IRMAR   **Université de Rennes 1 – France**

Documentation of the
Fig4TEX macro package

(version 1.9.2)

Yvon Lafranche

Contents

Abstract. This document describes a set of TeX macros designed to *create a figure* at compilation time of the document, and to *write text* on it in a straightforward way, using the TeX fonts, under total control of the user. These macros can also be used to write a legend on an existing figure, given as a file created by any other software. Fig4TeX can be used with the traditionnal plain TeX environment as well as with LaTeX, in both DVI and PDF modes. They have been designed from the user's point of view and 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 in a TeX document has 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 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 TeX document, essentially because generally two softwares are needed: one for the TeX document, one for the figure. In particular, a question that arises very often is: can I write some text or caption using 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, respectively yes and no. Indeed, it is possible to write anything anywhere on a figure, and provided that we have followed some elementary recommendations, 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 can be designed in a *symbolic* way since the points involved are the result of mathematical transformations or geometrical definitions. Moreover, since the result of the process is a TeX box, every usual 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 so-called graphical 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 graphical 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 "external" file, i.e. a file created by any software in a compatible format (EPS, JBIG2, JPEG, PDF, PNG) can be handled as well; step 2 has then to be skipped. A tool macro (`\figscan`) has been designed to facilitate the positionning of text: see the section **Text annotation** for further information on this topic.

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 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. Brakets (`[]`), parentheses (`()`) and slashes (`//`) are used to surround groups of arguments that are logically linked together. In most of the cases, brakets are devoted to point numbers, parentheses are devoted to numerical values and slashes are used to set parameters in the transformation macros. TEX's braces (`{}`) are obviously always available.

d. Numerical values are intended to be given without any unit specification, except for the definition of some default dimensions (see `\figsetdefault`). However, specifying a unit is also allowed for dimensions that are logically independant of the scale of the figure, e.g. a writing distance (see argument `Distance` of `\figwrite*` in the section **Writing text on the figure**).

The name of the user macros all begin with `\fig`. They are divided in three groups:

- Macros beginning with `\figdraw` are commands used to create the graphical file. These macros are intended to be called during step 2, they have no effect otherwise.

- Macros beginning with `\figwrite`, so-called non-mute macros, are the only ones that write something on the figure. Logically, they have to be called during step 3.

- The remaining macros are mute. They are related to the definition of the geometry or parameter settings. They can be used at any step.

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 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. Moreover, some elementary but fundamental informations about TEX are given in the section **Using this macro package**. LATEX users will also find there some practical hints.

§**2. A first look**

Here is a short example to show how it works. Consider the following file (with LATEX, simply put the following lines, except the last one, between `\begin{document}` and `\end{document}`):

```
\input fig4tex.tex
% 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 graphical file
\figdrawbegin{}
\figdrawline[1,2,3,1]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 1}{
\figwriten 1:(4)
\figwritew 2:(2)
\figwritee 3:(2)
\figset write(mark=$\bullet$)\figwrites 5:$G$(4)
}
\centerline{\box\figBoxA}
\bye
```

If we compile this file with TEX or LATEX, then we obtain the figure 1 . Now, let us comment these statements.

0. The first line loads the macro package.

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.
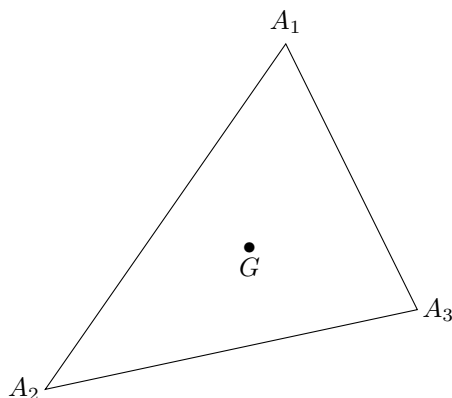
Figure 1

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.

2. Creation of the graphical file that will contain the figure, here the three edges of the triangle, drawn as a single closed broken line.

3. Writing text or comments on the figure. The macro `\figvisu` builds a box whose name (here `\figBoxA`) 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. The graphical file created just before is automatically included and "put" into the box.

