1;1,1]\figgetdist\RHO[2,1]
\figptsintercirc 3[1,\Rx;2,\RHO] % Computes points 3 (I') and 4 (J')
\figptbary 10:[0,1;6,-1]\figptbaryR 11:$B$[0,1;-1,2.5]
\figgetangle\ThetaI[1,11,3]
\figgetangle\ThetaJ[1,11,4]
\figptell 3:$I$: 1;\Rx,\Ry (\ThetaI,0)
\figptell 4:$J$: 1;\Rx,\Ry (\ThetaJ,0)
\figptbary 13:[0,3;-1,3]
\figptbary 14:[0,4;-1,3]
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psarcell 1;\Rx,\Ry (0,360,0)
\psline[10,11]\psline[0,13]\psline[0,14]
\psendfig
% 3. Writing text on the figure
\def\dist{4pt}
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPSfile}
\figwriten 0:(\dist)\figwriten 11:(\dist)
\figsetmark{+}\figwriten 2:(\dist)
\figsetmark{\figBullet}\figwriten 1:(\dist)
\figsetmark{\figtimes}\figwriten 3,4:(\dist)
}
\centerline{\box\demo}
```

#### 4. Curve

From the postscript capabilities, we derived a basic macro to draw a portion of curve. Its prototype is

```
\psBezier N [Pt_1, ..., Pt_{3N+1}]
```

It allows to draw polynomial curves of any shape using the Bézier representation, that is to say by giving a set of control points. Since the postscript language only provides cubic polynomials, this macro draws a curve formed by a succession of cubic arcs, each of them defined by four control points. The fourth control point of an arc is the first of the following arc. Thus, if the curve is composed of  $N$  arcs, the user must provide  $3N + 1$  control points. This is not checked.

As a consequence of the Bézier representation, the points  $P_1, P_4, \dots, P_{3k+1}, \dots, P_{3N+1}$  lie on the curve. Moreover, the segment  $[P_{3k}, P_{3k+1}]$  (respectively  $[P_{3k+1}, P_{3k+2}]$ ) is tangent to the arc number  $k$  (respectively  $k + 1$ ) at point  $P_{3k+1}$ .

Remarks:

- Two successive points may coincide.
- We recall that the coordinates of a point lying on a cubic Bézier arc can be computed by the macro `\figptBezier` and the derivatives by the macro `\figvectDBezier`.

Here follows an example showing a curve consisting of three cubic arcs. The dashed polygon shows the control points. The curve is  $C^2$  except at point  $P_4$  where it is only  $C^0$  and at point  $P_7$  where it is  $C^1$  since  $P_8$  is the symmetric of  $P_6$  with respect to  $P_7$ .

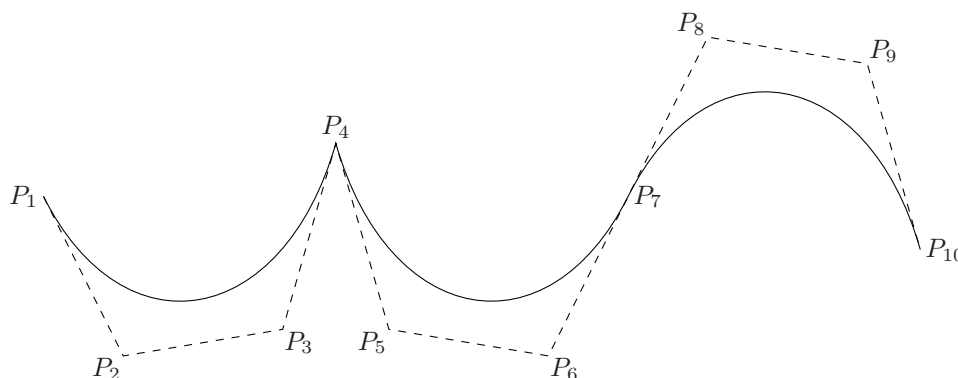


Figure 21

The text of the program that produces it is:

```
% 1. Definition of characteristic points
\figinit{pt}
\figpt 1:(-30,-10)\figpt 2:(0,-70)\figpt 3:(60,-60)\figpt 4:(80,10)
\figpt 0:(80,100) % Only used to define the line (0,4) for the symmetry
\figptssym 5=3,2,1/0,4/\figptsrot 8=6,5,4/7,180/
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psBezier 3[1,2,3,4,5,6,7,8,9,10]
\psset(dash=8)\psline[1,2,3,4,5,6,7,8,9,10]
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPSfile}\figsetptname{\$P_{\#1}\$}
\figwritew 1:(2)\figwritesw 2,5:(1)\figwritese 3,6:(1)\figwriten 4:(2)
\figwritee 7,10:(2)\figwritenw 8:(1)\figwritene 9:(1)
}
\centerline{\box\demo}
```

It turns out that the macro `\psBezier` is not very handy when one wants to draw a smooth curve. So, a new macro has been designed for that purpose whose prototype is

`\pscurve [Pt0,Pt1,... ,PtN,PtN+1]`

It draws a  $C^1$  curve that *interpolates* the points  $P_1, P_2, \dots, P_N$ . The curve consists of  $N - 1$  Bézier cubic arcs. The direction of the tangent at  $P_i$  is given by the vector  $\overrightarrow{P_{i-1}P_{i+1}}$ ,  $i = 1, \dots, N$ . To get a  $C^1$  closed curve, just let the last three points be the same as the first three ones.

The shape of the curve can be modified by a “roundness” parameter whose value is set by the macro `\psset curve(roundness=value)` where *value* is a real number to be chosen between 0.15 and 0.3 in order to obtain the best results. Its default value is 0.2, which can be set by saying `\psreset{curve}` or `\psset curve(roundness=\defaultroundness)`. The aim is to influence the roundness of the curve, and even its smoothness, since setting this value to 0 produces the polygonal line defined by the given points. Other values greater than 0.5 (or negative values) lead to (generally unwanted) strange shapes.

Remarks:

- Wittingly, no check is performed on the “roundness” parameter, so a large value may produce an arithmetic error.
- The macro `\pscurve` must obviously be called with at least 4 points. This is not checked.
- Two successive points may coincide: this feature may probably only be useful at the end points of the line by setting, for example,  $P_0 = P_1$  so that the line starts at  $P_1$  with the tangent  $\overrightarrow{P_1P_2}$ .
- The macro `\figptscontrolcurve` makes available to the user the control points used internally by `\pscurve` to draw the smooth curve.

The two following figures show what can be done with this macro.

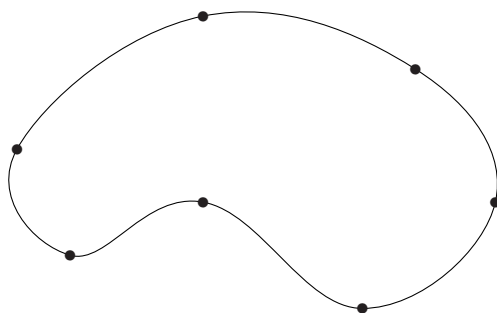


Figure 22

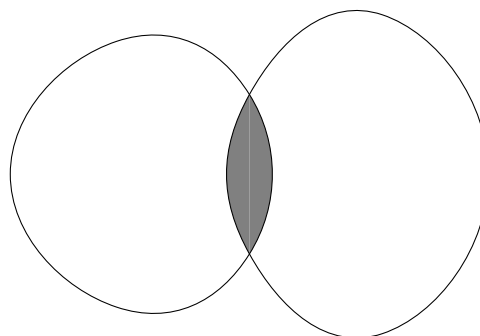


Figure 23

The text of the program that produces the left-hand side figure is:

```
% Left-hand side figure
% 1. Definition of characteristic points (Interpolation points)
\figinit{pt}
\figpt 1:(-100, 20)\figpt 2:(-30, 70)
\figpt 3:(50, 50)\figpt 4:(80,0)
\figpt 5:(30, -40)\figpt 6:(-30, 0)\figpt 7:(-80, -20)
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\pscurve[1,2,3,4,5,6,7,1,2,3]
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPSfile}
\figsetmark{\figBullet$}
\figwritep[1,2,3,4,5,6,7]
}
```

The text of the program that produces the right-hand side figure is given below. Here, we are anticipating on some features described in the **Filled area** section.

```
% Right-hand side figure
% 1. Definition of characteristic points (Interpolation points)
\figinit{pt}
\figpt 1:(0, 30)\figpt 2:(0, -30)
\figpt 10:(-50, 50)\figpt 11:(-50,-50)\figpt 12:(-90,0)
\figpt 20:(50, 60)\figpt 21:(50,-60)\figpt 22:(90,0)
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psset curve(roundness=0.23) % More roundness
\psset(fillmode=yes,color=0.5)
\pscurve[11,2,1,10]\pscurve[20,1,2,21] % Better to do this first
\psset(fillmode=no,color=0)
\pscurve[11,2,1,10,12,11,2,1]\pscurve[20,1,2,21,22,20,1,2]
\psendfig
% 3. Writing text on the figure
%\newbox\demotwo
\figvisu{\demotwo}{\bf Figure \the\Figno}{
\figinsert{\MyPSfile}
}
```

## 5. Filled area

Except arrow drawing macros, every drawing macro dealing with lines, arcs and curves presented so far can fill the area delimited by the line, the arc or the curve with colored ink, according to the current setting made with `\psset{color=Color Definition}`. The filling algorithm is governed by postscript own rules. In particular, for circular or elliptic arcs, the area is limited by the arc and the corresponding chord.

The macro `\psset{fillmode=switch}` allows to activate the filling algorithm if *switch* equals `yes` or not if *switch* equals `no`. The default is `no`.

To show how it works, we start from the two figures 19 and 21 and we replace the postscript file definition by

```
\psbeginfig{\MyPSfile}
\psset{fillmode=yes}
\psarccirc 1;\vaxis(\startangle,\stopangle) % Use default color
\psset{color=\Redrgb}\psarcell 3 ; \haxis,\vaxis(\startangle,\stopangle, \inclin)
\psset{color=\Bluergb}\psarcellPA 3,4,5(-180,-90)
\psendfig
```

in the first one and by

```
\psbeginfig{\MyPSfile}
\psset{fillmode=yes,color=0.7}
\psBezier 3[1,2,3,4,5,6,7,8,9,10]
\psendfig
```

in the second one. We then get the two new figures:

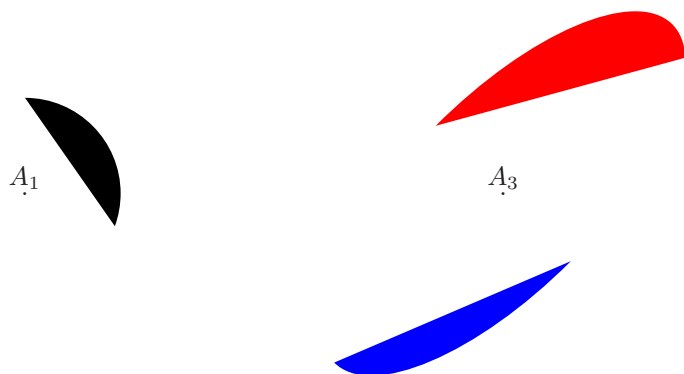


Figure 24



Figure 25

Note that to shorten the presentation some modifications have been made: 1) the dashed lines have been removed, 2) the point  $A_2$  has been removed along with one of the arcs on the ellipse, 3) the dimensions of the figures have been shrunk to 90% and 50% respectively.

To put the two figures on the same line, we just created a new box `\demotwo` for the curve, gave it as first argument to `\figvisu`, and made the classical assembly:

```
\centerline{\box\demo\hfil\box\demotwo}
```

Remarks:

- When the filling mode is active, the macro `\psline` gives the same result if the path is closed or not.
- Filling an area with white ink can be used to erase some existing drawing and then write some text at this “cleaned” place. Note that this feature may not be handled properly by some screen previewers although the result on the paper is correct.

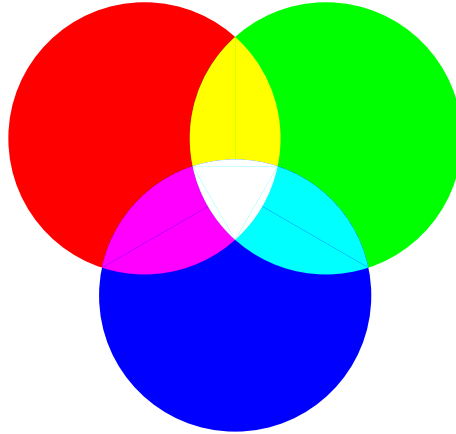


Figure 26

The following example is a reproduction of the well known RGB color disks shown on figure 26. The data are the centers  $(A, B, C)$  of the three disks and the radius. The points  $(A, B, C)$  are the vertices of an equilateral triangle,  $A$  and  $B$  lying on the X-axis. In order to fill the different areas, we need to compute their limits given by the intersections of two circles, which is required by the macro `\psarccircP`.

The program that produces the figure 26 is given below. Note that the intersections of two disks are filled with two symmetric areas. It may also be useful to specify that since postscript images are opaque, the order in which the different areas are painted is essential to get the expected result.

```
% 1. Definition of characteristic points
\figinit{0.6cm}
\def\xB{2} % Abscissa of B
\def\R{3} % Radius of the 3 circles,  $r < R < r*\sqrt{3}$ ,
%          r radius of the circumscribed circle ( $r=2*\xB/\sqrt{3}=2.3094$ )
\figpt 1:A(-\xB, 0) \figpt 2:B(\xB, 0) \figptrot 3:C=2/1,-60/
\figptsintercirc 4[1,\R;2,\R] \figptsintercirc 14[3,\R;1,\R]
\figptsintercirc 24[2,\R;3,\R]
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psset(fillmode=yes)
\psset(color=\Redrgb) \pscirc 1(\R)
\psset(color=\Greenrgb)\pscirc 2(\R)
\psset(color=\Bluergb) \pscirc 3(\R)
\psset(color=\Yellowrgb) \psarccircP1;\R[4,5] \psarccircP2;\R[5,4]
\psset(color=\Magentargb)\psarccircP1;\R[15,14]\psarccircP3;\R[14,15]
\psset(color=\Cyanrgb) \psarccircP3;\R[25,24]\psarccircP2;\R[24,25]
\psset(color=\Whitergb)
\psarccircP2;\R[24,4]\psarccircP1;\R[4,14]\psarccircP3;\R[14,24]
\psline[4,14,24]
\psendfig
% 3. Inserting the figure
\figvisu{\demo}{\bf Figure \the\Figno}{\figinsert{\MyPSfile}}
\centerline{\box\demo}
```

## 6. Arrow

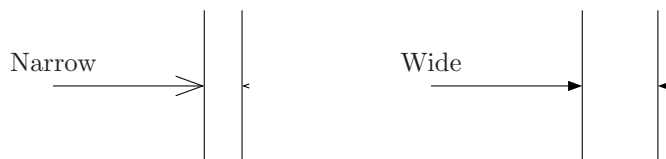


Figure 27

Corresponding program :

```
% 1. Definition of characteristic points
\figinit{cm}
\figpt 0:(-1,0)\figpt 1:(1,0)\figpt 2:(1,1)\figpt 3:(1,-1)
\figvectC 20(0.5,0)\def\VTrans{20}
\figptstra 4=1,2,3/1,\VTrans/
\figptcopy10:/0/
\def\LTrans{10}\figpttra 11:=0/\LTrans,\VTrans/
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psline[2,3]\psline[5,6]
\psset arrowhead(ratio=0.2)\psarrow [0,1]
\psset arrowhead(out=yes)\psarrowhead [1,4]
\figptstra 0=0,1,2,3,4,5,6/\LTrans,\VTrans/\figptstra 4=4,5,6/1,\VTrans/
\psline[2,3]\psline[5,6]
\psreset{arrowhead}
\psset arrowhead(fillmode=yes,length=0.2)\psarrow[0,1]
\psset arrowhead(out=yes)\psarrowhead[1,4]
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPSfile}
\figwriten 10:Narrow(0.2)\figwriten 11:Wide(0.2)
}
\centerline{\box\demo}
```

The two main macros are:

```
\psarrow [Pt1,Pt2]
\psarrowhead [Pt1,Pt2]
```

The first one draws a straight arrow from Pt1 to Pt2. The arrow-head is drawn according to its current attributes (see below). The second macro can be used whenever it is needed to draw an arrow-head alone (the body of the arrow is then not drawn).

The appearance of the arrow-head depends on four attributes: the opening angle, the length, the drawing mode and the position. Each of them can be modified by specific macros given below. To reset at once all the attributes to their default values, the best is just to say `\psreset{arrowhead}`.

- opening angle: `\psset arrowhead(angle=value)`  
sets the arrow-head half-angle to *value* (in degrees). The default value is 20 degrees, that can be set by saying `\psset arrowhead(angle=\defaultarrowheadangle)`.
- length: Precisely, we speak of the length of each of the two edges of the arrow-head. This attribute can be set by one of the two following macros which are mutually exclusive ; the default is to use the length.

```
\psset arrowhead(length=value)
```

sets the arrow-head length to *value* (in user coordinates). The default value is equivalent to a 8.0 pt length, that can be set by saying `\psset arrowhead(length=\defaultarrowheadlength)`.

`\psset arrowhead(ratio=value)`

sets the arrow-head ratio to *value*, real number usually in (0,1). The default value is 0.1, that can be set by saying `\psset arrowhead(ratio=\defaultarrowheadratio)`. The arrow-head length is then equal to *value* times the length of the body of the arrow.

- drawing mode: `\psset arrowhead(fillmode=switch)`

sets the arrow-head filling switch to *yes* or *no*. The default value is *no*. If *yes* is chosen, the effect is to fill the triangle with the current primary color.

- position: `\psset arrowhead(out=switch)`

sets the arrow-head "outside" switch to *yes* or *no*. The default value is *no*. If *yes* is chosen, the effect is to draw the arrow-head outside the segment [Pt1, Pt2].

Remark: As already said in section **Setting graphical attributes**, the default values of these attributes can be changed by the macro `\pssetdefault`. However, the length attribute is a particular case because when `\pssetdefault` is called, the unit is generally not yet defined. For this reason, in this case, the unit can be specified after the numerical value. If the unit is not specified, *pt* is assumed. Example: `\pssetdefault arrowhead(length=4mm)`

Figure 27 shows how to use these macros. The text of the program that produces it is given below it.

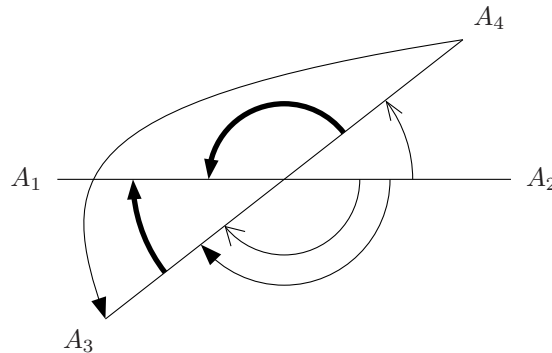


Figure 28

Corresponding program :

```
% 1. Definition of characteristic points
\figinit{cm}
\def\rotangle{38}
\figpt 1:(-3,0)\figpt 2:(3,0)\figpt 0:(0,0)
\figptsrot 3=1,2/0,\rotangle/\figpt 5:(-3,1)
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psline[1,2]\psline[3,4]
\psarrowcirc 0;1.7(0,\rotangle)\psset arrowhead(angle=30)
\psarrowcircP 0;-1[2,3]
\psset arrowhead(angle=\defaultarrowheadangle,fillmode=yes)\psset(width=2)
\psarrowcircP 0;1[4,1]\psarrowcircP 0;-2[3,1]
\psset(width=\defaultwidth)\psset arrowhead(length=0.3)
\psarrowcircP 0;-1.4[2,3]\psarrowBezier[4,5,1,3]
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPSfile}
\figwritew 1:(0.2)\figwritew 2:(0.2)\figwritesw 3:(0.2)\figwritene 4:(0.2)}
\centerline{\box\demo}
```



It is also very convenient to draw circular arrows. Two macros are available for that:

```
\psarrowcirc Center ; Radius (Ang1,Ang2)
\psarrowcircP Center ; Radius [Pt1,Pt2]
```

The macro `\psarrowcirc` draws a circular arrow such that the circular arc is centered at `Center`, has the radius `Radius` and is limited by the angles `Ang1` and `Ang2` given in degrees. If `Ang2 > Ang1`, the arrow is drawn counterclockwise, else it is drawn clockwise. The arrow-head is drawn according to the `\psarrowhead` macro settings.

The macro `\psarrowcircP` is the “point version” of the previous one. It draws a circular arrow such that the circular arc is centered at `Center`, has the radius `|Radius|` and is limited by the two half-lines `(Center, Pt1)` and `(Center, Pt2)`. By default, the arrow is drawn from `Pt1` towards `Pt2` turning counterclockwise around `Center`. Inserting a minus sign before the radius reverts the turning direction. In other words, the arrow is drawn counterclockwise if `Radius > 0`, clockwise if `Radius < 0`.

At last, a free form arrow can be drawn with the help of the macro

```
\psarrowBezier [Pt1,Pt2,Pt3,Pt4]
```

It draws the arrow that consists of the cubic Bézier curve defined by the four control points `Pt1`, `Pt2`, `Pt3` and `Pt4`, and the arrow-head at point `Pt4`.

Figure 28 shows how to use these macros. The text of the program that produces it is given below it.

## 7. Axes

In this paragraph, we present a macro designed to draw axes quickly. It is mainly for user convenience since it is a short-cut to avoid drawing arrows “manually”. The prototype of the macro has two forms:

```
\psaxes Origin(X1,X2, Y1,Y2)
\psaxes Origin(L)
```

It draws axes crossing at the point denoted `Origin`. Each axis is parallel to the corresponding absolute axis and is drawn as an oriented arrow between two values given in user coordinates: from `X1` to `X2` for the  $X$ -axis, from `Y1` to `Y2` for the  $Y$ -axis.

The second form of the macro is equivalent to `\psaxes Origin(0,L, 0,L)` and is intended to draw equal axes in a straightforward way.

Remark: The graphical attributes of the arrows can be modified by the macro `\psset` as explained in the previous section.

Since we may want to write a legend at the end points of the axes, we need to compute these points. This is the aim of the macro `\figptsaxes` whose prototype is nearly a copy of the one of `\psaxes` in order to simplify its use:

```
\figptsaxes NewPt1 : Origin(X1,X2, Y1,Y2)
\figptsaxes NewPt1 : Origin(L)
```

If `NewPt1` is equal to  $k$ , then this macro computes the end point of the  $X$ -axis bearing number  $k$  and the end point of the  $Y$ -axis bearing number  $k + 1$ . As for `\psaxes`, the second form of the prototype is equivalent to `\figptsaxes NewPt1 : Origin(0,L, 0,L)`.

Moreover, the text  $x$  and  $y$  is automatically joined to the points, so that writing the legend with the `\figwrite*` macros is more straightforward. For example, one can write something like:

```
\figwrites k:(3pt) \figwritew k+1:(3pt) .
```

However, as explained in the section **Writing text on the figure**, the text can be modified at this level.

The following figure shows how these macros can be used. It shows also some other macros related to curves. In this example, the curve interpolates four data points. It is made of three Bézier arcs, each of them being a parametric curve with the parameter value ranging in  $[0, 1]$ . We can notice that the update of the postscript file is enforced: this is because some points are computed during step 2 and are still needed during step 3. The text of the program that produces this figure is given below it.

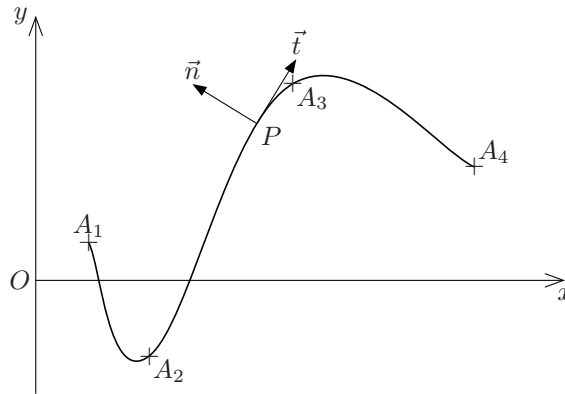


Figure 29 Frenet basis at point  $P$

Corresponding program :

```
% 1. Definition of characteristic points
\figinit{cm}
\def\ORIG{0} \figpt \ORIG:$0$(0, 0)
\figpt 1:(0.7,0.5)\figpt 2:(1.5,-1)
\figpt 3:(3.4,2.6)\figpt 4:(5.8,1.5)
\def\Xone{0}\def\Xtwo{7}\def\Yone{-1.5}\def\Ytwo{3.5}
% 2. Creation of the postscript file
\psset (update=yes)
\psbeginfig{\MyPSfile}
% Draw the axes and the curve that interpolates the data points
\psaxes \ORIG(\Xone,\Xtwo, \Yone,\Ytwo)
\psset (width=0.6)\pscurve [1,1,2,3,4,4]\psset (width=\defaultwidth)
% Compute the control points of the smooth curve
\figptscontrolcurve 11, \NbArcs [1,1,2,3,4,4]
% Compute a point lying on the second arc of the curve (between A2 and A3)
\def\PointName{$P$}
\def\Valt{0.8}\figptBezier 25 :\PointName: \Valt [14,15,16,17]
% Compute and normalize the tangent vector at this point
\figvectDBezier 21 : 1, \Valt [14,15,16,17]\figvectU 21[21]
% Draw the tangent and normal vectors
\psset arrowhead(length=0.2,fillmode=yes)
\figpttra 26:$\vec{t}$=25/1,21/\psarrow [25,26]
\figvectNV 22[21]\figpttra 27:$\vec{n}$=25/1,22/\psarrow [25,27]
\psendfig
% 3. Writing text on the figure
\figvisu{demo}{\bf Figure \the\Figno}\Frenet basis at point \PointName}{
\figinsert{\MyPSfile}
% Write the name of the points
\figsetmark{${}$}\figwriten 1:(2pt)\figwritese 2,3:(2pt)\figwritene 4:(2pt)
\figsetmark{}\figwritew \ORIG:(2pt)\figwritese 25:(2pt)
% Write the name of the 2 vectors
\figwriten 26,27:(2pt)
% Compute the end points of the axes and write the associated text
\figptsaxes 1:\ORIG(\Xone,\Xtwo, \Yone,\Ytwo)
\figwrites 1:(3pt) \figwritew 2:(3pt)
}
\centerline{\box\demo}
```

## 8. Triangle related macros

We now present a set of macros related to the geometry of the triangle.

The macro, whose prototype is

```
\psaltitude Dim [Pt1,Pt2,Pt3]
```

builds the altitude drawn from  $Pt1$  in the triangle  $(Pt1, Pt2, Pt3)$ .  $Dim$  is the dimension of the square at the foot  $H$  of the altitude on the segment  $[Pt2, Pt3]$ . The square is drawn on either side of this point according to the order of the two points  $Pt2$  and  $Pt3$ . If the point  $H$  is outside the segment  $[Pt2, Pt3]$ , a line is drawn to support the square. The attributes of this line are controlled by the secondary settings described in the **Setting graphical attributes** section: the line width, the line style and its color can be modified with the macro `\psset` with `second` as keyword and respectively `width`, `dash` and `color` as attributes. On Figure 30, we can see the two altitudes drawn from  $P_2$  and  $P_3$ , with respective end points  $H_2$  and  $H_3$ .

The macro, whose prototype is

```
\psnormal Length,Lambda [Pt1,Pt2]
```

draws the so-called exterior normal, i.e. the vector of length `Length` which is normal to the segment  $[Pt1, Pt2] = [P_1, P_2]$ . The direction of the vector is defined by saying that if one goes along the segment from  $P_1$  towards  $P_2$ , then the normal vector is directed towards the righthand side. The word “exterior” implicitly means exterior to a domain: if the segment is part of the boundary of a polygonal domain, the domain is on the left when one goes along the boundary from  $P_1$  towards  $P_2$ . The origin of the vector is the barycenter of  $\{(P_1; \lambda), (P_2; 1 - \lambda)\}$  where  $\lambda$  is a real number whose value is given by `Lambda`. Since this macro draws an arrow, we have to use the attributes presented in the section **Arrow** if we want to modify the appearance of the arrow-head.

Generally used in connection with `\psnormal`, the macro

```
\figptendnormal NewPt :Text: Length,Lambda [Pt1,Pt2]
```

computes the end point `NewPt` of the “exterior normal” to the segment  $[Pt1, Pt2]$ . The last four arguments have the same meaning as those of the macro `\psnormal`. This `NewPt` can then be used as an attach point of a text to be written on the figure. On Figure 30, we can see the normal vector  $\vec{n}_2$  to the segment  $[P_1, P_3]$ .

The text of the program that produces the figure 30 is:

```
% 1. Definition of characteristic points
\figinit{0.9pt}
\figpt 1:(-30,0)
\figpt 2:(170,60)
\figpt 3:(60,-40)
\figptorthoprojline 12:$H_3$=3/1,2/
\figptorthoprojline 13:$H_2$=2/1,3/
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psset(join=round)\psline[1,2,3,1]\psset(join=\defaultjoin)
\psaltitude 5[3,1,2]\psaltitude 5[2,3,1]
\psset arrowhead(ratio=0.2)\psnormal 20,0.5 [1,3]
\psarrowcircP 1;20 [3,2]
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPSfile}
\figsetptname{\$P_{#1}$}
\figwriten 1,2,12:(4)
\figwrites 3,13:(4)
\figptendnormal 4 :: 20,0.5 [1,3]\figwritew 4:$\vec{n}_2$(4)
}
\centerline{\box\demo}
```

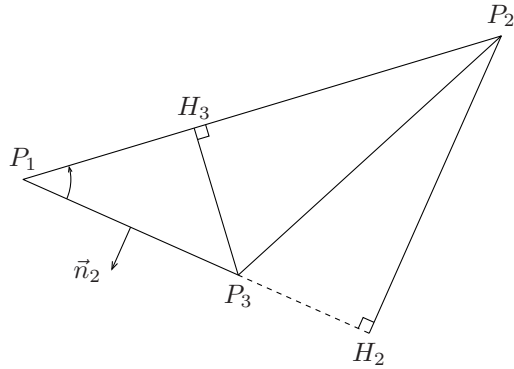


Figure 30

We may be interested in locating the following characteristic points of a triangle  $(P_1, P_2, P_3)$ :

- the intersection of the medians, which is the center of gravity,
- the intersection of the interior (resp. exterior) angle bisectors, which is the center of the inscribed circle or incenter (resp. center of the escribed circle or excenter),
- the intersection of the mid-perpendiculars, which is the center of the circumscribed circle,
- the intersection of the altitudes, which is the orthocenter.

The computation of the gravity center is straightforward since it is the barycenter of  $(P_1, P_2, P_3)$  bearing each the coefficient  $1/3$ . The incenter and excenter are also computed as barycenters of  $(P_1, P_2, P_3)$  (the weights are the length of the edges with the correct sign) and the last two points are obtained by lines intersection. This lead to the following macros:

```

\figptexcenter NewPt :Text [Pt1,Pt2,Pt3]
\figptincenter NewPt :Text [Pt1,Pt2,Pt3]
\figptcircumcenter NewPt :Text [Pt1,Pt2,Pt3]
\figptorthocenter NewPt :Text [Pt1,Pt2,Pt3]

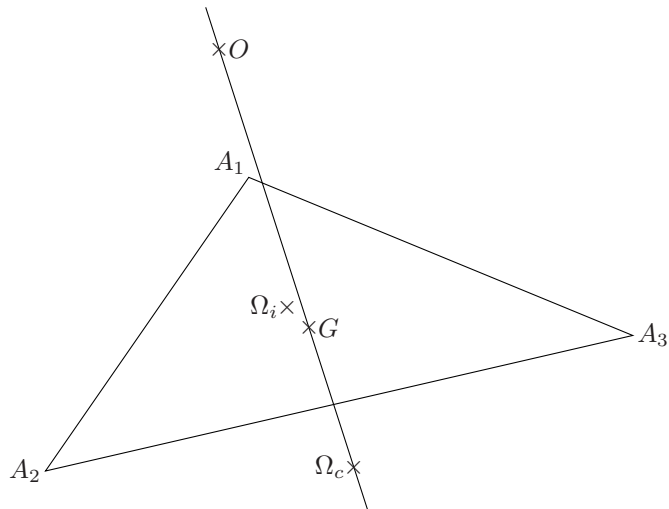
```

The figure 31 shows these four points along with Euler's line. The text of the program that produces it is:

```

% 1. Definition of characteristic points
\figinit{0.85pt}
\figpt 1:(80, 120)
\figpt 2:(-10, -10)
\figpt 3:(250, 50)
\figptbary 5:$G$[1,2,3 ; 1,1,1]
\figptincenter 6:$\Omega_i$[1,2,3]
\figptcircumcenter 7:$\Omega_c$[1,2,3]
\figptorthocenter 8:$O$[1,2,3]
\figptbary10:[7,8;-11,1]\figptbary11:[7,8;1,-11] % Euler's line
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psline[1,2,3,1]\psline[10,11]
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}\Euler's line in a triangle}{
\figinsert{\MyPSfile}
\figwritenw 1:(2)\figwritew 2:(2)\figwritee 3:(2)
\figsetmark{\times}
\figwritee 5,8:(4)\figwritew 6,7:(4)
}
\centerline{\box\demo}

```



**Figure 31** Euler's line in a triangle

## 9. Grid

The following macro can be used in the domain of finite element methods or in approximation theory. Its prototype is

```
\pstrimesh Type [Pt1,Pt2,Pt3]
```

It draws a triangle of type **Type**, according to finite element terminology, which is related to the degree of approximation, represented by the so-called Bézier mesh inside the triangle. The attributes of the line used to draw this mesh are controlled by the secondary settings described in the **Setting graphical attributes** section: the line width, the line style and its color can be modified with the macro `\psset` with **second** as keyword and respectively **width**, **dash** and **color** as attributes.

Figure 32 shows some examples. The text of the program that produces this figure is:

```
% 1. Definition of characteristic points
\figinit{pt}
\figpt 1:(-10, 10)
\figpt 2:(110, 40)
\figpt 3:(50, -40)
\figptbary 11:[1,2,3 ; 1,1,1]\figpttraC 11:=11/0,40/
\figvectC 10(150,0)\figptstra 12=11,12/1,10/
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\pstrimesh 1 [1,2,3]
\figptstra 1=1,2,3,4/1,10/\pstrimesh 2 [1,2,3]
\psset second(dash=5)
\figptstra 1=1,2,3,4/1,10/\pstrimesh 3 [1,2,3]
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}{
\figinsert{\MyPSfile}
\figwritec[11]{Type 1}\figwritec[12]{Type 2}\figwritec[13]{Type 3}
}
\centerline{\box\demo}
```

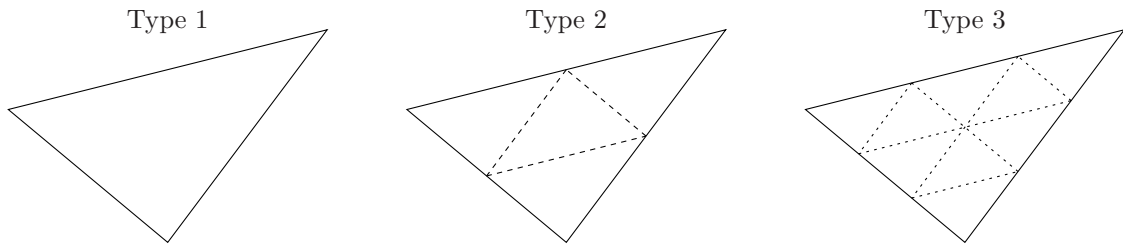


Figure 32

Drawing grids is also often necessary. The macro

```
\psmesh N1,N2 [Pt1,Pt2,Pt3,Pt4]
```

draws a mesh of  $N1 \times N2$  intervals in the quadrangle (Pt1, Pt2, Pt3, Pt4). More precisely, the two segments [Pt1, Pt2] and [Pt3, Pt4] are divided into  $N1$  equal parts, while the two other segments are divided into  $N2$  equal parts. The four points, in the order they are given, are logically assumed to define a convex polygon (although the macro works as well if this condition is not satisfied). As for the previous macro, the attributes of the line used to draw the internal mesh are controlled by the secondary settings described in the **Setting graphical attributes** section.

Moreover, with the help of the macro

```
\psset mesh(diag=Index)
```

it is possible to draw a triangulation. This macro sets a flag that controls the drawing of a grid mesh:

- . if  $Index = 1$ , the SW-NE diagonal is drawn inside each cell,
- . if  $Index = -1$ , the NW-SE diagonal is drawn inside each cell,
- . otherwise, each cell is empty ; this is the default, which can be set by saying:

```
\psset mesh(diag=\defaultmeshdiag)
```

Notice that the orientation of the diagonal depends on the order of the four given points. Figure 33 shows an example where the four points are the vertices of a rectangle and of a general quadrangle.

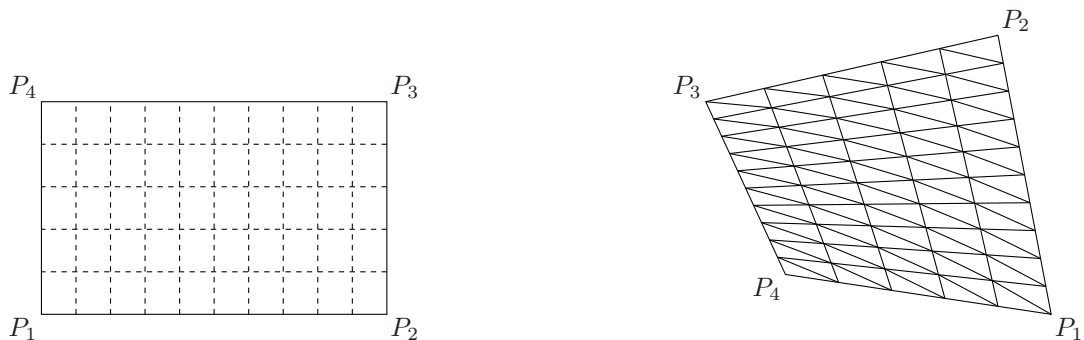


Figure 33

The text of the program that produces this figure is:

```
% 1. Definition of characteristic points
\figinit{pt}
\figpt 1:(0,0)
\figpttraC 2:=1/130,0/\figpttraC 3:=2/0,80/\figpttraC 4:=3/-130,0/
\figvectC 0(250,0)
\figptstra 11=2,3,4,1/1,0/\figpttraC 14:=14/30,15/\figpttraC 12:=12/-20,25/
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psmesh 10,5 [1,2,3,4]
\psset second(dash=\defaultdash)\psset mesh(diag=1)\psmesh 10,5 [11,12,13,14]
\psendfig
```

```

% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}{\figinsert{\MyPSfile}
\figwritesw 1:$P_1$(2)\figwritese 2:$P_2$(2)
\figwritene 3:$P_3$(2)\figwritenw 4:$P_4$(2)
\figwritese 11:$P_1$(2)\figwritene 12:$P_2$(2)
\figwritenw 13:$P_3$(2)\figwritesw 14:$P_4$(2)
}
\centerline{\box\demo}

```

## 10. Flow chart

A flow chart is a set of *nodes* connected to each other by arrows or lines. Each node can bear a text written into a frame. To create such a flow chart, one has to do the following actions:

1. Define points that are the locations of the nodes, more precisely the centers of the frames. It is advised to associate the text of the nodes at this level. The macro to use is `\figpt`, or any macro that create a point (`\figptbary,...`).
2. Create the postscript file containing the frames and the connections using the macros `\psfconnect` and `\psfnode`, *in this order*. Their prototypes are:

```

\psfconnect [Pt1,Pt2,... ,PtN]
\psfnode [Pt1,Pt2,... ,PtN] {Text}

```

3. Write the text corresponding to each node using the macro `\figwritec`, whose prototype is, as already seen in the section **Writing text on the figure**:

```

\figwritec [Pt1, Pt2, ..., PtN] {Text}

```

The appearance of the flow chart can be tuned by the macro `\psset` with the keyword made equal to `flowchart`. Moreover, the keywords `first`, `second`, `third`, `arrowhead` and `curve` may also be used.

Remarks:

- Some of the points defined during the action 1 above may be additional points which are not nodes, but which are used to define paths when non straight lines are needed to connect nodes.
- Each node appears as a frame whose dimensions depend on the text associated with the node. We recall that this text can be any `TEX` material including boxes. From a practical point of view, it is better to specify the text associated with a node when the corresponding point is defined, because in this case the text will be automatically taken into account in the following steps by the macros `\psfnode` and `\figwritec`, which can thus be called without any text argument.
- We stress the fact that, for a given node, if the text argument of the macro `\figwritec` is present, it must be the same as the one specified when the macro `\psfnode` was called, otherwise the dimensions of the frame may be incorrect. However, this text argument can be useful if one wants to write the *same* text on several nodes.

The macro `\psfconnect` draws a line between two nodes, passing through the points given as arguments. The line can be polygonal or can be a smooth curve ( $C^1$  Bézier curve). This attribute is set by saying

```

\psset flowchart(line = type)

```

where *type* can take the value `polygon` or `curve`. The default value is `polygon` (its name is `\defaultfcline`). When the line is polygonal, the corners can be rounded: they are replaced by circular arcs whose radius is chosen by the user by saying

```

\psset flowchart(radius = value)

```

where *value* is a dimension to be given in user coordinates. The default value is 0 (its name is `\defaultfcradius`). A 0 value indicates no rounding.

An arrow-head is drawn along the path, oriented from the node lying at `Pt1` towards the node lying at `PtN`. The attributes of the arrow-head are specified by the macro `\psset` with `arrowhead` as keyword (see section **Arrow**). Moreover, the position of the arrow-head along the path can be modified by saying

```

\psset flowchart(arrowposition = value)

```

where *value* is a real number ranging in  $[0,1]$ , 0 corresponding to Pt1 and 1 corresponding to PtN. The default value is 0.5 (its name is `\defaultfcarrowposition`), and makes the arrow-head to be drawn half way along the path. To be more precise, the position of the arrow-head is defined with respect to its start point or its end point. This reference point is chosen by saying

```
\psset flowchart(arrowrefpt = switch)
```

where *switch* can take the value `start` or `end`. The default value is `start` (its name is `\defaultfcarrowrefpt`).

Remarks:

- If no arrow-heads are wanted, the best is to set the arrow-head length to 0 by saying: `\psset arrowhead(length=0)`. An alternative solution is to use directly `\psline` or `\pscurve`.
- The `arrowposition` attribute is especially useful when the line path is polygonal because an arrow-head may appear too close to a corner, or even worse, outside the path if the corners are rounded. This situation must be handled manually by modifying the arrow-head position along the path.

Once the connections are settled, the frames can be drawn using the macro `\psfcnode`. Several shapes are available. This attribute can be specified by saying

```
\psset flowchart(shape = value)
```

where *value* can be one out of `rectangle`, `ellipse` or `lozenge`. The default value is `rectangle` (its name is `\defaultfcshape`). The dimensions of the frame are determined by the text associated with the node. If the text argument of `\psfcnode` is empty, the text joined to the corresponding point is taken into account. Starting from this initial proposal, the dimensions of the frame can be reduced or enlarged by specifying padding values, horizontally or vertically. This can be achieved by saying

```
\psset flowchart(xpadding = value1, ypadding = value2)
```

where *value1* and *value2* are positive or negative dimensions to be given in user coordinates. One can also simply say

```
\psset flowchart(padding = value)
```

which gives the same padding *value* horizontally and vertically. The default values are 0 and 0 (their names are `\defaultfcxpadding` and `\defaultfcypadding`).

The last attribute of a frame is its “thickness” which makes it look as if it was enlightened from the top left corner of the page. The dimension of this thickness is to be given in user coordinates by saying

```
\psset flowchart(thickness = value)
```

The default value is 0 (its name is `\defaultfcthickness`).

Remarks:

- The color, the line width or style of a frame border can be modified by the primary attributes. The color of the thickness is governed by the secondary color attribute. The color of the background of the frame is the ternary color attribute, which can be set by saying `\psset third(color = New Color)`.
- If no border is wanted, one has to make it invisible (white) by saying `\psset(color=1)` for example before calling `\psfcnode`, and reset the primary color to its previous value afterwards if needed.

On figure 34, we can see a simple network showing what can be obtained with the help of the previous macros. The text of the program that produces it is:

```
% 1. Definition of characteristic points
\figinit{cm}
% Node locations
\figpt 1:$A$(0,0)\figpt 2:$B$(-2,2)\figpt 3:$C$(3,2)
\figpt 4:$D$(-1.5,-2)\figpt 5:$E$(1.1,-1.5)
% Additional point to define the curved path
\figpt 0:(2,-2)
%
```



```

% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
% Draw the connections between nodes
\psfconnect[1,2]\psfconnect[1,3]\psfconnect[1,4]
\psfconnect[2,3]\psfconnect[2,4]
\psfconnect[3,5]
\psset flowchart(line=curve)\psfconnect[4,0,3]
% Draw the frames at each node
\psset (color=1)\psset flowchart(shape=ellipse)
\psfcnode[1,2,3,4,5]{ }
\psendfig
%
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}}{
\figinsert{\MyPSfile}
\figwritec[1,2,3,4,5]{ }
}
\centerline{\box\demo}

```

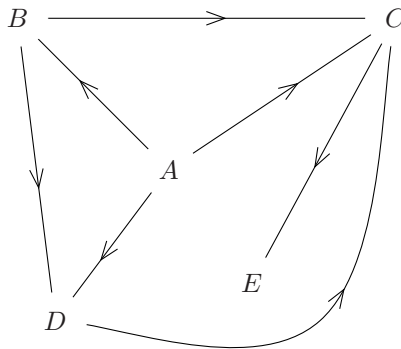


Figure 34

Another example can be found on figure 2. It shows a diagram where the principle of the `fig4tex` macro-package is summarized. It has been produced by the following program:

```

% 1. Definition of characteristic points
\figinit{cm}
% Two vectors to translate points horizontally and vertically:
\figvectC 1(1,0)\figvectC 2(0,-1)
\def\FrameWidth{3cm}
\figpt 10:\centertext{4cm}{\tt fig4tex} \bf macro-package\break
instructions diagram}(0,0)
\figpttra 20:\lefttext{\FrameWidth}{\tt\char'\input fig4tex\break
\char'\pssetdefault...}=10/3,2/
\figpttra 30:\lefttext{\FrameWidth}{\tt\char'\figinit$\{...\}$}\break
\char'\figpt...\break
...}=20/2.2,2/
\figpttra 40:\lefttext{\FrameWidth}{\tt\char'\psbeginfig$\{...\}$}\break
...\break
\char'\psendfig}=30/2.5,2/
\figpttra 50:\centertext{3.2cm}{Another\break
postscript file ?}=40/2.5,2/
\figpttra 51:yes=50/3,1/

```

```

\figpttra 60:\lefttext{\FrameWidth}{\tt\char'\figvisu...$\{$\break
\char'\figinsert...\break
\char'\figwrite...\break
$\}$}=50/3,2/
\figpttra 70:\lefttext{4.7cm}{Display the figure.\break
Continuation of the document.}=60/2.5,2/
\figpttra 80:Another figure ?=70/2,2/
\figpttra 81:yes=80/5,1/
%
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
% Draw the connections between nodes
\psfcconnect[20,30]\psfcconnect[30,40]
\psset flowchart(arrowpos=0.4)\psfcconnect[40,50]\psreset{flowchart}
\psfcconnect[50,60]\psfcconnect[60,70]\psfcconnect[70,80]
\psset arrowhead(fillmode = yes)\psset flowchart(radius=0.5)
\figpttra 41:=40/3,1/\psfcconnect[50,51,41,40]
\psset flowchart(radius=1)
\figpttra 31:=30/5,1/\psfcconnect[80,81,31,30]
% Draw the frames at each node
\psfcnode[20,30,40,60]{}
\psset third(color=0.9)
\psfcnode[70]{}
\psset flowchart(shape=lozenge,xpadding=-0.6)
\psfcnode[50]{}
\psset flowchart(xpadding=\defaultfcxpadding)
\psfcnode[80]{}
\psset flowchart(shape=ellipse,thick=0.1)\psset second(color=0.3)
\psreset{third}\psfcnode[10]{}
\psendfig
%
% 3. Writing text on the figure
\def\Xoffset{7}\def\CommentWidth{4cm}
\figvisu{\demo}{\bf Figure \the\Figno}}{\figinsert{\MyPSfile}
\figwritec[10,20,30,40,50,60,70,80]{}
\figwrites 51,81:(0.1)
\figwritee 30:\lefttext{\CommentWidth}{\bf Step 1.} Definition of\break
characteristic points.}{-\Xoffset}
\figwritee 40:\lefttext{\CommentWidth}{\bf Step 2.} Creation of a\break
postscript file.}{-\Xoffset}
\figwritee 60:\lefttext{\CommentWidth}{\bf Step 3.} Assembling the\break
figure and writing text.}{-\Xoffset}
\figpttra 0:=30/\Xoffset,1/\figwritep[0]% For the symmetry of the figure
}
\centerline{\box\demo}

```

Remarks:

- In order to display the text associated with the nodes more easily, the two macros `\centertext` and `\lefttext` have been used. Their aim is to display several lines of text, centered or left justified into a rectangular box whose width is given as first argument. The second argument is a text whose lines are separated by the control sequence `\break`. These macros are included into the `fig4tex.tex` file.
- The locations of the nodes are deduced from each other by applying translations. This facilitates the modification of the diagram when it is being built.
- At the end of step 3, the point `#0` is created only to “enlarge” the figure towards the right in order to make the legend be centered under the diagram.

## §9. Helpers

This section gathers some tools designed to help the user while he is creating a figure.

We first recall the existence of the macro `\psset(update=yes/no)` that can be used before `\psbeginfig` in order to enforce the postscript file to be updated at next compilation (see subsection **Principle**).

It often happens that points are defined, but will never be displayed on the figure because they are only used for the geometric construction. However, it is also often useful to show these points temporarily on the figure. For example, when defining a curve, this helps to adjust the position of the interpolating points. The macro `\figshowpts[Nmin, Nmax]` does this job: it displays on the figure every point defined since the beginning of the session, whose number lies in the interval `[Nmin, Nmax]` (which must not be too large, otherwise, an error message `! TeX capacity exceeded` may occur). The location of each point is marked by a bullet along with its joined text or, by default, its number. Obviously, this macro must be called during step 3.

We have seen that the user can modify some parameters such as those concerning the arrow-head, the line style, the color, etc. . . The macro `\figshowsettings` prints on the terminal the *current* value of each of these parameters, ordered by subject. This macro can be called at any place in the process.

We also recall that one can easily insert an existing encapsulated postscript file with the macro `\figinsert`. It is then possible to add any text on it: the macro `\figscan` is specially helpful to locate the position of the attach point. Refer to the section **Writing text on the figure** for more information.

## §10. Three-dimensional macros

### 1. Introduction

The explanations given so far concerned two-dimensional geometry. It so happens that, in their major part, they also hold for three-dimensional geometry. We can switch the 3D-mode on with the help of the macro `\figinit`. In the following sections, we describe how to use the macros in this mode. There are three sets of macros:

- macros that work only in the plane  $z = 0$ , which are: `\figptell`, `\figptendnormal`, `\figptsinterlinell`, `\psarcell`, `\psnormal`, `\psfcconnect`, `\psfcnode`.  
They exist because either the list of arguments is specific to the 2D (which is the case for example for `\figptell`), or the authors judged that the macro is very rarely used in 3D. For some of them, other versions fully compatible with 3D-mode are available. They can however be used in 3D-mode: they simply do not take the third component  $z$  into account during the computation.
- macros that are specific to 3D, which are: `\figptinterlineplane`, `\figptorthoprojplane`, `\figpt-sorthoprojplane`, `\figptvisilimXX`.  
Their existence is simply due to the fact that they have no equivalent in 2D. Thus, any attempt to use them in 2D-mode brings on the message  

```
*** The macro macro name is not available in the current context.
```

to be printed on the terminal.
- macros that can be used in both dimensions, which are all the other ones. These macros have the same name in 2D and in 3D, although one additional argument may be needed in 3D. Thus, a macro can have a slightly different calling sequence in 2D and in 3D. This is detailed in a further section.

The macros are still used in the same three steps scheme: 1) definition of characteristic points, 2) creation of the postscript file, 3) writing on the figure if needed. However, we need here one more tool, a projection, which is used during steps 2) and 3).

### 2. Projection

Three projections are available : cavalier, orthogonal and realistic. The latter is a perspective (or conical) projection, the first two are parallel (or cylindrical) projections.

The choice is made with the help of the macro `\figinit` (see next section).

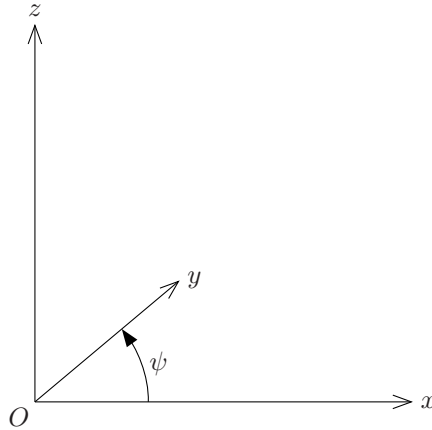
Then, we can modify the parameters governing the projection with the macro `\figset` with `proj` as keyword.

The **cavalier projection** is the default one. It provides a false perspective view and is generally used to show objects with parallel faces. Its characteristics are (see figure 35):

- the plane  $(x, z)$  always corresponds to the paper sheet plane,
- the three-dimensional effect can be tuned by the angle  $\psi = (Ox, Oy)$  and a real number  $\lambda$  usually chosen between 0 and 1, which is the depth reduction coefficient of the perspective.

The angle  $\psi$  can be changed by saying `\figset proj(psi=value)`. The default value is 40 degrees. Values of  $\psi$  such that  $0 \leq \psi \leq 180$  correspond to a view from above and to a view from beneath if  $-180 \leq \psi \leq 0$ .

In fact, the genuine *cavalier* projection is obtained with  $\lambda = 1$ . In this projection, all edges of a cube have the same length. This property may be interesting since measurements can be made directly. However, we may find that the objects seem too elongated. That is why the default value is  $\lambda = 0.5$ , which corresponds to the so-called *cabinet* projection and lead to a more realistic result. The coefficient  $\lambda$  can be changed by saying `\figset proj(lambda=value)`.



**Figure 35** Definition of the cavalier projection

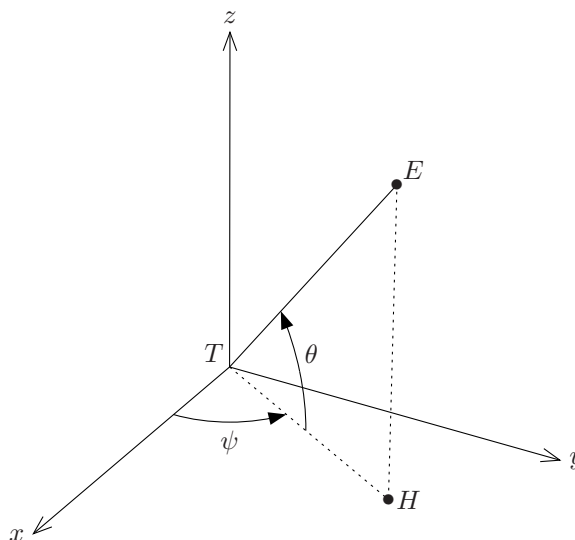
The text of the program that produces this figure is given below, although we are anticipating on the description of some of the macros used there. Just notice that the default settings are used, so no call to `\figset` is done.

```
% 1. Definition of characteristic points
\figinit{5cm, 3D}
\figpt 0:$0$(0,0)
\figptsaxes 1:0(1)
\def\Rpsi{0.3}\figptcirc 4:$\psi$:0,1,3;\Rpsi(20)
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psaxes 0(1)
\psset arrowhead(length=0.05,fillmode=yes)
\psarrowcirc 0,1,3;\Rpsi(0,40)
\psendfig
% 3. Writing text on the figure
\def\dist{3pt}
\figvisu{\demo}{\bf Figure \the\Figno}\Definition of the cavalier projection}{
\figinsert{\MyPSfile}
\figwritesw 0:(\dist)\figwritee 1,2,4:(\dist)\figwriten 3:(\dist)}
\centerline{\box\demo}
```

The **realistic projection** is defined by a target point  $T$  and an observation point  $E$ , also called eye point (see figure 36). The coordinates of the target point are those of a point  $Pt$  already defined or computed. They are set with the help of the macro `\figset proj` by saying `\figset proj(targetpt = point number)`. Once the position of the target point is set,  $Pt$  can be modified. By default, the target point is the center of the bounding box of the points defined so far. Then, the eye point position is defined by:

- the observation distance  $TE$ , which can be set by saying `\figset proj(distance = value)`. By default, this distance is equal to 5 times the largest dimension of the bounding box of the points defined so far.
- the longitude  $\psi = (\vec{x}, \vec{TH})$  and the latitude  $\theta = (\vec{TH}, \vec{TE})$ , angles to be given in degrees, that can be set by saying `\figset proj(psi=value1, theta=value2)`.  $H$  is the orthogonal projection of  $E$  onto the plane  $(T, x, y)$ .

The text of the program that produces the figure 36 is given below it.



**Figure 36** Definition of the perspective projection

Corresponding program :

```

% 1. Definition of characteristic points
\figinit{5cm, realistic}
\figpt 0:$T$(0,0)
\figptsaxes 1:0(1)
\figpt 4:$E$(0.4,0.7,0.9)
\figvectP10[0,3]\figptorthoprojplane 5:$H$(4/0,10/
\def\Rpsi{0.3} \figptcirc 6:$\psi$:0,1,2;\Rpsi(30)
\def\Rtheta{0.4}\figptcirc 7:$\theta$:0,5,3;\Rtheta(25)
% 2. Creation of the postscript file
\figset proj(psi=30,theta=30)
\psbeginfig{\MyPSfile}
\psaxes 0(1)
\psline[0,4]
\psset arrowhead(length=0.05,fillmode=yes)
\psarrowcircP 0;\Rpsi[1,5,2]\psarrowcircP 0;\Rtheta[5,4,3]
\psset(dash=5)\psline[0,5,4]
\psendfig
% 3. Writing text on the figure
\def\dist{3pt}
\figvisu{\demo}{\bf Figure \the\Figno}\Definition of the perspective projection}{
\figinsert{\MyPSfile}
\figwritenw 0:(\dist)\figwritew 1:(\dist)\figwritte 2:(\dist)\figwriten 3:(\dist)
\figwrites 6:(\dist)\figwritene 7:(\dist)
\figsetmark{\bullet}\figwritene 4:(\dist)\figwritte 5:(\dist)}
\centerline{\box\demo}

```

Notice that if the distance  $TE$  tends to infinity, the projection tends to be the orthogonal projection onto the plane, i.e. the paper sheet. No control is made on the value given as argument ; a too big one may bring on **TEX** to print the error message “! Dimension too large.” and stop the compilation process.

The **orthogonal projection** is a simplified version of the previous one since the eye point  $E$  is placed at infinity. It is an orthogonal projection onto a plane which is itself orthogonal to the observation direction. As a consequence, this projection is only defined by the observation direction, i.e. by the two angles  $\psi$  and  $\theta$  (see figure 36), which are set with the macro `\figset proj()`.

### 3. Macro specifications

In this section, we focus on macros that are specific to the 3D and macros that do not have the same calling sequence in both dimensions. Notice that for those last macros, an error occur if we attempt to use them with the 2D calling sequence. The other macros are not mentionned here since their description, which is valid in 3D, can be found in the previous sections.

#### Control

The full prototype of `\figinit` is:

`\figinit {ScaleFactorUnit, Dim/Proj}`

The first argument `ScaleFactorUnit` has been described in the **Unit and scale** subsection. The second argument `Dim/Proj` is optional. It sets the dimension of the space to be used until next call to `\figinit`. In 3D, it also sets the projection type. If this argument is absent or is given the value “2D”, then geometry in the plane is assumed, otherwise geometry in three dimensions is performed. Moreover, if this argument is equal to:

- . `orthogonal`, then the orthogonal projection is used,
- . `realistic`, then the realistic projection is used,
- . 3D or `cavalier` (or anything else), then the cavalier projection is used.

#### Vector creation

The definition of a vector by its components  $(X,Y,Z)$  can be done with the help of the macro whose prototype is:

`\figvectC NewVect (X,Y,Z)`

Notice that the 2D calling sequence `\figvectC NewVect (X,Y)` is still allowed, implying Z to be 0.

The macro

`\figvectN NewVect [Pt1,Pt2,Pt3]`

defines a vector normal to the plane defined by the 3 points `Pt1`, `Pt2` and `Pt3`. More precisely, let  $P_1$ ,  $P_2$ ,  $P_3$  and  $\vec{V}$  stand for `Pt1`, `Pt2`, `Pt3` and `NewVect`.

This macro computes  $\vec{V} = \frac{\overrightarrow{P_1P_2}}{\|\overrightarrow{P_1P_2}\|} \times \frac{\overrightarrow{P_1P_3}}{\|\overrightarrow{P_1P_3}\|}$ . Thus,  $(\overrightarrow{P_1P_2}, \overrightarrow{P_1P_3}, \vec{V})$  is a positively oriented basis.

The macro

`\figvectNV NewVect [Vector1, Vector2]`

defines a vector normal to the two vectors `Vector1` and `Vector2`.

More precisely, let  $\vec{V}_1$ ,  $\vec{V}_2$  and  $\vec{V}$  stand for `Vector1`, `Vector2` and `NewVect`.

This macro computes  $\vec{V} = \frac{\vec{V}_1}{\|\vec{V}_1\|} \times \frac{\vec{V}_2}{\|\vec{V}_2\|}$ . Thus,  $(\vec{V}_1, \vec{V}_2, \vec{V})$  is a positively oriented basis.

#### Basic macros

The macro

`\figpt NewPt :Text(X,Y,Z)`

creates a point with coordinates  $(X,Y,Z)$ . The third coordinate Z can be omitted, in which case the value  $Z = 0$  is assumed. In other words, the 2D calling sequence `\figpt NewPt :Text(X,Y)` is valid in 3D-mode as well.

Let  $C$ ,  $P_1$ ,  $P_2$  and  $P_3$  stand for `Center`, `Pt1`, `Pt2` and `Pt3`. The macro

`\figgetangle\Value[Center,Pt1,Pt2,Pt3]`

computes the value (in degrees) of the oriented angle  $(\overrightarrow{CP_1}, \overrightarrow{CP_2})$  and stores it in the macro `\Value` whose name is chosen by the user. The angle is measured counterclockwise around the vector  $\overrightarrow{CP_1} \times \overrightarrow{CP_3}$ . The points  $C$ ,  $P_1$ ,  $P_2$  and  $P_3$  must be coplanar, but  $P_3$  must not lie on the line  $(C, P_1)$ .

## Point transformation

Some transformation macros have been adapted for the 3D-mode. Each of them computes the image `NewPt` of the point `Pt`. Let  $P$ ,  $P'$ ,  $C$  and  $\vec{V}$  stand for `Pt`, `NewPt`, `Center` and `Vector`.

The macro

```
\figptrot NewPt :Text= Pt /Center, Angle, Vector/
```

refers to the rotation of angle `Angle` (given in degrees) around the axis  $(C, \vec{V})$ . The sense of rotation is such that  $(\vec{CP}, \vec{CP'}, \vec{V})$  is a positively oriented basis.

The macro

```
\figptsym NewPt :Text= Pt /PlanePt, NormalVector/
```

refers to the orthogonal symmetry with respect to the plane defined by the point `PlanePt` and the vector `NormalVector` which is normal to the plane. This vector may be computed with the help of `\figvectN` or `\figvectNV`.

The macro

```
\figpttraC NewPt :Text= Pt /X,Y,Z/
```

refers to the translation of vector  $(X, Y, Z)$ .

The macro

```
\figptmap NewPt :Text= Pt /InvPt/A11,A12,A13 ; A21,A22,A23 ; A31,A32,A33/
```

refers to the linear affine mapping such that  $\vec{IP'} = A \vec{IP}$  where  $A$  is the matrix the coefficients of which are  $A_{ij}$ ,  $1 \leq i \leq 3$ ,  $1 \leq j \leq 3$  and  $I$  stands for `InvPt`.

The macro

```
\figptorthoprojplane NewPt :Text= Pt /PlanePt, NormalVector/
```

refers to the orthogonal projection onto the plane defined by the point `PlanePt` and the vector `NormalVector` which is normal to the plane.

The corresponding available “set” versions have then the following prototypes:

```
\figptsrot NewPt1 = Pt1, Pt2, ..., PtN /Center, Angle, Vector/
```

```
\figptssym NewPt1 = Pt1, Pt2, ..., PtN /PlanePt, NormalVector/
```

```
\figptsmap NewPt1 = Pt1, Pt2, ..., PtN /InvPt/A11,A12,A13; A21,A22,A23; A31,A32,A33/
```

```
\figptsorthoprojplane NewPt1 = Pt1, Pt2, ..., PtN /PlanePt, NormalVector/
```

## Point computation

The macro

```
\figptcirc NewPt :Text: Center,Pt1,Pt2;Radius (Angle)
```

computes the coordinates of a point lying on a circle. Let  $C$ ,  $P_1$  and  $P_2$  stand for `Center`, `Pt1` and `Pt2`. The circle, of center  $C$  and radius `Radius`, is lying in the plane  $(C, P_1, P_2)$ . The two points  $P_1$  and  $P_2$  do not need to belong to the circle. The position of the point is defined by the angle `Angle` (given in degrees), measured counterclockwise around the vector  $\vec{CP_1} \times \vec{CP_2}$ , starting from the half-line  $(C, P_1)$ .

The macro

```
\figptinterlineplane NewPt :Text[LinePt,Vector; PlanePt,NormalVector]
```

computes the intersection of the line defined by the point `LinePt` and the vector `Vector`, and the plane defined by the point `PlanePt` and the vector `NormalVector` which is normal to the plane.

The macro

```
\figptvisilimSL NewPt :Text[SegPt1,SegPt2 ; LinePt1,LinePt2]
```

computes the apparent intersection of the segment `[SegPt1, SegPt2]` and the line defined by the two points `LinePt1` and `LinePt2`. This macro is useful to compute the visible limit of the segment hidden by an object whose boundary is delimited by the given line. If the projection of the line intersects the segment, the solution `NewPt` lies between `SegPt1` and `SegPt2`, otherwise `SegPt1` is the result point.



The macro

`\figptsintercirc NewPt1 [Center1,Radius1 ; Center2,Radius2 ; PlanePt]`

computes the intersections of the two circles defined by their center and radius. The plane containing the two circles is defined by the three points `Center1`, `Center2` and `PlanePt`. They must not lie on the same line ; this is not checked. Let  $C_1$ ,  $C_2$  and  $P$  stand for `Center1`, `Center2` and `PlanePt`, and  $S_1$ ,  $S_2$  stand for the two solutions. The points  $S_1$  and  $S_2$  are ordered so that the vectors  $\overrightarrow{C_1C_2} \times \overrightarrow{C_1P}$  and  $\overrightarrow{C_1S_1} \times \overrightarrow{C_1S_2}$  have the same direction.

The macro

`\figptsaxes NewPt1 : Origin(X1,X2, Y1,Y2, Z1,Z2)`

computes the end points of the axes corresponding to the arrows drawn by the calling statement:

`\psaxes Origin(X1,X2, Y1,Y2, Z1,Z2)`.

If `NewPt1` is equal to  $k$ , then the three resulting points bear the numbers  $k$ ,  $k + 1$  and  $k + 2$ , and the default text  $x$ ,  $y$  and  $z$  is joined to them. A short form exists: `\figptsaxes NewPt1 : Origin(L)`. It is equivalent to `\figptsaxes NewPt1 : Origin(0,L, 0,L, 0,L)`.

### Drawing macros

Let  $C$ ,  $P_1$  and  $P_2$  stand for `Center`, `Pt1` and `Pt2`. In the three following macros, we consider a circle (or circular arc) of center  $C$  and radius `Radius`, lying in the plane  $(C, P_1, P_2)$ . The two points  $P_1$  and  $P_2$  do not need to belong to the circle. The point  $P_2$  must not lie on the line  $(C, P_1)$ .

These macros are:

`\pscirc Center,Pt1,Pt2 (Radius)`

which draws the entire circle,

`\psarccirc Center,Pt1,Pt2 ; Radius (Ang1,Ang2)`

which draws the circular arc limited by the angles `Ang1` and `Ang2` given in degrees and measured counterclockwise around the vector  $\overrightarrow{CP_1} \times \overrightarrow{CP_2}$ , starting from the half-line  $(C, P_1)$ ,

`\psarrowcirc Center,Pt1,Pt2 ; Radius (Ang1,Ang2)`

which draws the circular arrow beared by the circular arc defined just before. If `Ang2` > `Ang1`, the arrow is drawn counterclockwise, else it is drawn clockwise. The arrow-head is drawn according to the `\psarrowhead` macro settings.

The last two macros have a so-called ‘‘point’’ version. Let  $P_3$  stand for `Pt3` and  $\vec{N} = \overrightarrow{CP_1} \times \overrightarrow{CP_3}$ . In the two following macros, we consider a circular arc of center  $C$ , lying in the plane  $(C, P_1, P_3)$  and limited by the two half-lines  $(C, P_1)$  and  $(C, P_2)$ . The three points  $P_1$ ,  $P_2$  and  $P_3$  do not need to belong to the circle. The points  $C$ ,  $P_1$ ,  $P_2$  and  $P_3$  must be coplanar, but  $P_3$  must not lie on the line  $(C, P_1)$ .

These macros are:

`\psarccircP Center ; Radius [Pt1,Pt2,Pt3]`

which draws the circular arc of radius `Radius`, from  $P_1$  towards  $P_2$  turning counterclockwise around the axis  $(C, \vec{N})$ ,

`\psarrowcircP Center ; Radius [Pt1,Pt2,Pt3]`

which draws the circular arrow of radius `|Radius|`, from  $P_1$  towards  $P_2$  turning around the axis  $(C, \vec{N})$ :

- . counterclockwise if the radius is positive,
- . clockwise if the radius is negative.

Remarks:

- An arrow (or an arrow-head) is always two-dimensional because the arrow which is really drawn is the two-dimensional arrow whose body is the projection of the true 3D arrow body.
- In the same order of idea, non-mute macros (whose name begin with `\figwrite`) write their text in the plane of the sheet with respect of the position of the projection of the attach point.

The macro

`\psaxes Origin(X1,X2, Y1,Y2, Z1,Z2)`

draws axes crossing at the point denoted `Origin`. Each axis is parallel to the corresponding absolute axis and is drawn as an oriented arrow between two values given in user coordinates: from `X1` to `X2` for the  $X$ -axis, from `Y1` to `Y2` for the  $Y$ -axis, from `Z1` to `Z2` for the  $Z$ -axis. A second form of the macro can be used: `\psaxes Origin(L)`. It is equivalent to `\psaxes Origin(0,L, 0,L, 0,L)`.

## 4. 3D examples

### • Cube

We begin with the representation of the unit cube defined by its 8 vertices  $P_1, P_2, \dots, P_8$  as follows:

```
% 1. Definition of characteristic points
\figinit{3cm, 3D}
\figpt 1:(0,0)
\figpttraC 2:=1/1,0,0/
\figpttraC 3:=1/1,1,0/
\figpttraC 4:=1/0,1,0/
\figvectC 10(0,0,1)\figptstra 5=1,2,3,4/1,10/
\figsetptname{\$P_#1$}
```

As we will need to draw the cube several times, we define the macro `\DrawCube` to create the postscript file whose name is contained in the macro `\MyPSfile`. The background lines are dashed ; obviously, they depend on the observation direction.

```
\def\DrawCube{
% 2. Creation of the postscript file
\psbeginfig{\MyPSfile}
\psline[1,2,3]\psline[5,6,7,8,5]
\psline[1,5]\psline[2,6]\psline[3,7]
\psset(dash=4)\psline[1,4,3]\psline[4,8]
\psendfig}
```

In the same way, to create the box and write the text on the figure, we define the macro `\WriteOnCube` which has the name of the box register as first argument and the legend to be printed under the figure as second argument.

```
\def\WriteOnCube#1#2{
% 3. Writing text on the figure
\def\dist{2pt}
\figvisu{#1}{\bf Figure \the\Figno}\#2}{\figinsert{\MyPSfile}
\figwritesw 1:(\dist)\figwritese 2:(\dist)
\figwritee 3:(\dist)\figwritenw 4:(\dist)
\figwritenw 5:(\dist)\figwritese 6:(\dist)
\figwritene 7:(\dist)\figwritenw 8:(\dist)
}}
```

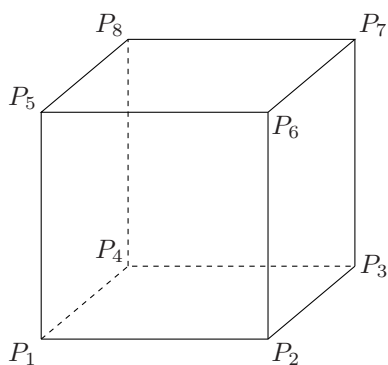


Figure 37 Default settings

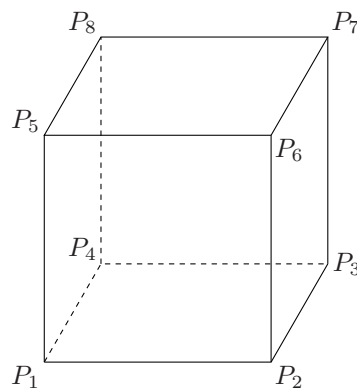


Figure 38  $\psi = 60$

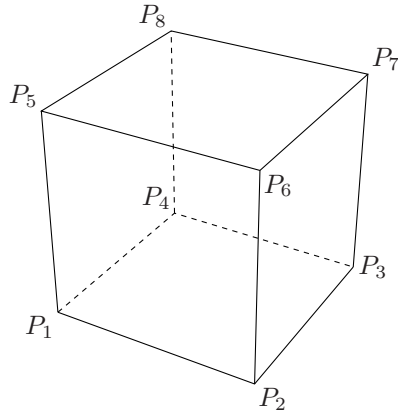
Thus, the figures 37 and 38 are obtained by simply writing:

```

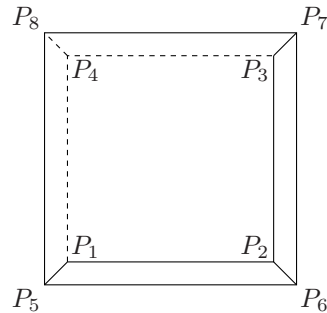
\newbox\demo \newbox\demotwo % if not already done
\NewPSfile{\MyPSfile}
\DrawCube\WriteOnCube{\demo}{Default settings}
\NewPSfile{\MyPSfile}
\def\Valpsi{60} \figset proj(psi=\Valpsi)
\DrawCube\WriteOnCube{\demotwo}{\psi = \Valpsi}
\centerline{\box\demo\hfil\box\demotwo}

```

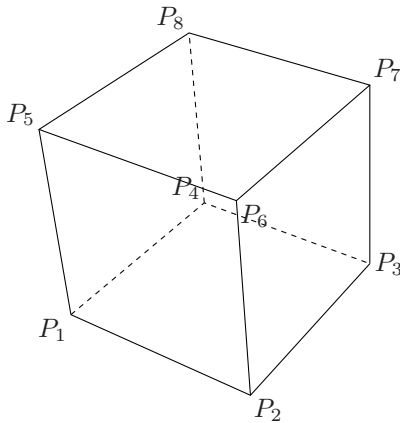
These figures are drawn with the cavalier projection ; we can observe the influence of the angle  $\psi$ . The four following figures use the realistic projection.



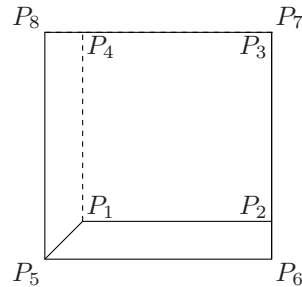
**Figure 39**  $(\psi, \theta) = (-60, 30)$



**Figure 40**  $(\psi, \theta) = (-90, 90)$



**Figure 41**  $(\psi, \theta) = (-60, 30), T = (1, 1, 1)$



**Figure 42**  $(\psi, \theta) = (-90, 90), T = (1, 1, 1)$

We recall that the observation direction is defined by the angles  $(\psi, \theta)$ . The right-hand side figures correspond to the ordinary 2D view, for which the position of the text has been adapted (macro `\WriteOnCubeB`). The program that produces the first two figures is:

```

\figinit{3cm, realistic}
...
\NewPSfile{\MyPSfile}
\def\Valpsi{-60}\def\Valtheta{30} \figset proj(psi=\Valpsi, theta=\Valtheta)
\DrawCube\WriteOnCube{\demo}{\psi = \Valpsi, \theta = \Valtheta}
\NewPSfile{\MyPSfile}
\def\Valpsi{-90}\def\Valtheta{90} \figset proj(psi=\Valpsi, theta=\Valtheta)
\DrawCube\WriteOnCubeB{\demotwo}{\psi = \Valpsi, \theta = \Valtheta}
\centerline{\box\demo\hfil\box\demotwo}

```

After this, if we add the command `\figshowsettings`, we get the following information to be printed on the terminal at compilation time:

```

=====
Current settings about:
--- GENERAL ---
Scale factor and Unit = 3cm (85.35826pt) -> \figinit{ScaleFactorUnit}
Update mode = no -> \psset(update=yes/no) or \pssetdefault(update=yes/no)
--- PRINTING ---
Implicit point name = $P_i$ -> \figsetptname{Name}
Point marker = -> \figsetmark{Mark}
Print rounded coordinates = yes -> \figsetroundcoord{yes/no}
--- POSTSCRIPT (general) ---
First-level (or primary) settings:
Color = 0 -> \psset(color=ColorDefinition)
Filling mode = no -> \psset(fillmode=yes/no)
Line join = miter -> \psset(join=miter/round/bevel)
Line style = 1 -> \psset(dash=Index/Pattern)
Line width = 0.4 -> \psset(width=real in PostScript units)
Second-level (or secondary) settings:
Color = 0 -> \psset second(color=ColorDefinition)
Line style = 4 -> \psset second(dash=Index/Pattern)
Line width = 0.4 -> \psset second(width=real in PostScript units)
Third-level (or ternary) settings:
Color = 1 -> \psset third(color=ColorDefinition)
--- POSTSCRIPT (specific) ---
Arrow-head:
(half-)Angle = 20 -> \psset arrowhead(angle=real in degrees)
Filling mode = no -> \psset arrowhead(fillmode=yes/no)
"Outside" = no -> \psset arrowhead(out=yes/no)
Length = 0.09363 (active) -> \psset arrowhead(length=real in user coord.)
Ratio = 0.1 (not active) -> \psset arrowhead(ratio=real in [0,1])
Curve: Roundness = 0.2 -> \psset curve(roundness=real in [0,0.5])
Mesh: Diagonal = 0 -> \psset mesh(diag=integer in {-1,0,1})
Flow chart:
Arrow position = 0.5 -> \psset flowchart(arrowposition=real in [0,1])
Arrow reference point = start -> \psset flowchart(arrowrefpt = start/end)
Line type = polygon -> \psset flowchart(line=polygon/curve)
Padding = (0, 0) -> \psset flowchart(padding = real in user coord.)
(or \psset flowchart(xpadding=real, ypadding=real) )
Radius = 0 -> \psset flowchart(radius=positive real in user coord.)
Shape = rectangle -> \psset flowchart(shape = rectangle, ellipse or lozenge)
Thickness = 0 -> \psset flowchart(thickness = real in user coord.)
--- 3D to 2D PROJECTION ---
Projection : realistic -> \figinit{ScaleFactorUnit, ProjType}
Longitude (psi) = -90 -> \figset proj(psi=real in degrees)
Latitude (theta) = 90 -> \figset proj(theta=real in degrees)
Observation distance = 4.99493 -> \figset proj(dist=real in user coord.)
Target point = CenterBoundingBox -> \figset proj(targetpt=pt number)
Its coordinates are (0.49948 , 0.49948 , 0.49948 )
=====

```

Remark: In 2D-mode the last section (3D to 2D PROJECTION) is not printed.

We can see that the target point is the center of the cube as expected (numerical values are not rounded

here). If we change it to the point  $P_7$  by saying `\figset proj(targetpt=7)` and then execute the same program, then we obtain the two last figures 41 and 42.

The observation distance can be modified in the same way, for instance by saying