   The text associated to each point is printed at the place chosen by the user, i.e. at $4\,\mathrm{pt}$ towards the north for point #1, at $2\,\mathrm{pt}$ towards the west for point #2, at $2\,\mathrm{pt}$ towards the east for point #3, at $4\,\mathrm{pt}$ 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 • (`$\bullet$`) is 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, which 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.
- In the above example, the graphical information is stored in a temporary file included just after its creation. If the drawing is to be referred later in the document, it must be stored in a permanent file whose name, chosen by the user, is passed as argument firstly to `\figdrawbegin` for the creation and secondly to `\figinsert` for the inclusion (see the section **Graphical files handling**).
- The box `\figBoxA` is preallocated in Fig4TeX and is available to the user, along with two other ones, `\figBoxB` and `\figBoxC`. It must be emphasized that the *same* box (e.g. `\figBoxA`) can be used for several figures. For example, this documentation uses no more than these three boxes (see the section **Writing text on the figure**).
- 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, very few control is made on the validity of the arguments of the macros.
- In addition to the graphical files explicitly wanted by the user and the classical files produced by TeX, this package creates temporary files whose names have the form `\jobname*.anx`. They can be safely removed.
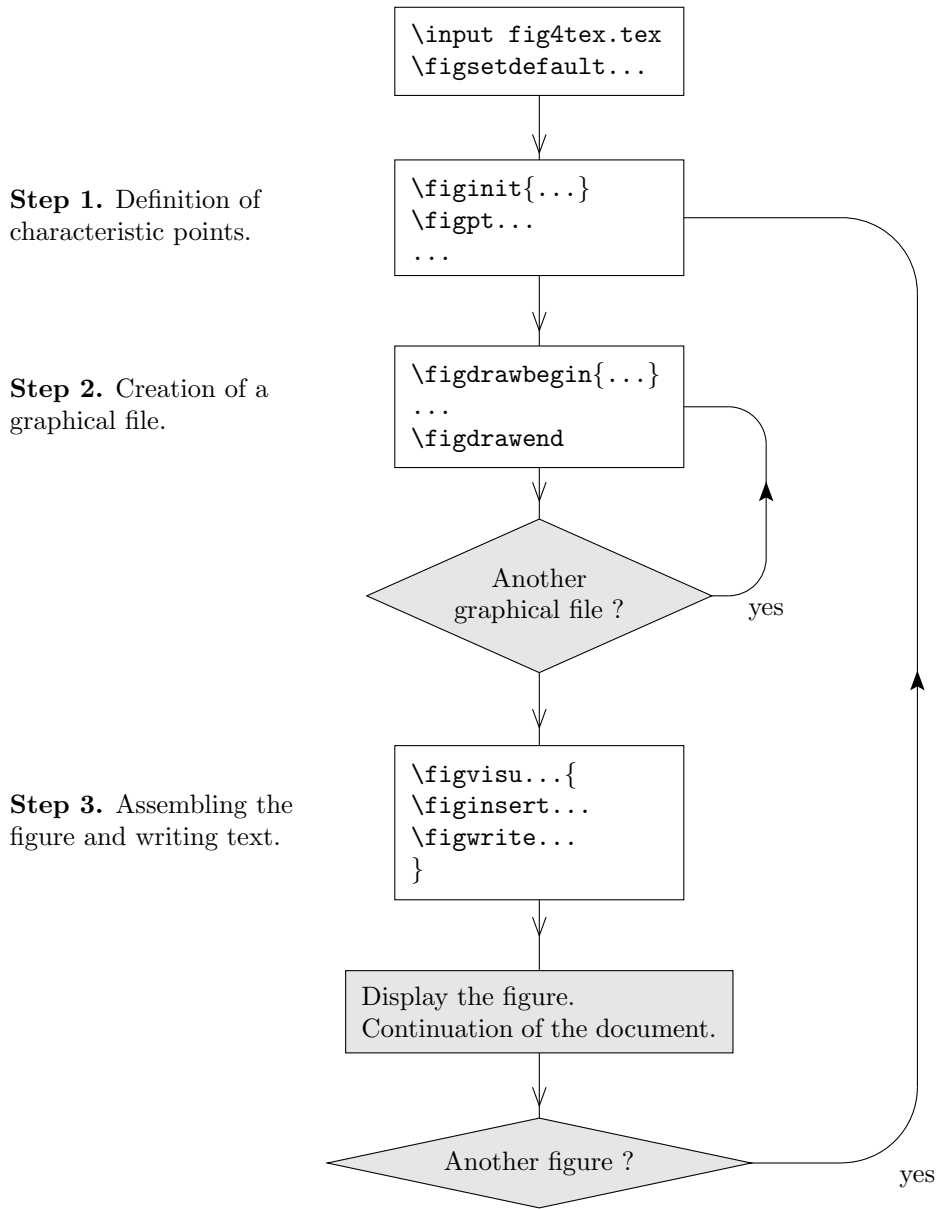
Figure 2

This figure shows the general usage of Fig4TEX. The internal loop is seldom used. The text of the program that produces this figure is given in the subsection **Flow chart**.

<center>§**3. Definition of points**</center>

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 TeXbook:

<center>

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).

</center>

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 specify a scale factor before the unit. It may be useful for example when the user wants to give the 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

<center>`\figpt NewPt :Text(X,Y)`</center>

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.

The second argument `Text` is the so-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. Let's 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. This default behaviour can be changed by saying for instance, `\figset write(ptname={$X^{(#1)}$})` which will now display $X^{(i)}$ instead of A_i. In the argument field, `#1` stands for the point number. The default setting is thus equivalent to `\figset write(ptname={$A_{#1}$})`.

To remove the automatic association of text, just say `\figset write(ptname={})`. 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 `\figset write(ptname={})` which from now on 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 {}, for example `\figpt 1 :{The coordinates of this point are:(1,2)}(1,2)` .
  - a box. Inside a `\vbox`, think to set `\parindent=0pt` if needed and use `\hfill\break` instead of `\par`.

<center>5</center>

By default, the position of the point is not displayed. However, it is sometimes needed and the user can choose anything he wants among TEX's possibilities to highlight the point location. For this purpose, markers like •, ○, ., ×, + are most commonly chosen. For example, saying `\figset write(mark=$\times$)` will select the symbol ×. To get back to the default state, just say `\figset write(mark={})`.

In fact, the following marks are not quite vertically centered: ∗ (`\ast`), • (`\bullet`), ○ (`\circ`) and ◇ (`\diamond`). Thus, 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:
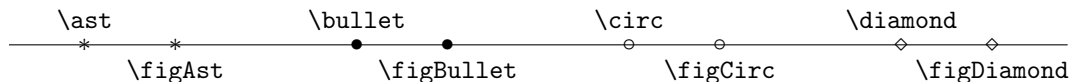
Remark: By the way, for the same reason, do not use the macro `\cdot`: just say `\figset write(mark={.})`.

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 example shown on figure 1, one could have written during step 1:

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

or during step 3:

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

which would have printed $A_3{\scriptstyle(130.00,20.00)}$ on the graph.

The argument `Ndec` of `\figcoord` is a positive integer setting the number of decimals to be printed. TEX provides at most 5 decimals. Missing decimals, for integer values for example, are printed as 0. By default, the printed values are rounded to the integer part (`Ndec` = 0), or up to the first `Ndec` decimals, according to the value of the argument `Ndec`. To disable this rounding, one can say `\figset write(rounding = no)`, and then `\figset write(rounding = yes)` to enable it again for subsequent printed values. Notice that if `Ndec` > 5, the values are not rounded: we obtain the internal raw value.

Moreover, besides the values `yes` and `no`, this form of the `\figset` macro also accepts a number `Ndec` (`\figset write(rounding = Ndec)`) allowing to specify the number of decimals to be printed. The `yes` and `no` values are respectively equivalent to 2 (the default value) and 5.

If the argument `Ndec` of `\figcoord` is omitted, then the number of decimals to print is the one set by the last call to `\figset write(rounding = ...)` ; if the argument `Ndec` of `\figcoord` is present, this specification is obeyed.

Remarks:
- `\figcoord` can be used in a `ptname` context in order to print together the point name and its coordinates, for example, by saying `\figset write(ptname={$A_{#1}$ \figcoord{1}})`. This will automatically apply for any point, except for those bearing a specific joined text, as just said.
- We used here the general macro `\figset` with the keyword `write`. This macro is used to set many other parameters and is fully described in the section **Setting graphical attributes**.

At last, let us mention that the text argument described in this section may contain a real number. When this value is not under the control of the user, e.g. if it is the result of a computation (see next section), it may be useful to print a rounded value. This can be achieved with the macro `\figround{RealNumber}`, to be called in the definition of the text argument in the same way as `\figcoord`. As presented above, the number of decimals to print is set by saying `\figset write(rounding = Ndec)`.

4. Symbolic constants

As already mentionned in the remarks following the first example, a symbolic constant can be used to store a string or a numerical value. This is done by defining a TEX macro, using the syntax:

$$\def\MyMacro\{definition\}$$

Take note that the macro `\MyMacro` is created or overwritten if it already existed (see details about macros in the section **Using this macro package**).

Thus, the name of the graphical file to be created (see the section **Graphical files handling**) can be put in a macro definition. The reference to the file is then done via this macro, `\MyGrFile` in the following example:

```
% 2.  Creation of the graphical file
\def\MyGrFile{Figure.gra}  % Definition of the macro
\figdrawbegin{\MyGrFile}   % First reference
...
% 3.  Writing text on the figure
\figvisu{\figBoxA}{Caption}{
\figinsert{\MyGrFile}      % Second reference
...
```

Going to the numerical side, this feature can also 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\Abscissa{12.3456}
\figpt 10:(\Abscissa, 120)
\figpt 11:(\Abscissa, -10)
```

The points #10 and #11 have the same abscissa which is defined just before as a TeX macro.

Moreover, the statement
```
\figget distance = \Result[Pt1,Pt2]
```
computes the euclidian distance between the two points `Pt1` and `Pt2` (in user unit), and stores it in the macro `\Result` 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 section **Example**). This feature allows us to build geometrical constructions depending on the coordinates of only a few characteristic points.

Let C, P_1 and P_2 stand for `Center`, `Pt1` and `Pt2`. In the same idea as the previous one, the macro
```
\figget angle = \Result[Center,Pt1,Pt2]
```
computes the value (in degrees) of the oriented angle $(\overrightarrow{CP_1}, \overrightarrow{CP_2})$.

The two previous commands also come with a more general syntax allowing enhanced possibilities:
```
\figget distance OP value = \Result[Pt1,Pt2]
\figget angle    OP value = \Result[Center,Pt1,Pt2]
```
where `OP` is an arithmetical operator to be chosen among `*`, `/` or `+`, and `value` is a symbolic numerical constant (as defined above), or can simply be a number. The result stored in the macro `\Result` is the distance or the angle, modified by the specified operation whose second operand is `value`. For example,
```
\figget distance * 3.45 = \Coef[1,2]
\figget distance + -\D12 = \Zero[1,2]
```
stores in `\Coef` the distance between the points #1 and #2, multiplied by 3.45, and, assuming `\D12` holds the distance between the points #1 and #2, stores 0 in `\Zero`, result of the substraction of the two same quantities. Thus, if `e` denotes the neutral element of the operator `OP`, the simplified syntax first given above is equivalent to:
```
\figget distance OP e = \Result[Pt1,Pt2]
\figget angle    OP e = \Result[Center,Pt1,Pt2]
```
Remarks:
- If the operand `value` is missing, then the operation is ignored.
- The `\Result` macro can be used as a text to write on a figure. The printed value can be rounded by saying `\figround{\Result}` (see details in the previous section).

7

5. Geometry: 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. Indeed, 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 `\figget distance`.

The macro `\figvectDBezier` computes the derivative vector 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*/

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.
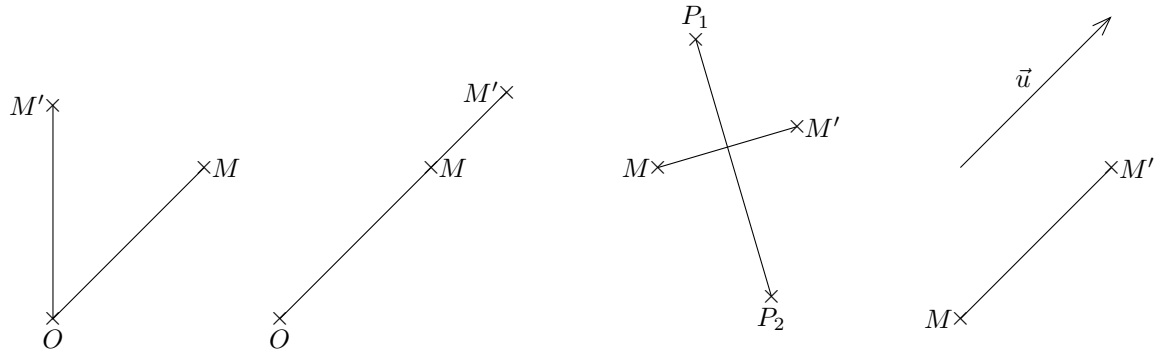

Figure 4. Rotation ($\pi/4$), homothety (ratio 1.5), symmetry / $[P_1, P_2]$ and translation of vector $\vec{u}$