```
\figset proj(distance=10)
```

which doubles the current distance.

### • Intersecting planes

We now consider two intersecting planes, represented by two rectangles (vertices 1, 2, 3, 4 and 11, 12, 13, 14). The intersection line is defined by the points 5 and 6. All the points are defined only once and the parameters of the projection are the same for the three figures. The corresponding part of the program is:

```
% 1. Definition of characteristic points
\figinit{3cm, realistic}
\figpt 1:(0,0)
\figpttraC 2:=1/0,2,0/
\figpttraC 3:=2/1.2,0,0/
\figpttraC 4:=3/0,-2,0/
\figpt 5:(0,0.9)\figpt 6:(1.2,1.3)
\figvectC 10(1,0,0)\figptsrot 11 = 1,2 /5,60,10/
\figvectP 10[5,6]\figpttra 13:=12/1,10/\figpttra 14:=11/1,10/
\def\Valpsi{-30}\def\Valtheta{30}\figset proj(psi=\Valpsi, theta=\Valtheta)
```

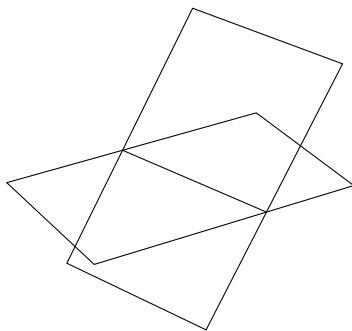


Figure 43

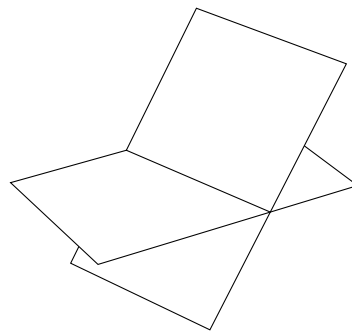


Figure 44

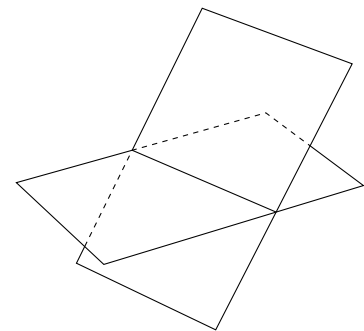


Figure 45

The left-hand side figure is created by the sections 2a and 3a of the program given below. It provides a straightforward representation, which looks somewhat ambiguous because the hidden parts of the scene are not handled.

```
% 2a. Creation of the first postscript file
\NewPSfile{\MyPSfile}
\psbeginfig{\MyPSfile}
\psline[1,2,3,4,1]\psline[11,12,13,14,11]\psline[5,6]
\psendfig
% 3a. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}
{\figinsert{\MyPSfile}}
```

A solution to this is proposed on the figure 44 (created by the sections 2b and 3b of the program) where the planes are shown using three filled areas drawn from the most distant to the nearest. Notice that this algorithm holds as long as the observation angle  $\theta$  is positive. With this kind of representation, we could of course choose the color of each plane. Here, since they are both white, a black border is necessary, which is drawn at the same time without forgetting to switch to the correct filling mode.

```

% 2b. Creation of the second postscript file
\NewPSfile{\MyPSfile}
\psbeginfig{\MyPSfile}
\def\white{1}\def\black{0}
\psset(fillmode=yes,color=\white)\psline[11,14,6,5,11]
\psset(fillmode=no,color=\black)\psline[11,14,6,5,11]
\psset(fillmode=yes,color=\white)\psline[1,2,3,4,1]
\psset(fillmode=no,color=\black)\psline[1,2,3,4,1]
\psset(fillmode=yes,color=\white)\psline[5,6,13,12,5]
\psset(fillmode=no,color=\black)\psline[5,6,13,12,5]
\psendfig
% 3b. Writing text on the figure
\figvisu{\demotwo}{\bf Figure \the\Figno}
{\figinsert{\MyPSfile}}

```

Another solution is to represent the hidden lines as dashed lines. We stress the fact that their limits depend on the observation direction and may lead to unexpected results if the values of  $\psi$  and  $\theta$  are far from the current values used here while keeping the same points to define the crossing segments and lines. The corresponding drawing is shown on the right-hand side figure, created by the sections 2c and 3c of the program:

```

% 2c. Creation of the third postscript file
\NewPSfile{\MyPSfile}
\psbeginfig{\MyPSfile}
\figptvisilimSL 7:[2,3 ; 6,13]
\figptvisilimSL 15:[5,11 ; 1,4]
\psline[5,1,4,3,7]\psline[5,12,13,14,11,15]\psline[5,6]
\psset(dash=4)\psline[15,5,2,7]
\psendfig
% 3c. Writing text on the figure
\figvisu{\demothree}{\bf Figure \the\Figno}
{\figinsert{\MyPSfile}}

```

The three figures are then displayed by saying:

```

\centerline{\box\demo\hfil\box\demotwo\hfil\box\demothree}

```

## • Fun

Here is a recreative example. It shows a movement, called “jibe”, made by a funboard turning around a floating mark. This will be the opportunity to illustrate how a jointed object can be handled.

The funboard is made of two main rigid parts: the board, bearing the fin (macro `\board`) and the rigging set: sail, mast and wishbone (macro `\sail`). The bottom point of the mast is attached to the middle part of the board according to a spherical link, allowing the sail to rotate in any direction over the board.

Three orthogonal axes are linked to the board (macro `\axes`). Thus, the geometric description is entirely done with respect to the basis  $(\vec{u}_1, \vec{u}_2, \vec{u}_3)$  and the projection  $P$  of the prow point of the board onto the trajectory along the  $\vec{u}_3$  direction. This also gives the sail a neutral position in the plane  $(P; \vec{u}_1, \vec{u}_3)$ . The vector  $\vec{u}_1$  is oriented from the front to the rear and the vector  $\vec{u}_3$  from the bottom to the top. In the program, the vectors  $\vec{u}_1$ ,  $\vec{u}_2$  and  $\vec{u}_3$  bear respectively the numbers 11, 12 and 13, and the prow point  $P$  number 0.

In its neutral position, the funboard is symmetric with respect to the plane  $(P; \vec{u}_1, \vec{u}_3)$ . To give the funboard the right position, we just have to translate the point  $P$  and rotate the basis (macro `\boardposition`). Then, the position of the sail is tuned by rotating the basis again (macro `\sailposition`). The corresponding calls are made in the macro `\funboard`.

The funboard is shown in three characteristic positions along its trajectory which is represented here as a Bézier curve lying in the plane  $z = 0$ . To obtain this result, the point  $P$  occupies three positions on the trajectory at the abscissae 0.2, 0.55 and 0.95. At each of these positions,  $\vec{u}_1$  is made colinear with the tangent vector to the curve (macro `\jibe`).

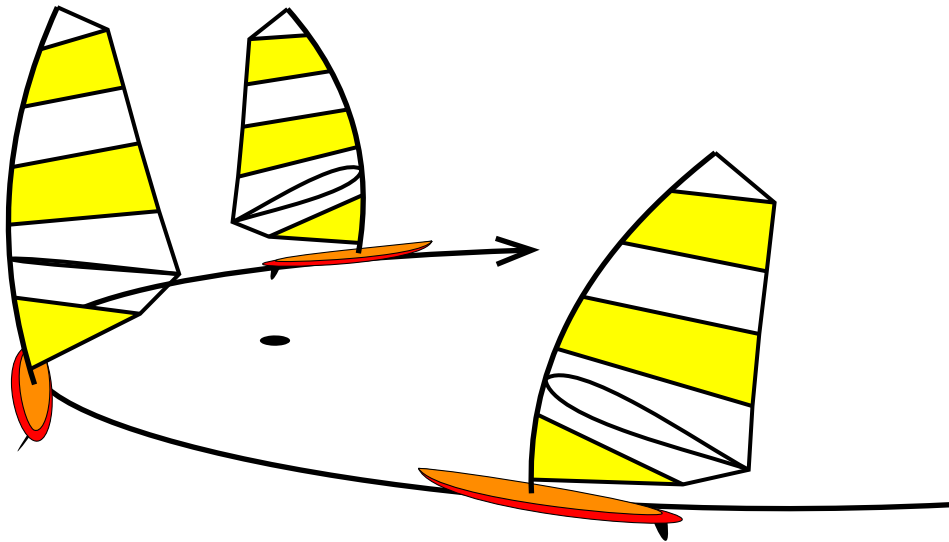


Figure 46  $(\psi, \theta) = (-35, 20)$

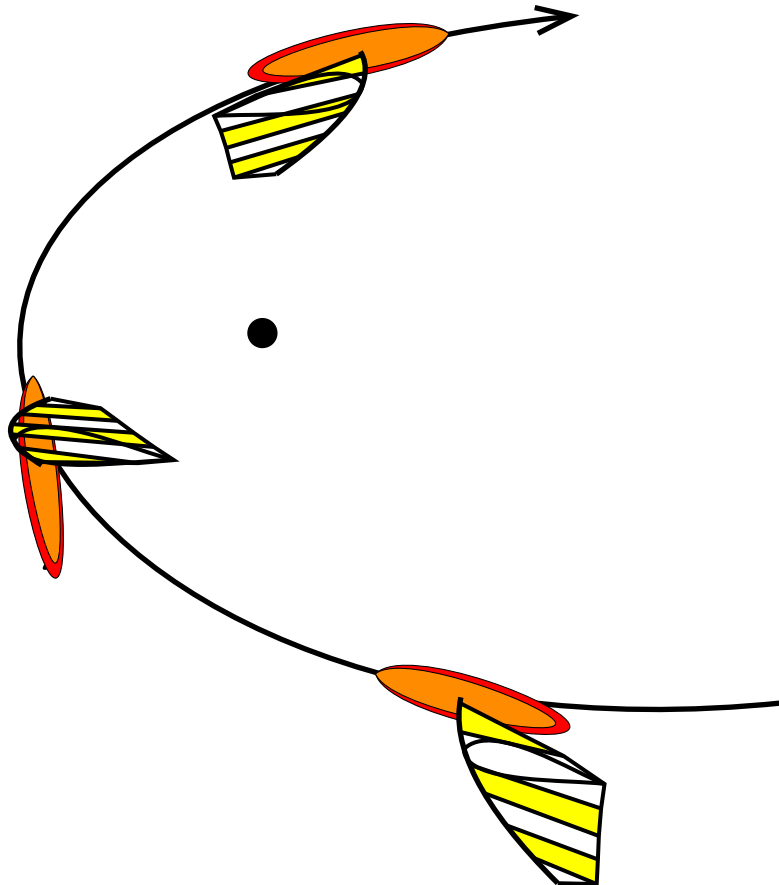


Figure 47  $(\psi, \theta) = (-35, 90)$



The only data are the four control points of the trajectory (points number 15, 16, 17 and 18). The floating mark (point 19) is the center of curvature computed at the abscissa  $t = 0.5$ . There is no good reason for that except that this shows how to use the macro `\figptcurvcenter`. These points are the only ones that keep their values all along the process. The trajectory and the floating mark are drawn by the macro `\trajectory`.

Two angles of observation are chosen ; the second one is a view from the top. The numerical values are given in meters in true dimensions. The scale of the model is  $\frac{1}{100}$  since the first argument of `\figinit` is cm. Notice the use the macro `\figvectU` that gives the tangent vector the right length. The target point is set to the floating mark and the observation distance is set to 20 m (from this point). Since the eye point is rather close to the scenery, we can observe the depth effect on the first figure: the funboard at the first position is on the foreground and is bigger than the other ones.

The program that produces the two figures is given below. Each task corresponds to a macro which is self explanatory ; the main program at the end is then very short.

```
%-----
% Definition of macros
%-----
\def\board{% Draws the board
% Main points
\figpttra 1:=0/0.22,13/
\figpttra 2:=0/0.3,11/\figpttra 3:=0/2.7,11/\figpttra 4:=0/2.7,11/
\figpttra 2:=2/0.5,12/\figpttra 3:=3/0.3,12/
\figpttra 5:=0/0.3,11/\figpttra 6:=0/2.5,11/\figpttra 7:=0/2.5,11/
\figpttra 5:=5/0.4,12/\figpttra 6:=6/0.2,12/
\figpttra 5:=5/0.07,13/\figpttra 6:=6/0.06,13/\figpttra 7:=7/0.04,13/
% Bottom point of the mast
\figptBezier 8::0.45[1,5,6,7]\figptorthoprojplane 8:=8/1,12/
% Define the fin and draw it
\figpttra 20:=4/-0.30,11/\figpttra 21:=4/-0.25,11/
\figpttra 22:=4/-0.05,11/\figpttra 23:=4/-0.15,11/
\figpttra 21:=21/-0.15,13/\figpttra 22:=22/-0.5,13/
\psset(fillmode=yes,color=\Blackrgb)\psBezier 1[20,21,22,23]
% Compute the symmetric part of the board and draw it
\figptssym 20=2,3/1,12/
\psset(width=\defaultwidth,color=\Redrgb)\psBezier 2[1,2,3,4,21,20,1]
\psset(fillmode=no,color=\Blackrgb)\psBezier 2[1,2,3,4,21,20,1]
\figptssym 20=5,6/1,12/
\psset(fillmode=yes,color=\DarkOrangergb)\psBezier 2[1,5,6,7,21,20,1]
\psset(fillmode=no,color=\Blackrgb)\psBezier 2[1,5,6,7,21,20,1]}
%-----
\def\sail{% Draws the rigging set
% Mast
\figpttra 21:=8/1.5,13/\figpttra 22:=8/3,13/\figpttra 23:=8/4.5,13/
\figpttra 21:=21/-0.6,11/\figpttra 22:=22/-0.6,11/
% Sail
\figptBezier 40::0.04[8,21,22,23]
\figptBezier 31::0.230[8,21,22,23]\figpttra 41:=31/1.56,11/\figptrot41:=41/31,2,12/
\figptBezier 32::0.333[8,21,22,23]\figpttra 42:=32/2.08,11/\figptrot42:=42/32,0,12/
\figptBezier 33::0.423[8,21,22,23]\figpttra 43:=33/1.85,11/\figptrot43:=43/33,-10,12/
\figptBezier 34::0.576[8,21,22,23]\figpttra 44:=34/1.58,11/\figptrot44:=44/34,-15,12/
\figptBezier 35::0.736[8,21,22,23]\figpttra 45:=35/1.25,11/\figptrot45:=45/35,-15,12/
\figptBezier 36::0.888[8,21,22,23]\figpttra 46:=36/0.85,11/\figptrot46:=46/36,-20,12/
```

```

% Wishbone
\figpttra 37:=32/0.2,11/\figpttra 38:=32/1.6,11/
\figpttra 37:=37/0.5,12/\figpttra 38:=38/0.05,12/
% Draw the sail
\psset(fillmode=yes,color=\Yellowrgb)
\psline[40,31,41]\psline[33,34,44,43]\psline[35,36,46,45]
\psset(fillmode=no,width=1.5,color=\Blackrgb)
\psline[31,41]\psline[33,43]\psline[34,44]\psline[35,45]\psline[36,46]
\psline[40,41,42,43,44,45,46,23]
% Draw the wishbone
\psBezier 1[32,37,38,42]\figptssym 37=37,38/8,12/\psBezier 1[32,37,38,42]
% Draw the mast
\psset(width=2)\psBezier 1[8,21,22,23]}
%-----
\def\funboard{\boardposition\board\sailposition\sail}
%-----
\def\axes{% Defines the local axes linked to the board
\figptBezier 0:$P$:\abscissa[15,16,17,18]
\figvectDBezier 21:1,\abscissa[15,16,17,18]\figvectU 21[21]\figpttra 1:=0/-1,21/
\figvectP 11[0,1]\figvectC 13(0,0,1)\figptrot 2:=1/0,90,13/\figvectP 12[0,2]}
%-----
\def\jibe{% Draws the funboard at 3 successive positions
% First position
\def\boardposition{\def\abscissa{0.2}\axes}
\def\sailposition{\figpt 20:(0,0)\figptrot 11:=11/20,20,12/\figptrot 13:=13/20,20,12/
\figptsrot 12=12,13/20,20,11/}
\funboard
% Second position
\def\boardposition{\def\abscissa{0.55}\axes\figpt20:(0,0)\figptsrot12=12,13/20,-40,11/}
\def\sailposition{\figpt 20:(0,0)\figptsrot 11=11,12/20,70,13/
\figptrot 11:=11/20,-35,12/\figptrot 13:=13/20,-35,12/}
\funboard
% Third position
\def\boardposition{\def\abscissa{0.95}\axes}
\def\sailposition{\figpt 20:(0,0)\figptrot 11:=11/20,20,12/\figptrot 13:=13/20,20,12/
\figptsrot 12=12,13/20,-20,11/}
\funboard}
%-----
\def\trajectory{% Draws the trajectory and the floating mark
\psset(width=2)\psset arrowhead(length=0.6)\psarrowBezier [15,16,17,18]
\figpttraC 1:=19/1,0,0/\figpttraC 2:=19/0,1,0/
\psset(fillmode=yes,color=\Blackrgb)\pscirc 19,1,2(0.2)}
%-----
\def\newfig{% Creates a new postscript file and a new figure
% 2. Creation of the postscript file
\NewPSfile{\MyPSfile}
\psbeginfig{\MyPSfile}
\trajectory\jibe
\psendfig
% 3. Writing text on the figure
\figvisu{\demo}{\bf Figure \the\Figno}\$$(\psi, \theta)=(\Valpsi, \Valtheta)$}
{\figinsert{\MyPSfile}}
\vskip1.5cm
\centerline{\box\demo}}

```

```

%-----
% Program
%-----
\figinit{cm, realistic}
% 1. Trajectory control points
\figpt 15:(7,3)\figpt 16:(2,-6)\figpt 17:(-8,-5)\figpt 18:(-2,6)
% Floating mark position
\figptcurvcenter 19::0.5[15,16,17,18]
% First figure
\figset proj(targetpt=19, distance=20)
\def\Valpsi{-35}\def\Valtheta{20}\figset proj(psi=\Valpsi, theta=\Valtheta)
\newfig
% Second figure
\def\Valtheta{90}\figset proj(theta=\Valtheta, distance=50)
\newfig

```

Remark: In the macros `\boardposition` and `\sailposition`, in order to rotate the basis  $(\vec{u}_1, \vec{u}_2, \vec{u}_3)$ , we have used the macro `\figptrot`, which acts on a point, to rotate a vector, after having the rotation axis through the origin. This has the advantage that the result is obtained in a straightforward way. However, this is not mathematically consistent since the object created is a point and not a vector. In this macro package, the only consequence would be on the macro `\figshowpts` which show the points but not the vectors. If we are purist, we can generate vectors by creating intermediate points. For example, in the first call to `\sailposition`, we can replace