```
% 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 graphical file
\figdrawbegin{}
\figdrawline[1,0,2]                      % rotation
\figdrawline[10,11,12]                   % homothety
\figdrawline[21,22]\figdrawline[23,24]   % symmetry
\figdrawline[31,32]\figdrawarrow[34,35]  % translation
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{{\bf Figure 4.}\\Rotation ($\pi/4$), homothety (ratio 1.5),
symmetry / $[P_1, P_2]$ and translation of vector $\vec{u}$}{
\figset write(mark={$\times$})
\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:(0.15)\figwrites 24:(0.15)                         % symmetry
\figwritew 31:(0.1) \figwritee 32:(0.1)                          % translation
\figset write(mark=)\figptbary36:$\vec{u}$[34,35;1,1]\figwritenw 36:(0.1) % translation
}
\centerline{\box\figBoxA}
```

The inversion is a transformation also available with the macro

\figptinv NewPt :Text= Pt /Center, Ratio/

which computes the image NewPt of Pt by the inversion of center Center and of ratio Ratio. Center and ratio are also called pole and power. Its use is illustrated on the figure 5.
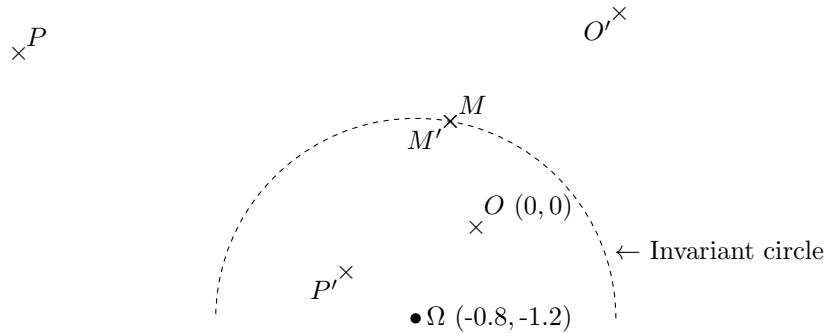
Figure 5. Inversion of center Ω and ratio 7.

Here the ratio is positive; if it was equal to -7, the points O', P (since it is computed starting from O') and M' would have been their symmetric with respect to Ω. The program that produces this figure is the following:

```
\figinit{cm}
% 1.  Definition of characteristic points
\figpt 0:$\Omega$ \figcoord{1}(-0.8,-1.2) % Inversion center
\def\Rinv{7}\def\sRinv{2.64575}          % Inversion ratio and sqrt(ratio)
\figpt 1:$O$ \figcoord{0}(0,0)    % Origin O
\figptinv 11 :$O'$= 1 /0,\Rinv/   % Its image O' by the inversion
\figptrot 2 := 11 /0,90/\figpthom 2:$P$=2 /0,1.3/ % Compute another point P...
\figptinv 12 :$P'$= 2 /0,\Rinv/   % ...  and its image P' by the inversion
\figptcirc 3 :$M$:  0;\sRinv (80) % Point M lying on the invariant circle...
\figptinv 13 :$M'$= 3 /0,\Rinv/   % ...  and its image M' by the inversion
%
% 2.  Creation of the graphical file
\figdrawbegin{}
% Draw (half of) the invariant circle
\figset (dash=4)
\figdrawarccirc 0;\sRinv(0,180)
\figdrawend
%
% 3.  Writing text on the figure
\figvisu{\figBoxA}{{\bf Figure 5.} Inversion of center $\Omega$ and ratio \Rinv.}{
% Show the invariant circle
\figptcirc 5 ::  0;\sRinv (20)
\figwritee 5:$\leftarrow$ Invariant circle(4pt)
% Show the involved points
\figset write(mark=$\figBullet$)\figwritee 0:(4pt)   % Inversion center
\figset write(mark=$\times$)\figwritene 1,2,3:(4pt)  % Points O, P, M,...
\figwritesw 11,12,13:(4pt)    % ...and their images O', P', M' by the inversion
}
\centerline{\box\figBoxA}
```

The transformations shown on figure 4 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

$$\text{\figptmap NewPt :Text= Pt /InvPt/A11,A12 ; A21,A22/}$$

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 Aij, $1 \le i \le 2$, $1 \le j \le 2$, 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 6: 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. The text of the program that produces this figure is given further, after the "set" versions of the macros have been described.

Indeed, to make some geometrical constructions easier, a "set" or "multiple points" version of each of these macros has 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 $k, k+1, \ldots, k+N-1$. Text eventually previously associated with the result points is lost and moreover no text can be joined to the result points.
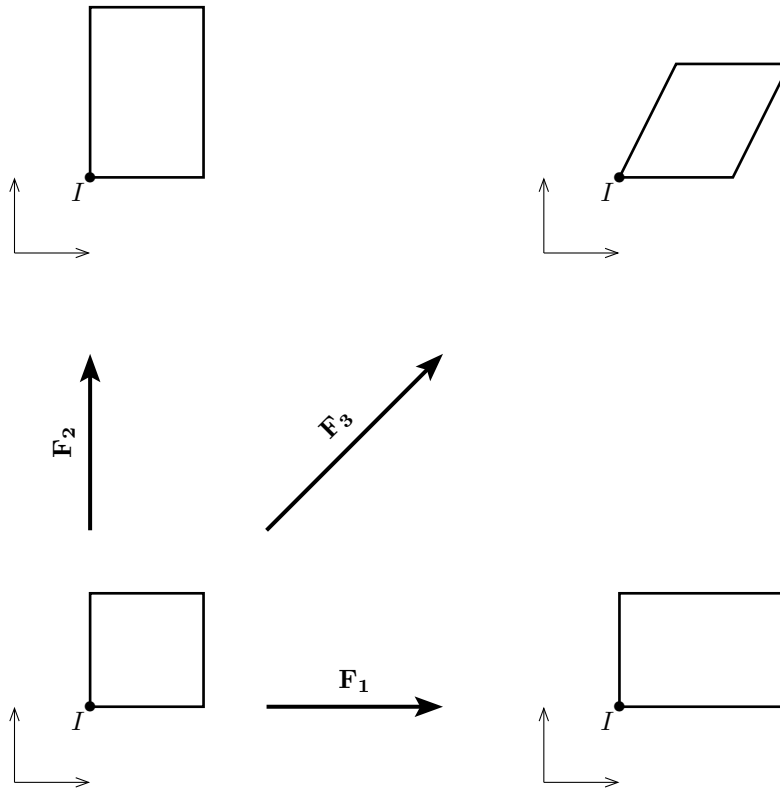

Figure 6

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 since the data points are taken into account sequentially, starting from the first element in the list, some of them may be overwritten before they are used as data.

However, we can take advantage of this last situation to compute, in a quite simple way, a sequence of points $P_{n+1} = f(P_n)$, $n = 1, \ldots N$, starting from the *only* data point P_1, by just saying:

> \macroname P2 = P1,P2,... PN */transformation definition/*

since P_2, once computed, is used immediately after as data to compute P_3, and so on.

The case $I_r = I_d$ is very useful ; it corresponds to the "in place" computation of $P_n = f(P_n)$, $n = 1, \ldots N$:

$$\texttt{\textbackslash macroname P1 = P1,P2,... PN } \textit{/transformation definition/}$$

Since these macros are extensions of the previous ones `\figpt`*xxx* and can produce more than one result, their names are `\figpts`*xxx*. 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/
\figptsinv NewPt1 = Pt1, Pt2, ..., PtN /Center, Ratio/
\figptsmap NewPt1 = Pt1, Pt2, ..., PtN /InvPt/A11,A12 ; A21,A22/
```

The text of the program that produces the figure 6 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 graphical file
\figdrawbegin{}
\figset arrowhead(length=0.2)
\figdrawaxes 0(1)\figdrawaxes 10(1)\figdrawaxes 20(1)\figdrawaxes 30(1)
\figset(width=1)
\figdrawline[1,2,3,4,1]\figdrawline[11,12,13,14,11]
\figdrawline[21,22,23,24,21]\figdrawline[31,32,33,34,31]
\figset(width=1.5)\figset arrowhead(fill=yes,length=0.4)
\figdrawarrow[15,16]\figdrawarrow[25,26]\figdrawarrow[35,36]
\figdrawend
%
```

```
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 6}{
% Name of each transformation
\figwriteln 15,16:\bf F$_{\hbox{$\bf\scriptstyle 1$}}$(0.2)
\figwriteln 25,26:\bf F$_{\hbox{$\bf\scriptstyle 2$}}$(0.2)
\figwriteln 35,36:\bf F$_{\hbox{$\bf\scriptstyle 3$}}$(0.2)
% Mark each invariant point
\figset write(mark=$\figBullet$)\figwritesw 1,11,21,31:$I$(0.1)
}
\centerline{\box\figBoxA}
```

The last macro is

$$\text{\texttt{\textbackslash figptorthoprojline NewPt :Text= Pt /LinePt1, LinePt2/}}$$

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

$$\text{\texttt{\textbackslash figptsorthoprojline NewPt1 = Pt1, Pt2, ..., PtN /LinePt1, LinePt2/}}$$

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

$$\text{\texttt{\textbackslash figptbary NewPt :Text[Pt1,...\ \ ,PtN ; Coef1,...\ \ ,CoefN]}}$$

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

$$\text{\texttt{\textbackslash figptell NewPt :Text: Center;XRad,YRad (Angle,Inclination)}}$$

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 α 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

$$\text{\texttt{\textbackslash figptellP NewPt :Text: Center,PtAxis1,PtAxis2 (Angle)}}$$

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 $(\overrightarrow{CA_1}, \overrightarrow{CA_2})$. 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

<div align="center">\figptcirc NewPt :Text: Center;Radius (Angle)</div>

has been written to obtain the result in a more straightforward way (see figure 5 for a first example). This calling sequence is equivalent to

<div align="center">\figptell NewPt :Text: Center;Radius,Radius (Angle,0)</div>

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

<div align="center">\figptinterlines NewPt :Text[LinePt1,Vector1; LinePt2,Vector2]</div>

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

<div align="center">\figptsintercirc NewPt1 [Center1,Radius1 ; Center2,Radius2]</div>

computes the intersection of two circles. More precisely, we consider the two circles defined by their center and radius, here respectively (Center1, Radius1) and (Center2, 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 (NewPt1, Center1, 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

<div align="center">\figptsinterlinell NewPt1 [Center,XRad,YRad,Inclination ; LinePt1,LinePt2]</div>

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 \figptell. Also, as explained jut 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:

<div align="center">\figptsinterlinellP NewPt1 [Center,PtAxis1,PtAxis2 ; LinePt1,LinePt2]</div>

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 \figptellP 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 weigths 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

<div align="center">\figptBezier NewPt :Text: t [Pt1,Pt2,Pt3,Pt4]</div>

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

<div align="center">\figvectDBezier NewVect : n, t [Pt1,Pt2,Pt3,Pt4]</div>

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.

<div align="center">14</div>

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 `\figdrawcurve` when it is invoqued by the corresponding calling sequence `\figdrawcurve [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 $k, k+1,\ldots k+M$ with $M = 3(N-1)$. `\NbArcs` denotes a TEX 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 `\figdrawBezier \NbArcs [NewPt1, NewPt1+1,..., NewPt1+M]` draws exactly the same curve, provided that the value of the curve roundness has not been changed. As a consequence, the data points `Pti`, $\mathtt{i} = 1,\ldots N$ are duplicated on output: indeed, the point number $k + j$ is equal to the point number `Pti` where $j = 3(\mathtt{i}-1), \mathtt{i} = 1,\ldots N$. The numbers assigned to the points created must be different from any of the data points `Pt0`, `Pt1`,...`PtN`, `PtN+1`.

Remarks:
- this macro uses the current value of the curve roundness, which must logically be the same used by `\figdrawcurve`,
- 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 `\figdrawcurve`, using `\figptBezier` in a second step.

Special cases

We mention here some macros that compute points linked to a particular context and are useful 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**.

1. Principle

The first macro to call is `\figdrawbegin`. It initializes all the settings related to the drawing macros. Its only argument is the name of the graphical file to be created. If this argument is empty, a default file is used, which is updated at each compilation (see section **Graphical files handling** for more information, in particular about the typesetting mode and the update process).

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

Between `\figdrawbegin` and `\figdrawend`, 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 *whenever* the graphical file is created or updated. This is the default behavior. On the contrary, every drawing or computing macro called between `\figdrawbegin` and `\figdrawend` is ignored if the graphical file exists and no update is wanted. This can be useful to ensure 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 arrowhead. They are all set by the macro `\figset`.

Each of these attributes have a default value, corresponding to what is generally expected by most people. It may happen however that some of these choices do not meet the user's agreement: for example, one may prefer to use a particular color, line width or arrowhead opening angle by default. For this reason, the macro `\figsetdefault` 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

                    `\figdrawline[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 exist. Their prototypes are

                    `\figdrawlineC(X1 Y1, X2 Y2,..., XN YN)`
                    `\figdrawlineF{FileName}`

For the first macro, the coordinates of the points defining the line are given as argument in 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 ; generally, each group of coordinates (defining one point) appear on one line.

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 blank 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

                    `\figdrawcirc Center (Radius)`

3. Example

Now, let us examine an example to illustrate some of the macros presented so far. First, we draw a circle centered on the origin O and passing through a given point M. Note the use of the macro `\figget distance` to compute the radius r (macro `\DistOM` in the program below). Then, we create the point P, intersection of the circle and the X axis (computed as the result of the translation of O by the vector $(r, 0)$), and the point Q such that $\overrightarrow{PQ} = 1.5\,\overrightarrow{OP}$. At last, we draw the circle with center Q and radius PQ.

We can see the result on the figure 7. 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 new result.
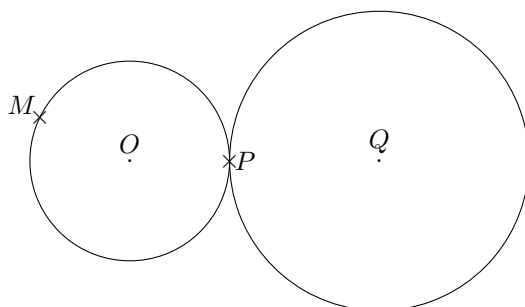
Figure 7