```
\figpt 20:(0,0)\figptrot 11:=11/20,20,12/
```

by

```
\figpt 20:(0,0)\figpttra 21:=20/1,11/\figptrot 21:=21/20,20,12/\figvectP 11[20,21]
```

## §11. Using this macro-package

This macro-package is ready to be used with plain T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X under any Unix system (including Linux and Mac OS X) and Windows. With TeXtures on a MacIntosh running MacOS 9 or less, the macro file `fig4tex.tex` must be modified as explained in its header, in order to handle the postscript files correctly.

Remark concerning L<sup>A</sup>T<sub>E</sub>X:

If a *french* package is used, then the macro file `fig4tex.tex` must be loaded *after* the statement `\begin{document}`. In case of trouble with another package, please try this as a first workaround. This gives the following structure:

```
\documentclass[12pt]{article} % for example
\usepackage{french}          % or \usepackage[french]{babel}

\begin{document}
\input{fig4tex.tex}
.
.
.
\end{document}
```

### Bug report

The authors are interested to receive back any information about trouble in using this macro-package. Please mail your message to [Yvon.Lafranche@univ-rennes1.fr](mailto:Yvon.Lafranche@univ-rennes1.fr) or [Daniel.Martin@univ-rennes1.fr](mailto:Daniel.Martin@univ-rennes1.fr).

## §12. List of user macros

An HTML version of this list is available at:

<http://perso.univ-rennes1.fr/yvon.lafranche/fig4tex/>

Geometrical macros - Mute macros

-----

----- Control macros

`\figinit{ScaleFactorUnit}` or `\figinit{ScaleFactorUnit, 2D}`

`\figinit{ScaleFactorUnit, X}` with X in {3D, cavalier, orthogonal, realistic}

Initialization before the creation of a new figure. It is necessary to call this macro in case of successive figures. No check is performed on the arguments.

First argument : Choice of the unit and the scale factor.

The unit can be one those defined in the TeX Book, namely: pt (TeX point), pc (pica, 1pc = 12pt), in (inch, 1in = 72.27pt), bp (big point, 72bp = 1in), cm (centimeter, 2.54cm = 1in), mm (millimeter, 10mm = 1cm), dd (point didot, 1157dd = 1238pt), cc (cicero, 1cc = 12dd), sp (scaled point, 65536sp = 1pt)

By default, pt is assumed and the scale factor is 1. For example, `\figinit{in}` is equivalent to `\figinit{2.54cm}`.

Second argument (optional) : Choice of the space dimension and, in 3D, the projection type.

If this argument is equal to 2D or absent, then geometry in the plane is assumed, otherwise geometry in three dimensions is performed.

Moreover, if this argument is equal to:

- . orthogonal, then the orthogonal projection is used,
- . realistic, then the realistic projection is used,
- . 3D or cavalier (or anything else), then the cavalier projection is used.

`\figinsert{FileName}` or `\figinsert{FileName, ScaleFactor}`

Insertion of the postscript file FileName in the page, scaled by ScaleFactor which by default equals to 1.

`\figscan FileName(HX,HY)`

Draws a rectangular grid with horizontal and vertical steps HX and HY, with numerical values corresponding to the bounding box read in the file FileName.

`\figset keyword (attribute1=value1, attribute2=value2,...)`

Setting parameters acting on the appearance of the figure, which are to be given as pairs of the form attribute = value. For each keyword, the attributes are given below, along with the possibles values, their default value and their definition:

- keyword = projection

- . depth (or lambda) = real in [0,1] (default = 0.5): depth reduction coefficient (for the cavalier projection).
- . distance = real in user coordinates (default = 5 x largest dimension of the 3D bounding box): observation distance (for the realistic projection).
- . latitude (or theta) = angle in degrees (default = 25): latitude of the observation direction.
- . longitude (or psi) = angle in degrees (default = 40): longitude of the observation direction.
- . targetpt = point number (default = center of the 3D bounding box): target point (for the realistic projection).

Nota: These attributes must be set before step 2 (`\psbeginfig`) and 3 (`\figvisu`).

The cavalier projection is only defined by depth and psi. In this case, values of psi lying in [0,180] correspond to a view from above and to a view from beneath for values lying in [-180,0].

The orthogonal projection is only defined by theta and psi.

The realistic projection is defined by theta, psi, distance and targetpt.

For those two projections, the observation direction is defined by the longitude  $\psi$  and the latitude  $\theta$ , with  $(\psi, \theta) = (0, 0)$  corresponding to the  $Ox$  line.

#### `\figshowsettings`

Prints to the terminal the current settings.

#### `\figvisu{Vbox}{Caption}{Commands}`

Creation of a Vbox containing the figure and the legends defined by the Commands, with a Caption centered below. The box must be previously allocated by the user. Commands and Caption can be void.

----- Basic macros

#### `\figpt NewPt :Text(X,Y)`

#### `\figpt NewPt :Text(X,Y,Z)`

Definition of the point NewPt whose coordinates are (X,Y) or (X,Y,Z), with a joined Text. If Z is missing, then Z=0 is assumed.

The default Text is  $A_i$  where  $i$  is the number of the point (see `\figsetptname`).

#### `\figptbary NewPt :Text[Pt1,... ,PtN ; Coef1,... ,CoefN]`

Barycenter or centroid (with a joined Text) of N points bearing integer coefficients

#### `\figptbaryR NewPt :Text[Pt1,... ,PtN ; Coef1,... ,CoefN]`

Barycenter or centroid (with a joined Text) of N points bearing real coefficients

#### `\figptcopy NewPt :Text/Pt/`

Definition of NewPt, with an associated Text, with the same coordinates as Pt.

#### `\figgetangle \Value [Center,Pt1,Pt2]`

#### `\figgetangle \Value [Center,Pt1,Pt2,Pt3]`

Computation of the value  $\backslash\text{Value}$  (in degrees) of the oriented angle (CP1, CP2), where  $C=\text{Center}$ ,  $P1=\text{Pt1}$ ,  $P2=\text{Pt2}$ .  $\backslash\text{Value}$  is a macro whose name is chosen by the user and can then be used as a symbolic numerical constant.

In 3D, the plane is oriented so that the angle is measured counterclockwise around the vector  $CP1 \times CP3$ , where  $P3=\text{Pt3}$ . Notice that C, P1, P2 and P3 must be coplanar, but P3 must not lie on the line (C,P1).

#### `\figgetdist\Value[Pt1,Pt2]`

Computation of the euclidian distance  $\backslash\text{Value}$  between the points Pt1 and Pt2.

$\backslash\text{Value}$  is a macro whose name is chosen by the user and can then be used as a symbolic numerical constant.

#### `\figvectC NewVect (X,Y)`

#### `\figvectC NewVect (X,Y,Z)`

Definition of the vector NewVect with components (X,Y) or (X,Y,Z).

If Z is missing, then Z=0 is assumed.

#### `\figvectN NewVect [Pt1,Pt2]`

#### `\figvectN NewVect [Pt1,Pt2,Pt3]`

Definition of the vector NewVect which is:

- . in 2D, the vector with origin point Pt1 and end point Pt2 rotated by  $\pi/2$  (so normal to [Pt1,Pt2]),

- . in 3D, the vector product  $P1P2 / ||P1P2|| \times P1P3 / ||P1P3||$ , ie a vector normal to the plane defined by the 3 points and so that (P1P2, P1P3, NewVect) is a positively oriented basis.

#### `\figvectNV NewVect [Vector]`

#### `\figvectNV NewVect [Vector1, Vector2]`

Definition of the vector NewVect which is:

- . in 2D, the vector Vector rotated by  $\pi/2$

- . in 3D, the vector product  $\text{Vector1} / ||\text{Vector1}|| \times \text{Vector2} / ||\text{Vector2}||$  so that (Vector1, Vector2, NewVect) is a positively oriented basis.

#### `\figvectP NewVect [Pt1,Pt2]`

Definition of the vector NewVect with origin point Pt1 and end point Pt2.

```

\figvectU NewVect [Vector]
  Definition of the unitary vector NewVect which is Vector normalized according to
  the unit and the scale factor chosen by the user (see \figinit).

----- Macros for elementary geometry

--- Transformation macros (one result) ---
\figtmap NewPt :Text= Pt /InvPt/A11,A12 ; A21,A22/
\figtmap NewPt :Text= Pt /InvPt/A11,A12,A13 ; A21,A22,A23 ; A31,A32,A33/
  Image NewPt (with a joined Text) of point Pt by the mapping defined by:
  . an invariant point InvPt
  . the matrix A whose coefficients are Aij, i,j in [1,d], d=2 in 2D or d=3 in 3D.
  If I=InvPt, P=Pt, P'=NewPt, P' is computed as  $IP' = A (IP)$ .
\figpthom NewPt :Text= Pt /Center, Ratio/
  Image NewPt of point Pt by the homothety of center Center and of ratio Ratio
  with a joined Text
\figtrot NewPt :Text= Pt /Center, Angle/
\figtrot NewPt :Text= Pt /Center, Angle, Vector/
  Image NewPt (with a joined Text) of point Pt by the rotation defined:
  . in 2D, by the center Center and the angle Angle (in degrees),
  . in 3D, by the axis (Center, Vector) and the angle Angle (in degrees).
  Pt is rotated by Angle around the axis. The sense of rotation is such that
  (CP, CP', Vector) is a positively oriented basis, where C=Center, P=Pt, P'=NewPt.
\figtsym NewPt :Text= Pt /LinePt1, LinePt2/
\figtsym NewPt :Text= Pt /PlanePt, NormalVector/
  Image NewPt (with a joined Text) of point Pt by the orthogonal symmetry
  with respect to:
  . in 2D, the line defined by the points LinePt1 and LinePt2,
  . in 3D, the plane defined by the point PlanePt and the vector NormalVector normal
  to the plane.
\figttra NewPt :Text= Pt /Lambda, Vector/
  Image NewPt of point Pt by the translation of vector Lambda * Vector
  with a joined Text.
\figttraC NewPt :Text= Pt /X,Y/
\figttraC NewPt :Text= Pt /X,Y,Z/
  Image NewPt of point Pt by the translation of vector (X,Y) in 2D, (X,Y,Z) in 3D
  with a joined Text.
\figtorthoprojline NewPt :Text= Pt /LinePt1, LinePt2/
  Image NewPt of point Pt by the orthogonal projection onto the line (LinePt1,LinePt2),
  with a joined Text.
\figtorthoprojplane NewPt :Text= Pt /PlanePt, NormalVector/
  Image NewPt of point Pt by the orthogonal projection onto the plane defined by
  the point PlanePt and the normal vector NormalVector, with a joined Text.