```
% 1.  Definition of characteristic points
\figinit{0.5in}
\figpt 1:$O$(0,0)\figpt 2:$M$(-0.9354, 0.45467)
\figget distance=\DistOM[1,2] \figvectC 10(\DistOM,0)
\figpttra 3:$P$= 1 /1,10/ \figpttra 4:$Q$= 3 /1.5,10/
% 2.  Creation of the graphical file
\figdrawbegin{}
\figdrawcirc 1(\DistOM)
\figget distance=\DistPQ[3,4] \figdrawcirc 4(\DistPQ)
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 7}{
\figset write(mark=$\times$)\figwritenw 2:(2pt)\figwritee 3:(2pt)
\figset write(mark=.)\figwriten 1,4:(3pt)}
\centerline{\box\figBoxA}
```

Notice that instead of `\figget distance=\DistPQ[3,4]`
we could have written `\figget distance*1.5=\DistPQ[1,2]`
although the first command is more suitable, since we just inquire information once all geometric computations have been made. Indeed, the factor 1.5 should be the same in this context and in the translation used to compute the point Q. However, if we persist in doing that, in order to achieve it in a more clean and robust way, one solution is to define a symbolic constant used in the two contexts:

```
\def\Factor{1.5}
\figpttra 4:$Q$= 3 /\Factor,10/
    ⋮
\figget distance*\Factor = \DistPQ[1,2]
```

§5. Writing text on the figure

Since the graphical 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, the rigth macro to use is \figvisu. 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. This box is filled according to the third argument and is the result of the macro \figvisu. Note that the box produced is a *vertical* box (\vbox).

Remark: The three boxes \figBoxA, \figBoxB and \figBoxC have already been reserved in Fig4TeX and are available to the user. Most of the time, one box is sufficient in a whole document, except when figures must be displayed side by side. For example, two figures can be displayed side by side by saying: \centerline{\box\figBoxA\hfil\box\figBoxB}. If a fourth box is necessary, the best is to make it private: first choose a name, e.g. \figBoxD (logical choice !), and add the statement \newbox\figBoxD near \input fig4tex.tex or \usepackage{fig4tex} in the document header.

The second argument is the figure caption. It can be void, but can also be, for example, a box containing several lines. The current figure number is available to the user in the document with the help of the expandable macro \figforTeXFigno. Moreover, in order to allow a forward reference before next call to \figvisu, the macro \figforTeXnextFigno provides the number of the next figure number. They are both updated by \figvisu. The first one can be used for example in the caption :

\figvisu{\figBoxA}{\bf Figure \figforTeXFigno}{ ...

The third argument is a list of commands that makes 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 in order to prevent unexpected white spaces to disturb the placement of the legend.

To show the drawing, one has to insert it in the box. The most common situation is to call \figdrawbegin without any argument; in this case, the drawing is *automatically* taken into account by the following call to \figvisu. On the contrary, if \figdrawbegin has been given a file name, the corresponding drawing must be inserted manually into the final box with the help the macro \figinsert whose first argument is the name of the graphical 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, each of them created by a separate call to \figdrawbegin with a specific file name. The drawings will then overlap since the origin is common. The second argument of \figinsert is a scale factor. This argument is also optional; it is described in the section **Text annotation**.

⟶ If no argument is given to \figinsert, it is equivalent to use the automatic behavior which consists, as explained just above, in inserting the default file created by the previous call to \figdrawbegin{} (see the section **Graphical files handling**).

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. The location of the attach point can be made visible by a marker by saying \figset write(mark=*marker*) as already seen in section **Text argument**. By default, the point marker is empty which makes the attach point invisible.

The simplest macro is \figwritep[Pt1, Pt2, ..., PtN] which writes the point marker 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 TeX'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 \figset write(ptname=*new name*).

18

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 `Pti`, 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 `Pti`, 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 8 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. The figure is followed by the text of the program that produces it.
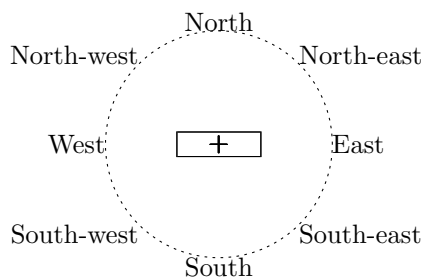
Figure 8

```
% 1. Definition of characteristic points
\figinit{cm}
\def\Dist{1.5}\figpt 0:(0, 0)
% 2. Creation of the graphical file
\figdrawbegin{}
\figset(dash=5)\figdrawcirc 0(\Dist)
\figdrawend
% 3. Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 8}{
\figset write(mark=+)
\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:
\figset write(mark=\boxit{1pt}{\phantom{Center}})\figwritep[0]
} \centerline{\box\figBoxA}
```

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 distance between the point and the text 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 figure 7). 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, we 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 9 which is followed by the program that produces it.
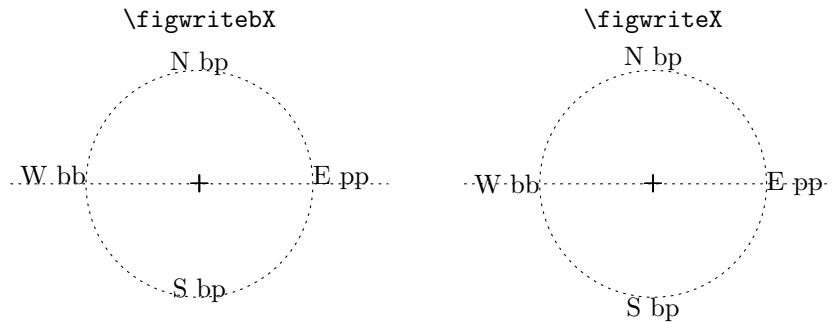
Figure 9