--- Transformation macros (multiple result) ---
\figtsmap NewPt1 = Pt1, Pt2, ..., PtN /InvPt/A11,A12 ; A21,A22/
\figtsmap NewPt1 = Pt1, Pt2, ..., PtN /InvPt/A11,A12,A13; A21,A22,A23; A31,A32,A33/
  Images NewPt1, NewPt1+1, ..., NewPt1+(N-1) of points Pt1, Pt2, ..., PtN
  by the mapping defined by:
  . an invariant point InvPt
  . the matrix A whose coefficients are Aij, i,j in [1,d], d=2 in 2D or d=3 in 3D.
  If I=InvPt, P=Pti, P'=NewPti, P' is computed as  $IP' = A (IP)$ .
  See following information at \figtstra.

```

`\figptshom`  $\text{NewPt1} = \text{Pt1}, \text{Pt2}, \dots, \text{PtN}$  /Center, Ratio/  
 Images  $\text{NewPt1}, \text{NewPt1+1}, \dots, \text{NewPt1+(N-1)}$  of points  $\text{Pt1}, \text{Pt2}, \dots, \text{PtN}$   
 by the homothety of center `Center` and of ratio `Ratio`.  
 See following information at `\figptstra`.

`\figptsrot`  $\text{NewPt1} = \text{Pt1}, \text{Pt2}, \dots, \text{PtN}$  /Center, Angle/  
`\figptsrot`  $\text{NewPt1} = \text{Pt1}, \text{Pt2}, \dots, \text{PtN}$  /Center, Angle, Vector/  
 Images  $\text{NewPt1}, \text{NewPt1+1}, \dots, \text{NewPt1+(N-1)}$  of points  $\text{Pt1}, \text{Pt2}, \dots, \text{PtN}$   
 by the rotation defined:  
 . in 2D, by the center `Center` and the angle `Angle` (in degrees),  
 . in 3D, by the axis (`Center, Vector`) and the angle `Angle` (in degrees). Each point  
 $\text{Pti}$  is rotated by `Angle` around the axis. The sense of rotation is such that  
 (`CP, CP', Vector`) is a positively oriented basis, where  $C=\text{Center}$ ,  $P=\text{Pti}$ ,  $P'=\text{NewPti}$ .  
 See following information at `\figptstra`.

`\figptssym`  $\text{NewPt1} = \text{Pt1}, \text{Pt2}, \dots, \text{PtN}$  /LinePt1, LinePt2/  
`\figptssym`  $\text{NewPt1} = \text{Pt1}, \text{Pt2}, \dots, \text{PtN}$  /PlanePt, NormalVector/  
 Images  $\text{NewPt1}, \text{NewPt1+1}, \dots, \text{NewPt1+(N-1)}$  of points  $\text{Pt1}, \text{Pt2}, \dots, \text{PtN}$   
 by the orthogonal symmetry with respect to:  
 . in 2D, the line defined by the points `LinePt1` and `LinePt2`,  
 . in 3D, the plane defined by the point `PlanePt` and the vector `NormalVector` normal  
 to the plane. See following information at `\figptstra`.

`\figptstra`  $\text{NewPt1} = \text{Pt1}, \text{Pt2}, \dots, \text{PtN}$  /Lambda, Vector/  
 Images  $\text{NewPt1}, \text{NewPt1+1}, \dots, \text{NewPt1+(N-1)}$  of points  $\text{Pt1}, \text{Pt2}, \dots, \text{PtN}$   
 by the translation of vector `Lambda * Vector`  
 Text eventually previously associated with the result points is lost.  
 The result points  $\text{NewPti}$  have successive numbers.  
 The data points  $\text{Pti}$  can be given in any order.  
 WARNING : Let  $I_r$  (resp.  $I_d$ ) the set of the result points numbers (resp. the set  
 of the given points numbers). Let  $J$  the intersection of  $I_r$  and  $I_d$ .  
 1)  $I_r = I_d$ , or  $J$  is empty, or  $\text{NewPt1}$  does not belong to  $J$ : no problem.  
 2) Otherwise, the result given by the macro MAY BE WRONG, at least partially:  
 this is because, in this case,  $\text{NewPt1}$  belongs to  $J$ , and the given points are  
 taken into account sequentially, beginning from the first element in the list.

`\figptsorthoprojline`  $\text{NewPt1} = \text{Pt1}, \text{Pt2}, \dots, \text{PtN}$  /LinePt1, LinePt2/  
 Images  $\text{NewPt1}, \text{NewPt1+1}, \dots, \text{NewPt1+(N-1)}$  of points  $\text{Pt1}, \text{Pt2}, \dots, \text{PtN}$   
 by the orthogonal projection onto the line (`LinePt1, LinePt2`).

`\figptsorthoprojplane`  $\text{NewPt1} = \text{Pt1}, \text{Pt2}, \dots, \text{PtN}$  /PlanePt, NormalVector/  
 Images  $\text{NewPt1}, \text{NewPt1+1}, \dots, \text{NewPt1+(N-1)}$  of points  $\text{Pt1}, \text{Pt2}, \dots, \text{PtN}$   
 by the orthogonal projection onto the plane defined by the point `PlanePt` and the  
 normal vector `NormalVector`.

--- Geometrical construction macros ---

`\figptinterlines` `NewPt :Text[LinePt1,Vector1; LinePt2,Vector2]`  
 Intersection of the line defined by the point `LinePt1` and the vector `Vector1`  
 and the line defined by the point `LinePt2` and the vector `Vector2`  
 with a joined `Text`

`\figptinterlineplane` `NewPt :Text[LinePt,Vector; PlanePt,NormalVector]`  
 Intersection of the line defined by the point `LinePt` and the vector `Vector`  
 and the plane defined by the point `PlanePt` and the normal vector `NormalVector`  
 with a joined `Text`.

`\figptsaxes` `NewPt1 : Origin(X1,X2, Y1,Y2)`  
`\figptsaxes` `NewPt1 : Origin(X1,X2, Y1,Y2, Z1,Z2)`  
`\figptsaxes` `NewPt1 : Origin(L)`  
 End points  $\text{NewPt1}, \text{NewPt1+1}$  (and  $\text{NewPt1+2}$  in 3D) of the arrows corresponding to  
 the axes, drawn by the macro `\psaxes`. The text  $\$x\$, \$y\%$  (and  $\$z\%$  in 3D) is



automatically joined to the points, so that writing the legend with the `\figwriteX` macros is more straightforward. For example, in 2D, one can write something like: `\figwrites NewPt1:(3pt) \figwritew NewPt1+1:(3pt)`

Remember that these macros can override the default text setting.

The short form `\figptsaxes NewPt1 : Origin(L)` is equivalent to

`\figptsaxes NewPt1 : Origin(0,L, 0,L)` in 2D and

`\figptsaxes NewPt1 : Origin(0,L, 0,L, 0,L)` in 3D.

NOTA: For simplicity of use, the arguments of this macro are deduced from those of `\psaxes`.

`\figptendnormal NewPt :Text: Length,Lambda [Pt1,Pt2]`

End point (with a joined Text) of the "exterior normal" to the segment [Pt1, Pt2].

The length of the normal vector is Length. Lambda is the barycentric coordinate of the origin of the normal with respect to the segment [Pt1, Pt2], which sets the position of the vector along [Pt1, Pt2]. In 3D-mode, it works only in the plane Z=0.

`\figptsintercirc NewPt1 [Center1,Radius1 ; Center2,Radius2]`

`\figptsintercirc NewPt1 [Center1,Radius1 ; Center2,Radius2 ; PlanePt]`

Intersections NewPt1 and NewPt2 (=NewPt1+1) of the two circles defined by their center and radius, (Center1,Radius1) and (Center2,Radius2).

NewPt1 and NewPt2 must be different from Center1 and Center2.

NewPt1 and NewPt2 are ordered so that the angle (NewPt1, Center1, NewPt2) is positive.

If the two circles do not intersect, then NewPt1=Center1 and NewPt2=Center2.

In 3D, the plane containing the two circles is defined by the three points Center1,

Center2 and PlanePt (which must not lie on the same line). The three points

(NewPt1, Center1, NewPt2) defined the same orientation as (Center2, Center1, PlanePt).

`\figptsinterlinell NewPt1 [Center,XRad,YRad,Inclination ; LinePt1,LinePt2]`

Intersections NewPt1 and NewPt2 (=NewPt1+1) of the line (LinePt1, LinePt2) and the ellipse defined by Center, XRad, YRad and Inclination (see `\figptell`).

NewPt1 and NewPt2 must be different from LinePt1.

NewPt1 and NewPt2 are ordered so that the vectors NewPt1NewPt2 and LinePt1LinePt2 have the same direction.

If the line does not intersect the ellipse, then NewPt1=LinePt1 and NewPt2=LinePt2.

In 3D-mode, it works only in the plane Z=0.

`\figptsinterlinellP NewPt1 [Center,PtAxis1,PtAxis2 ; LinePt1,LinePt2]`

Intersections NewPt1 and NewPt2 (=NewPt1+1) of the line (LinePt1, LinePt2) and the ellipse defined by its Center, and the end points of its axes, A1=PtAxis1 and A2=PtAxis2. The local axes are then defined by the basis (CA1, CA2).

NewPt1 and NewPt2 must be different from LinePt1.

NewPt1 and NewPt2 are ordered so that the vectors NewPt1NewPt2 and LinePt1LinePt2 have the same direction.

If the line does not intersect the ellipse, then NewPt1=LinePt1 and NewPt2=LinePt2.

In 3D, the 5 data points must lie in the same plane ; this is not checked.

`\figptvisilimSL NewPt:Text [SegPt1,SegPt2 ; LinePt1,LinePt2]`

Apparent intersection NewPt (with a joined Text) of the segment [SegPt1,SegPt2] and the line (LinePt1,LinePt2). NewPt is the visible limit of the segment hidden by an object an edge of which is lying on the line.

If the projection of the line intersects the segment, the solution NewPt lies between SegPt1 and SegPt2, otherwise NewPt = SegPt1.

--- Macros related to the triangle ---

`\figptcircumcenter NewPt :Text [Pt1,Pt2,Pt3]`

Center NewPt of the circumscribed circle to the triangle (Pt1,Pt2,Pt3) with a joined Text.

`\figptexcenter NewPt :Text [Pt1,Pt2,Pt3]`

Center NewPt (with a joined Text) of the escribed circle to the triangle (Pt1,Pt2,Pt3) opposite to Pt1.

`\figptincenter` NewPt :Text [Pt1,Pt2,Pt3]  
Center NewPt (with a joined Text) of the inscribed circle to the triangle (Pt1,Pt2,Pt3).

`\figptorthocenter` NewPt :Text [Pt1,Pt2,Pt3]  
Orthocenter NewPt of the triangle (Pt1,Pt2,Pt3) with a joined Text.

----- Macros related to arcs and curves

`\figptcirc` NewPt :Text: Center;Radius (Angle)  
`\figptcirc` NewPt :Text: Center,Pt1,Pt2;Radius (Angle)  
Creation of the point NewPt, with an associated Text, on the circle defined by its Center, and its Radius. The position of the point is set by the Angle given in degrees. In 3D, the circle is lying in the plane (Center,Pt1,Pt2) = (C,P1,P2) and the Angle is measured counterclockwise around the vector CP1 x CP2, starting from the half-line (C,P1).

`\figptell` NewPt :Text: Center;XRad,YRad (Angle,Inclination)  
Creation of the point NewPt, with an associated Text, on the ellipse defined by Center, XRad, YRad and Inclination. Inclination is the rotation angle of the local axes with respect to the paper sheet. The position of the point is set by the parametrization Angle given in local axes. The two angles are given in degrees. In 3D-mode, it works only in the plane Z=0.

`\figptellP` NewPt :Text: Center,PtAxis1,PtAxis2 (Angle)  
Creation of the point NewPt, with an associated Text, on the ellipse defined by its Center, and the end points of its axes, A1=PtAxis1 and A2=PtAxis2. The local axes are then defined by the basis (CA1, CA2). The position of the point is set by the parametrization Angle given in degrees, starting from the half-line (C,A1) and measured counterclockwise around the vector CA1 x CA2.

`\figptBezier` NewPt :Text: t [Pt1,Pt2,Pt3,Pt4]  
Computes the point NewPt (with a joined Text) lying on the cubic Bezier curve defined by the four control points Pt1, Pt2, Pt3 and Pt4, for the parameter value t. We recall that if t=0, NewPt=Pt1 and if t=1, NewPt=Pt4.

`\figptscontrol` NewPt1 [Pt1,Pt2,Pt3,Pt4]  
Computes the two control points NewPt1 and NewPt2 (=NewPt1+1) so that the cubic Bezier curve defined by the control points Pt1, NewPt1, NewPt2 and Pt4 interpolates the four points Pt1, Pt2, Pt3 and Pt4 with respective parameter values of 0, 1/3, 2/3 and 1. The result points numbers can be anyone including Pt1, Pt2, Pt3 and Pt4.

`\figptscontrolcurve` NewPt1, \NbArcs [Pt0,Pt1,... ,PtN,PtN+1]  
Computes the control points NewPt1, NewPt1+1,..., NewPt1+M used by the macro `\pscurve` when it is invoked by `\pscurve` [Pt0,Pt1,... ,PtN,PtN+1] which draws a curve that consists in N-1 cubic Bezier arcs. `\NbArcs` is a TeX macro whose name is chosen by the user. It can be considered as a variable and on output, its value is N-1 so that the calling sequence `\psBezier` \NbArcs [NewPt1, NewPt1+1,..., NewPt1+M] draws exactly the same curve. As a consequence, the data points Pti are duplicated on output : indeed, we have NewPt1+J = Pti with J=3\*(i-1), i=1,...N. The result points numbers must be different from any of Pt0,Pt1,... ,PtN,PtN+1. On output, a total of M+1 points are created, with M = 3 \* \NbArcs. NOTA : this macro uses the current value of curve roudness.

`\figptcurvcenter` NewPt :Text: t [Pt1,Pt2,Pt3,Pt4]  
Curvature center NewPt (with a joined Text) at the point lying on the cubic Bezier curve defined by the four control points Pt1, Pt2, Pt3 and Pt4, for the parameter value t.

`\figvectDBezier` NewVect : n, t [Pt1,Pt2,Pt3,Pt4]  
Computes the vector NewVect corresponding to the derivative of order n of the cubic Bezier curve defined by the four control points Pt1, Pt2, Pt3 and Pt4, for the parameter value t. The order n must be equal to 1 or 2.

----- Control macros

`\figcoord{NDec}`  
 To write the coordinates of a point. To be used in the joined text argument of the macros that create points and the non-mute macros `\figwrit*`. Only NDec decimals are printed. See also `\figsetroundcoord`.

`\figsetmark{Mark}`  
 Definition of the point marker to be written, for example a point `(.)` or `$$\bullet$`. By default, nothing is written.

`\figsetptname{Name}`  
 Sets the new default name for the points created. The default name for the point  $i$  is  $A_i$ : `\figsetptname{X^{(#1)}}{}` changes it to  $X^{(i)}$ .

`\figsetroundcoord{yes/no}`  
 Switches on rounding of decimals printed with `\figcoord`.

`\figshowpts[Nmin, Nmax]`  
 Shows on the figure the location of every point defined since the beginning of the session, whose number lies in the interval  $[Nmin, Nmax]$ . Writes a bullet at each location point along with the number of the point or the joined text if any.  
 CAUTION :  
 If  $Nmax - Nmin$  is too large, an error message " ! TeX capacity exceeded" may occur.

----- Writing macros

`\figwrite`[Pt1, Pt2, ..., PtN]{Text}  
 Writing a Text after Pt1, Pt2, ..., PtN according to TeX's alignment.

`\figwritec`[Pt1, Pt2, ..., PtN]{Text}  
 Writing a Text vertically and horizontally centered at Pt1, Pt2, ..., PtN.

`\figwritep`[Pt1, Pt2, ..., PtN]  
 Writing the point marker at the locations defined by Pt1, Pt2, ..., PtN.

`\figwritew` Pt1, Pt2, ..., PtN :Text(Distance)  
`\figwritee` Pt1, Pt2, ..., PtN :Text(Distance)  
`\figwriten` Pt1, Pt2, ..., PtN :Text(Distance)  
`\figwrites` Pt1, Pt2, ..., PtN :Text(Distance)  
 Writing the point marker at the locations defined by Pt1, Pt2, ..., PtN, with a Text placed, with respect to each point, at a given Distance from each point towards the west, the east, the north or the south. Distance measures the shortest path between each Pti and the bounding box of the Text.

`\figwritenw` Pt1, Pt2, ..., PtN :Text(Distance)  
`\figwritesw` Pt1, Pt2, ..., PtN :Text(Distance)  
`\figwritene` Pt1, Pt2, ..., PtN :Text(Distance)  
`\figwritese` Pt1, Pt2, ..., PtN :Text(Distance)  
 Writing the point marker at the locations defined by Pt1, Pt2, ..., PtN, with a Text placed, with respect to each point, at a given Distance from each point towards the north-west, the south-west, the north-east or the south-east. Distance measures the shortest path between each Pti and the bounding box of the Text.

`\figwritebw` Pt1, Pt2, ..., PtN :Text(Distance)  
`\figwritebe` Pt1, Pt2, ..., PtN :Text(Distance)  
`\figwritebn` Pt1, Pt2, ..., PtN :Text(Distance)  
`\figwritebs` Pt1, Pt2, ..., PtN :Text(Distance)  
 Writing a point marker at the locations defined by Pt1, Pt2, ..., PtN, with a Text placed, with respect to each point, at a given Distance from each point towards

the west, the east, the north or the south (BASELINE version).

If X=w or e, Distance measures the length between Pti and the end (resp. the beginning) of the Text, while the baseline of the Text is set to Pti's ordinate.

If X=n or s, the Text is horizontally centered and Distance measures the length between Pti and the baseline of the Text.

`\figwritegcw Pt1, Pt2, ..., PtN :Text(DistanceX,DistanceY)`

`\figwritege Pt1, Pt2, ..., PtN :Text(DistanceX,DistanceY)`

Writing the point marker at the locations defined by Pt1, Pt2, ..., PtN, with a Text placed, with respect to each point, at a horizontal distance DistanceX (west or east) and a vertical distance DistanceY between the point and the mid-height of the text.

`\figwritegw Pt1, Pt2, ..., PtN :Text(DistanceX,DistanceY)`

`\figwritege Pt1, Pt2, ..., PtN :Text(DistanceX,DistanceY)`

Writing the point marker at the locations defined by Pt1, Pt2, ..., PtN, with a Text placed, with respect to each point, at a horizontal distance DistanceX (west or east) and a vertical distance DistanceY from the bottom of the bounding box of the Text if DistanceY > 0, from its top if DistanceY < 0. If DistanceY = 0, the Text is vertically centered.

#### Macros for postscript generation

-----

----- Control macros

`\psbeginfig{FileName}`

Starting the creation of a PostScript file whose name is FileName.

`\psendfig`

End of the current PostScript file.

`\psreset{keyword1, keyword2,...}`

To reset one or several groups of graphical attributes to their default values.

The groups of attributes are denoted with keywords which are:

- arrowhead to reset the arrow-head attributes,
- curve to reset the (\ps)curve attributes,
- first to reset the first-level (or primary) attributes,
- flowchart to reset the flow chart attributes,
- mesh to reset the (\ps)mesh attributes,
- second to reset the second-level (or secondary) attributes,
- third to reset the third-level (or ternary) attributes.

`\psset keyword (attribute1=value1, attribute2=value2,...)`

Setting the graphical attributes, which are to be given as pairs of the form attribute = value. For each keyword, the attributes are given below, along with the possible values, their default name in parentheses and their definition.

The current value can be obtained with the help of `\figshowsettings`.

- NO keyword or keyword = first

- . color = color definition (`\defaultcolor`): primary color in cmyk or rgb color code, or in gray tone (real number between 0 (black) and 1 (white)).
- . dash = index or dash pattern (`\defaultdash`): line style given by Index (Index = 1 (solid) to 10) or by Pattern.
- . fillmode = yes/no (`\defaultfill`): setting the filling mode to "no" tells the drawing macros to draw lines ; setting the filling mode to "yes" tells the macros to fill the appropriate area using the current color or gray shade.
- . join = miter (V), round (U) or bevel (\\_/) (`\defaultjoin`): line join parameter that establishes the shape to be put at the corners of a polygonal line. Best rendering is obtained with join=round when more than 2 segments meet at a corner.
- . update = yes/no (`\defaultupdate`): setting the update mode to "yes" (resp. to "no") before `\psbeginfig` enforces (resp. avoids) the postscript file to be recreated at

each compilation.

- . width = dimension in PostScript units (`\defaultwidth`): line width.
- Note : has no effect on a straight line after `\psset` (fillmode=yes).
- keyword = arrowhead
  - . angle = real in degrees (`\defaultarrowheadangle`): arrow-head opening half-angle.
  - . fillmode = yes/no (`\defaultarrowheadfill`): arrow-head filling switch, which tells whether the interior of the arrow-head must be filled or not.
  - . length = real in user coordinates (`\defaultarrowheadlength`): length of each edge of the arrow-head.
  - . out = yes/no (`\defaultarrowheadout`): arrow-head "outside" switch, which tells whether the arrow-head must be drawn outside the body of the arrow (as if it was rotated by 180 degrees around the end point) or not.
  - . ratio = real in [0,1] (`\defaultarrowheadratio`): arrow-head ratio defining the arrow-head length to be ratio x (body length).
- Nota : The two attributes length and ratio are mutually exclusive ; the default is to use the length.
- keyword = curve
  - . roundness = real usually in [0,0.5] (`\defaultroundness`): roundness of a C1 curve.
- keyword = flowchart
  - . arrowposition = real in [0,1] (`\defaultfcarrowposition`): arrow-head position along the path, 0 corresponding to the beginning and 1 to the end of the path.
  - . arrowrefpt = start/end (`\defaultfcarrowrefpt`): reference point of the arrow-head to its start point or to its end point ; the arrow-head position is computed with respect to this point.
  - . line = polygon/curve (`\defaultfcline`): tells `\psfconnect` to draw the path between two nodes using polygonal lines or smooth curved lines.
  - . padding = real in user coordinates (see `xpadding` and `ypadding`): space surrounding the text inside a frame. Both horizontal and vertical padding are set to the same value. This attribute allows to reduce or enlarge the frames drawn at the nodes of the flow chart, starting from the internal default dimensions.
  - . radius = positive real in user coordinates (`\defaultfcradius`): radius of the circular arcs to be drawn at the corners of the path when the line is polygonal.
  - . shape = rectangle, ellipse or lozenge (`\defaultfcshape`): shape of the frames.
  - . thickness = real in user coordinates (`\defaultfcthickness`): thickness of the frames.
  - . xpadding = real in user coordinates (`\defaultfcxpadding`): horizontal space around the text inside a frame.
  - . ypadding = real in user coordinates (`\defaultfcypadding`): vertical space around the text inside a frame.
- Notice that the frame color is set by the primary color, the thickness color is set by the secondary color and the background color of the frame is set by the ternary color.
- keyword = mesh
  - . diag = integer in {-1,0,1} (`\defaultmeshdiag`): controls the drawing of a grid mesh. If `diag=0`, each cell is empty ; if `diag=1` (resp. `diag=-1`), the SW-NE diagonal (resp. the NW-SE diagonal) is drawn inside each cell.
- keyword = second
  - . color = color definition (`\defaultsecondcolor`): secondary color (see primary one).
  - . dash = index or dash pattern (`\defaultseconddash`): secondary line style.
  - . width = dimension in PostScript units (`\defaultsecondwidth`): secondary line width.
- keyword = third
  - . color = color definition (`\defaultthirdcolor`): ternary color (see primary one).

`\pssetdefault` keyword (attribute1=value1, attribute2=value2,...)

Global setting of the default values of graphical attributes, which are to be given as pairs of the form attribute=NewDefaultValue.

The attributes are set up to the end of the document or up to next call to this macro.

For each keyword, the attributes are listed below ; see `\psset` for their definition (particular case : `\pssetdefault arrowhead(length=DimWithUnit)`).

- NO keyword or keyword = first: color, dash, fillmode, join, update, width,
- keyword = arrowhead: angle, fillmode, length, out, ratio,
- keyword = curve: roundness,
- keyword = flowchart: arrowposition, arrowrefpt, line, padding, shape, thickness, xpadding, ypadding,
- keyword = mesh: diag,
- keyword = second: color, dash, width,
- keyword = third: color.

----- Basic drawing macros

`\pscirc` Center (Radius)

`\pscirc` Center,Pt1,Pt2 (Radius)

Circle of center Center and of radius Radius.

In 3D, the circle is in the plane defined the 3 points Center, Pt1 and Pt2 ;

Pt1 and Pt2 do not need to lie on the circle.

`\psline`[Pt1,Pt2,... ,PtN]

Line defined by N points, closed if the last point number equals the first one.

`\pslineC`(X1 Y1, X2 Y2,..., XN YN)

`\pslineC`(X1 Y1 Z1, X2 Y2 Z2,..., XN YN ZN)

Line defined by N points, closed if the last point number equals the first one.

The points are given by their coordinates, each of them separated by a white space.

`\pslineF`{Filename}

Line defined by points read from the text file Filename, which contains the coordinates of the points, one point per line, given as X Y in 2D and as X Y Z in 3D (the values must be separated by a white space and nothing else).

The line is closed if the last point equals the first one.

----- Other drawing macros

--- Arc ---

`\psarccirc` Center ; Radius (Ang1,Ang2)

`\psarccirc` Center,Pt1,Pt2 ; Radius (Ang1,Ang2)

Circular arc of center Center and radius Radius limited by the angles Ang1 and Ang2 given in degrees.

In 2D, the angles are measured counterclockwise.

In 3D, the arc lies in the plane defined by the 3 points Center, Pt1 and Pt2.

The angles are measured counterclockwise around the vector  $CP_1 \times CP_2$ , starting from the half-line (C, P1), where C=Center, P1=Pt1, P2=Pt2.

`\psarccircP` Center ; Radius [Pt1,Pt2]

`\psarccircP` Center ; Radius [Pt1,Pt2,Pt3]

Point version of `\psarccirc`

Circular arc of center Center and of radius Radius limited by the two half-lines (Center, Pt1) and (Center, Pt2).

Let N be the normal vector that orients the plane in which lies the arc.

In 2D, N is defined as usual by  $N = Z = X \times Y$ .

In 3D,  $N = CP_1 \times CP_3$ , where C=Center, P1=Pt1, P3=Pt3.

The arc is drawn from Pt1 towards Pt2 turning counterclockwise around the axis (C; N).

Notice that C, P1, P2 and P3 must be coplanar, but P3 must not lie on the line (C,P1).

`\psarcell` Center ; XRad,YRad (Ang1,Ang2, Inclination)

Arc of ellipse of center Center, of axes XRad and YRad, limited by the parametrization angles Ang1 and Ang2. The major axis is turned by an angle Inclination from the horizontal line. The angles are given in degrees and measured counterclockwise.

In 3D-mode, it works only in the plane  $Z=0$ .

`\psarcellPA` Center,PtAxis1,PtAxis2 (Ang1, Ang2)  
 Arc of ellipse of center Center limited by the parametrization angles Ang1 and Ang2.  
 The end point of the major axis is PtAxis1 and the end point of the minor axis is PtAxis2.  
 The angles are given in degrees and measured counterclockwise around the vector CA1 x CA2.

`\psarcellPP` Center,PtAxis1,PtAxis2 [Pt1,Pt2]  
 Arc of ellipse of center Center limited by the two half-lines (Center,Pt1) and (Center,Pt2). The end point of the major axis is PtAxis1 and the end point of the minor axis is PtAxis2. The arc is drawn counterclockwise around the vector CA1 x CA2, where C=Center, A1=PtAxis1, A2=PtAxis2.

--- Arrow ---

`\psarrow` [Pt1,Pt2]  
 Arrow from Pt1 to Pt2.  
 The arrow-head is drawn according to the `\psarrowhead` macro settings.

`\psarrowBezier` [Pt1,Pt2,Pt3,Pt4]  
 Arrow that consists of the cubic Bezier curve defined by the four control points Pt1, Pt2, Pt3 and Pt4, and the arrow-head at point Pt4.

`\psarrowcirc` Center ; Radius (Ang1,Ang2)  
`\psarrowcirc` Center,Pt1,Pt2 ; Radius (Ang1,Ang2)  
 Circular arrow such that the circular arc is centered at Center, has the radius Radius and is limited by the angles Ang1 and Ang2 given in degrees.  
 If Ang2 > Ang1, the arrow is drawn counterclockwise, else it is drawn clockwise.  
 The arrow-head is drawn according to the `\psarrowhead` macro settings.  
 In 3D, the arc lies in the plane defined the 3 points Center, Pt1 and Pt2.  
 The angles are measured counterclockwise around the vector CP1 x CP2, starting from the half-line (C, P1), where C=Center, P1=Pt1, P2=Pt2.

`\psarrowcircP` Center ; Radius [Pt1,Pt2]  
`\psarrowcircP` Center ; Radius [Pt1,Pt2,Pt3]  
 Point version of `\psarrowcirc`  
 Circular arrow such that the circular arc is centered at Center, has the radius |Radius| and is limited by the two half-lines (Center, Pt1) and (Center, Pt2).  
 Let N be the normal vector that orients the plane in which lies the arc.  
 In 2D, N is defined as usual by  $N = Z = X \times Y$ .  
 In 3D,  $N = CP1 \times CP3$ , where C=Center, P1=Pt1, P3=Pt3.  
 The arrow is drawn from Pt1 towards Pt2 turning around the axis (Center; N):  
 . counterclockwise if Radius > 0,  
 . clockwise if Radius < 0.  
 Notice that C, P1, P2 and P3 must be coplanar, but P3 must not lie on the line (C,P1).

`\psarrowhead` [Pt1,Pt2]  
 Arrow-head of the arrow from Pt1 to Pt2. The segment [Pt1, Pt2] (body) is not drawn.  
 The appearance of the arrow-head can be modified with the help of attributes set by the macro `\psset arrowhead(...)`.

`\psaxes` Origin(X1,X2, Y1,Y2)  
`\psaxes` Origin(X1,X2, Y1,Y2, Z1,Z2)  
`\psaxes` Origin(L)  
 Axes defined by their Origin and coordinate ranges. Each axis is parallel to the corresponding absolute axis and is drawn as an oriented arrow from start value to stop value.  
 The short form `\psaxes` Origin(L) is equivalent to `\psaxes` Origin(0,L, 0,L) in 2D and `\psaxes` Origin(0,L, 0,L, 0,L) in 3D.  
 The end points of the axes are given by `\figptsaxes`.

--- Curve ---

`\psBezier` N [Pt<sub>1</sub>, ..., Pt<sub>{3N+1}</sub>]

Bezier curve defined by N cubic arcs. The arc number i is defined by the four control points P<sub>{j}</sub>, P<sub>{j+1}</sub>, P<sub>{j+2}</sub>, P<sub>{j+3}</sub> with j=3i-2.

The curve interpolates each 3 points beginning with the first, i.e. at P<sub>{3i-2}</sub>, i=1,...,N+1. At these points, the curve is only C0. To obtain G1 continuity at P<sub>j</sub>, the points P<sub>{j-1}</sub>, P<sub>j</sub> and P<sub>{j+1}</sub> must be aligned.

The total number of points must be 3N+1 and is not checked.

`\pscurve` [Pt0,Pt1,...,PtN,PtN+1]

C1 curve that interpolates the points P1, P2,...,Pn. The curve consists of n-1 Bezier cubic arcs. The direction of the tangent at Pi is given by Pi-1Pi+1, i=1,...,n. To get a C1 closed curve, the last three points must be the same as the first three ones.

The shape of the curve can be modified by a roundness coefficient to be set by the macro `\psset curve(...)`. The best values for this coefficient are in the interval [0.15, 0.3] (0 gives a polygonal line).

--- Flow chart ---

`\psfcconnect` [Pt1,Pt2,...,PtN]

Connections between nodes in a flow chart. Each connection path can be a polygonal line or a curved line. An arrow-head is drawn by default at the middle point of the path joining the points Pt1,Pt2,...,PtN.

The attributes of the path are set by the macro `\psset flowchart(...)`.

The frames at the nodes are drawn with the help of the macro `\psfcnode`.

`\psfcnode` [Pt1,Pt2,...,PtN]{Text}

Frames containing Text at each node centered on points Pt1,Pt2,...,PtN in a flow chart.

If the Text argument is empty, the text joined to the points when they are defined is used ; otherwise, this Text is used for every point Pt1,Pt2,...,PtN. The frame surrounding the text is reduced or enlarged according to the padding dimensions.

This attribute along with the other ones is set by the macro `\psset flowchart(...)`.

The connections between the nodes are usually drawn by the macro `\psfcconnect`. If no arrows are wanted, the best is to say `\psset arrowhead(length=0)` ; the macros `\psline` or `\pscurve` can also be used.

--- Triangle related macros ---

`\psaltitude` Dim [Pt1,Pt2,Pt3]

Altitude from Pt1 in the triangle (Pt1, Pt2, Pt3).

Dim is the dimension of the square at the foot of the altitude, which is drawn on either side of the altitude according to the order of the 2 points Pt2 and Pt3.

`\psnormal` Length,Lambda [Pt1,Pt2]

Exterior normal of length Length to the segment [Pt1, Pt2]. Lambda is the barycentric coordinate of the origin of the normal with respect to the segment [Pt1, Pt2], which sets the position of the vector along [Pt1, Pt2].

In 3D-mode, it works only in the plane Z=0.

--- Grid ---

`\psmesh` N1,N2 [Pt1,Pt2,Pt3,Pt4]

Mesh of N1 x N2 intervals on the quadrangle (Pt1, Pt2, Pt3, Pt4)

(N1 along [Pt1, Pt2] and [Pt3, Pt4], N2 along the 2 other segments).

A flag set by `\psset mesh(...)` allow to draw a diagonal inside each cell.

`\pstrimesh` Type [Pt1,Pt2,Pt3]

Mesh of the type Type triangle on the triangle (Pt1, Pt2, Pt3).



## INDEX

To help the user to locate the information, page numbers are printed according to the context in which each word of the index appears. When a word appears in the text, the corresponding page numbers are printed in roman font. When a word appears in an **example**, the corresponding page numbers are printed in **bold face** font. When a word appears in the previous *list*, the corresponding page numbers are printed in *italic* font.

<code>\defaultarrowheadangle</code> ...	23, 35, <b>36</b> , <i>73</i>
<code>\defaultarrowheadfill</code> ....	23, <i>73</i>
<code>\defaultarrowheadlength</code> ..	23, 35, <i>73</i>
<code>\defaultarrowheadout</code> ....	23, <i>73</i>
<code>\defaultarrowheadratio</code> ...	23, 36, <i>73</i>
<code>\defaultcolor</code> .....	23, <i>72</i>
<code>\defaultdash</code> .....	23, 25, <b>28</b> , <b>42</b> , <i>72</i>
<code>\defaultfcarrowposition</code> ..	24, 44, <i>73</i>
<code>\defaultfcarrowrefpt</code> ....	24, 44, <i>73</i>
<code>\defaultfcline</code> .....	24, 43, <i>73</i>
<code>\defaultfcradius</code> .....	24, 43, <i>73</i>
<code>\defaultfcshape</code> .....	24, 44, <i>73</i>
<code>\defaultfcthickness</code> .....	24, 44, <i>73</i>
<code>\defaultfcxpadding</code> .....	24, 44, <b>46</b> , <i>73</i>
<code>\defaultfcypadding</code> .....	24, 44, <i>73</i>
<code>\defaultfill</code> .....	23, <i>72</i>
<code>\defaultjoin</code> .....	23, <b>39</b> , <i>72</i>
<code>\defaultmeshdiag</code> .....	24, 42, <i>73</i>
<code>\defaultroundness</code> .....	24, 31, <i>73</i>
<code>\defaultsecondcolor</code> .....	24, <i>73</i>
<code>\defaultseconddash</code> .....	24, <i>73</i>
<code>\defaultsecondwidth</code> .....	24, <i>73</i>
<code>\defaultthirdcolor</code> .....	24, <i>73</i>
<code>\defaultupdate</code> .....	23, <i>72</i>
<code>\defaultwidth</code> .....	23, 24, <b>28</b> , <b>36</b> , <b>38</b> , <b>61</b> , <i>73</i>
<code>\figcoord</code> .....	6, <i>71</i>
<code>\figgetangle</code> .....	6, <b>29</b> , 51, <i>66</i>
<code>\figgetdist</code> .....	6, 7, <b>15</b> , 15, <b>29</b> , <i>66</i>
<code>\figinit</code> .....	<b>2</b> , 5, 6, 7, <b>8</b> , <b>10</b> , <b>15</b> , <b>17</b> , 17, <b>18</b> , <b>20</b> , <b>21</b> , <b>22</b> , 22, 23, <b>28</b> , <b>29</b> , <b>30</b> , <b>31</b> , <b>32</b> , <b>34</b> , <b>35</b> , <b>36</b> , <b>38</b> , <b>39</b> , <b>40</b> , <b>41</b> , <b>42</b> , <b>44</b> , <b>45</b> , 48, <b>49</b> , <b>50</b> , 51, <b>55</b> , <b>56</b> , <b>57</b> , <b>58</b> , 61, <b>63</b> , <i>65</i> , <i>67</i>
<code>\figinsert</code> .....	<b>2</b> , 3, <b>7</b> , <b>8</b> , <b>11</b> , <b>15</b> , 16, <b>17</b> , <b>18</b> , <b>20</b> , 20, <b>21</b> , 21, <b>22</b> , 22, <b>28</b> , <b>29</b> , <b>30</b> , <b>31</b> , <b>32</b> , <b>34</b> , <b>35</b> , <b>36</b> , <b>38</b> , <b>39</b> , <b>40</b> , <b>41</b> , <b>43</b> , <b>45</b> , <b>46</b> , 47, <b>49</b> , <b>50</b> , <b>55</b> , <b>58</b> , <b>59</b> , <b>62</b> , <i>65</i>
<code>\figpt</code> .....	<b>2</b> , 5, <b>6</b> , 6, 7, <b>8</b> , <b>10</b> , <b>15</b> , <b>17</b> , <b>18</b> , <b>20</b> , <b>21</b> , <b>22</b> , <b>28</b> , <b>29</b> , <b>30</b> , <b>31</b> , <b>32</b> , <b>34</b> , <b>35</b> , <b>36</b> , <b>38</b> , <b>39</b> , <b>40</b> , <b>41</b> , <b>42</b> , 43, <b>44</b> , <b>45</b> , <b>49</b> , <b>50</b> , 51, <b>55</b> , <b>58</b> , <b>62</b> , <b>63</b> , <i>66</i>
<code>\figptbary</code> .....	<b>2</b> , <b>8</b> , <b>10</b> , <b>11</b> , 11, <b>29</b> , <b>40</b> , <b>41</b> , 43, <i>66</i>
<code>\figptbaryR</code> .....	11, <b>29</b> , <i>66</i>
<code>\figptBezier</code> .....	12, 13, 30, <b>38</b> , <b>61</b> , <b>62</b> , <i>70</i>
<code>\figptcirc</code> .....	12, <b>49</b> , <b>50</b> , 52, <i>70</i>
<code>\figptcircumcenter</code> .....	<b>40</b> , 40, <i>69</i>
<code>\figptcopy</code> .....	7, <b>35</b> , <i>66</i>
<code>\figptcurvcenter</code> .....	13, 61, <b>63</b> , <i>70</i>
<code>\figptell</code> .....	11, 12, 27, <b>28</b> , <b>29</b> , 48, <i>69</i> , <i>70</i>

<code>\figptellP</code>	11, 12, 27, 70
<code>\figptendnormal</code>	13, <b>39</b> , 39, 48, 69
<code>\figptexcenter</code>	40, 69
<code>\figpthom</code>	<b>8</b> , 8, 11, <b>22</b> , 67
<code>\figptincenter</code>	<b>40</b> , 40, 70
<code>\figptinterlineplane</code>	48, 52, 68
<code>\figptinterlines</code>	12, 27, 68
<code>\figtmap</code>	9, 11, 52, 67
<code>\figptorthocenter</code>	<b>40</b> , 40, 70
<code>\figptorthoprojline</code>	11, <b>39</b> , 67
<code>\figptorthoprojplane</code>	48, <b>50</b> , 52, <b>61</b> , 67
<code>\figptrot</code>	<b>8</b> , 8, 11, <b>34</b> , 52, <b>61</b> , <b>62</b> , <b>63</b> , 63, 67
<code>\figptsaxes</code>	13, 37, <b>38</b> , <b>49</b> , <b>50</b> , 53, 68, 69, 75
<code>\figptscontrol</code>	13, 70
<code>\figptscontrolcurve</code>	13, 31, <b>38</b> , 70
<code>\figptshom</code>	10, 68
<code>\figptsintercirc</code>	12, <b>29</b> , <b>34</b> , 53, 69
<code>\figptsinterlinell</code>	12, 48, 69
<code>\figptsinterlinellP</code>	12, 69
<code>\figptsmap</code>	<b>10</b> , 10, 52, 67
<code>\figptsorthoprojline</code>	11, 68
<code>\figptsorthoprojplane</code>	48, 52, 68
<code>\figptsrot</code>	10, <b>30</b> , <b>36</b> , 52, <b>58</b> , <b>62</b> , 68
<code>\figptssym</code>	10, <b>30</b> , 52, <b>61</b> , <b>62</b> , 68
<code>\figptstra</code>	<b>8</b> , <b>10</b> , 10, <b>18</b> , <b>35</b> , <b>41</b> , <b>42</b> , <b>55</b> , 67, 68
<code>\figptsym</code>	<b>8</b> , 8, 11, 52, 67
<code>\figpttra</code>	<b>8</b> , 8, <b>15</b> , <b>35</b> , <b>38</b> , <b>45</b> , <b>46</b> , <b>58</b> , <b>61</b> , <b>62</b> , <b>63</b> , 67
<code>\figpttraC</code>	<b>8</b> , 9, <b>10</b> , <b>28</b> , <b>41</b> , <b>42</b> , 52, <b>55</b> , <b>58</b> , <b>62</b> , 67
<code>\figptvisilimSL</code>	52, <b>59</b> , 69
<code>\figscan</code>	1, 20, <b>21</b> , 21, 22, 27, 47, 65
<code>\figset</code>	48, 49, 65
<code>\figset proj</code>	48, 49, <b>50</b> , 50, <b>56</b> , <b>57</b> , <b>58</b> , 58, <b>63</b>
<code>\figsetmark</code>	<b>2</b> , 6, <b>8</b> , <b>11</b> , <b>15</b> , <b>16</b> , <b>17</b> , <b>18</b> , <b>19</b> , <b>28</b> , <b>29</b> , <b>31</b> , <b>38</b> , <b>40</b> , <b>50</b> , <b>57</b> , 71
<code>\figsetpname</code>	5, 6, 16, <b>30</b> , <b>39</b> , <b>55</b> , <b>57</b> , 66, 71
<code>\figsetroundcoord</code>	6, <b>57</b> , 71
<code>\figshowpts</code>	63, 71
<code>\figshowsettings</code>	24, 47, 57, 66, 72
<code>\figvectC</code>	7, <b>8</b> , <b>10</b> , <b>15</b> , <b>18</b> , <b>35</b> , <b>41</b> , <b>42</b> , <b>45</b> , 51, <b>55</b> , <b>58</b> , <b>62</b> , 66
<code>\figvectDBezier</code>	7, 12, 30, <b>38</b> , <b>62</b> , 70
<code>\figvectN</code>	7, 51, 52, 66
<code>\figvectNV</code>	7, <b>38</b> , 51, 52, 66
<code>\figvectP</code>	7, <b>50</b> , <b>58</b> , <b>62</b> , <b>63</b> , 66
<code>\figvectU</code>	7, <b>38</b> , 61, <b>62</b> , 67
<code>\figvisu</code>	<b>2</b> , 3, <b>7</b> , <b>8</b> , <b>11</b> , <b>15</b> , <b>16</b> , <b>17</b> , <b>18</b> , <b>20</b> , 20, <b>21</b> , <b>22</b> , <b>28</b> , <b>29</b> , <b>30</b> , <b>31</b> , <b>32</b> , <b>33</b> , <b>34</b> , <b>35</b> , <b>36</b> , <b>38</b> , <b>39</b> , <b>40</b> , <b>41</b> , <b>43</b> , <b>45</b> , <b>46</b> , <b>49</b> , <b>50</b> , <b>55</b> , <b>58</b> , <b>59</b> , <b>62</b> , 65, 66
<code>\figwrite</code>	2, 5, 16, <b>20</b> , <b>46</b> , 53, 69, 71
<code>\figwritebe</code>	17, 18, 71
<code>\figwritebn</code>	17, 18, 71
<code>\figwritebs</code>	17, 18, 71
<code>\figwritebw</code>	17, 18, 71
<code>\figwrittec</code>	16, 17, 17, <b>21</b> , <b>22</b> , <b>41</b> , <b>43</b> , <b>45</b> , <b>46</b> , 71
<code>\figwriteee</code>	<b>2</b> , 6, <b>8</b> , <b>11</b> , <b>15</b> , <b>16</b> , <b>17</b> , <b>18</b> , <b>30</b> , <b>36</b> , <b>40</b> , <b>46</b> , <b>49</b> , <b>50</b> , <b>55</b> , 71

<code>\figwritegce</code>	19, 19, 72
<code>\figwritegcw</code>	19, 19, 72
<code>\figwritege</code>	19, 19, 72
<code>\figwritegw</code>	19, 19, 72
<code>\figwriten</code>	2, 8, 11, 15, 16, 17, 18, 28, 29, 30, 35, 38, 39, 49, 50, 71
<code>\figwritene</code>	16, 17, 30, 36, 38, 43, 50, 55, 71
<code>\figwritenw</code>	8, 11, 15, 16, 17, 30, 40, 43, 50, 55, 71
<code>\figwritep</code>	16, 17, 17, 31, 46, 71
<code>\figwrites</code>	2, 8, 16, 17, 18, 37, 38, 39, 46, 50, 69, 71
<code>\figwritese</code>	16, 17, 30, 38, 43, 55, 71
<code>\figwritesw</code>	11, 16, 17, 28, 30, 36, 43, 49, 55, 71
<code>\figwritew</code>	2, 8, 16, 17, 18, 30, 36, 37, 38, 39, 40, 50, 69, 71
<code>\psaltitude</code>	39, 39, 76
<code>\psarccirc</code>	27, 28, 33, 53, 74
<code>\psarccircP</code>	27, 28, 34, 34, 53, 74
<code>\psarcell</code>	27, 28, 28, 29, 33, 48, 74
<code>\psarcellPA</code>	27, 28, 33, 75
<code>\psarcellPP</code>	27, 28, 75
<code>\psarrow</code>	8, 11, 35, 35, 38, 75
<code>\psarrowBezier</code>	36, 37, 62, 75
<code>\psarrowcirc</code>	36, 37, 49, 53, 75
<code>\psarrowcircP</code>	36, 37, 39, 50, 53, 75
<code>\psarrowhead</code>	35, 35, 37, 53, 75
<code>\psaxes</code>	10, 37, 38, 49, 50, 53, 54, 68, 69, 75
<code>\psbeginfig</code>	2, 3, 7, 8, 10, 14, 15, 17, 18, 23, 28, 29, 30, 31, 32, 33, 34, 35, 36, 38, 39, 40, 41, 42, 45, 46, 47, 49, 50, 55, 58, 59, 62, 65, 72
<code>\psBezier</code>	13, 30, 30, 31, 33, 61, 62, 70, 76
<code>\pscirc</code>	6, 14, 15, 17, 18, 27, 34, 53, 62, 74
<code>\pscurve</code>	13, 23, 31, 31, 32, 38, 44, 70, 76
<code>\psendfig</code>	2, 3, 8, 11, 14, 15, 17, 18, 28, 29, 30, 31, 32, 33, 34, 35, 36, 38, 39, 40, 41, 42, 45, 46, 49, 50, 55, 58, 59, 62, 72
<code>\psfcconnect</code>	43, 45, 46, 48, 73, 76
<code>\psfcnode</code>	43, 44, 45, 46, 48, 76
<code>\psline</code>	2, 8, 10, 14, 18, 28, 29, 30, 33, 34, 35, 36, 39, 40, 44, 50, 55, 58, 59, 62, 74, 76
<code>\pslineC</code>	14, 74
<code>\pslineF</code>	14, 74
<code>\psmesh</code>	23, 42, 42, 76
<code>\psnormal</code>	39, 39, 48, 76
<code>\psreset</code>	23, 31, 35, 35, 46, 72
<code>\psset</code>	10, 11, 14, 17, 18, 23, 24, 25, 26, 28, 30, 32, 33, 33, 34, 36, 37, 38, 39, 39, 41, 43, 44, 45, 47, 50, 55, 57, 59, 61, 62, 72, 73, 74
<code>\psset arrowhead</code>	10, 11, 35, 35, 36, 36, 38, 39, 44, 46, 49, 50, 57, 62, 75, 76
<code>\psset curve</code>	31, 32, 57, 76
<code>\psset flowchart</code>	43, 44, 45, 46, 57, 76
<code>\psset mesh</code>	42, 42, 57, 76
<code>\psset second</code>	41, 42, 46, 57
<code>\psset third</code>	44, 46, 57
<code>\pssetdefault</code>	2, 14, 23, 23, 24, 36, 45, 57, 73, 74
<code>\psstrimesh</code>	41, 41, 76