```
% 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 graphical file
\figdrawbegin{}
\figset(dash=5)
\figdrawcirc 0(\Dist)\figdrawline[1,2]
\figdrawcirc 10(\Dist)\figdrawline[11,12]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 9}{
\figset write(mark=+)
\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)\figwritee 10:E pp(\Dist)
\figwriten 10:N bp(\Dist)\figwrites 10:S bp(\Dist)
\figwriten 10:\CtrlSq{figwriteX}(\DDist)
} \centerline{\box\figBoxA}
```

20

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 10, 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.

\figwritebX   \figwriteX

bb   pp   mm   bb   pp   mm
——+———+———+———   ——+———+———+———
bb   pp   mm   bb   pp   mm

Figure 10

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

        \figwritegw Pt1, Pt2, ..., PtN :Text(DistanceX,DistanceY)
        \figwritege 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 11 where the vectors (DistanceX,DistanceY) are drawn from the attach point.

General West| |General Centered East

                              *

General Centered West|/ |General East
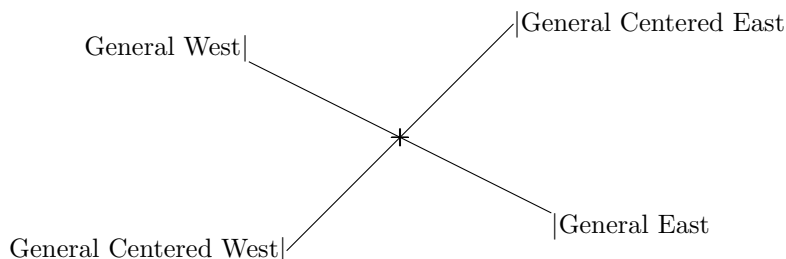
Figure 11

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:

        \figset write(mark=+)
        \figwritegw 0:General West$|$(2,10mm)
        \figwritegcw 0:General Centered West$|$(1.5,-1.5)
        \figwritegce 0:$|$General Centered East(1.5,1.5)
        \figwritege 0:$|$General East(2,-1)

The following macros are intended to write some text with respect to a broken line (the "l" in their name stands for "line"):

        \figwritelw Pt1, Pt2, ..., PtN :Text(Distance)
        \figwritele Pt1, Pt2, ..., PtN :Text(Distance)
        \figwriteln Pt1, Pt2, ..., PtN :Text(Distance)
        \figwritels Pt1, Pt2, ..., PtN :Text(Distance)
        \figwriteln Pt1, Pt2, ..., PtN :Text(Distance,lambda)
        \figwritels Pt1, Pt2, ..., PtN :Text(Distance,lambda)

21

Each part of the line is an oriented segment $[P_i, P_{i+1}]$ whose end points are two consecutive points in the given list. Such a segment plays the same role as the attach point for the writing macros presented above. Indeed, the text is placed at the west, east, north or south of the segment, considered as the reference object and the writing direction is defined by the vector $\overrightarrow{P_i P_{i+1}}$. The text is written at the given distance from the segment. More precisely, the shortest distance between the bounding box of the text and the segment is considered. For the west and east position, the center of the text bounding box is aligned with the segment. Without unit specification, the current unit (chosen by \figinit call) is assumed. This principle is illustrated on the following figure where self explained words are written with respect to a first horizontal segment and with respect to a second one rotated by 45°.

By default, for north or south position, the text is centered with respect to the middle of the segment. By adding an optional real parameter (lambda) after the mandatory writing distance, the position of the text can be shifted along the segment: the value 0 corresponds to the first end point P_i and 1 to the second end point P_{i+1} of the segment ; thus by default, this parameter is set to 0.5. The third segment on the figure shows the text "North" (resp. "South") positionned at 80% (resp. 10%) of the length of the segment.
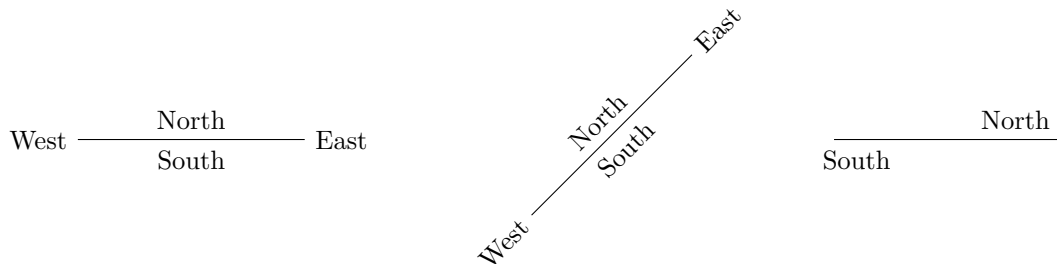
Figure 12. Position of a text with respect to a line.

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

```
% 1.  Definition of characteristic points
\figinit{cm}
\figpt 1:(0,1)
\figpt 2:(3,1)
\figpt 3:(6,0)
\figpt 4:(8.121,2.121)
\figvectC 10(10,0)\figptstra 5=1,2/1,10/
% 2.  Creation of the graphical file
\figdrawbegin{}
\figdrawline[1,2]\figdrawline[3,4]\figdrawline[5,6]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{{\bf Figure 12.} Position of a text with respect to a line.}
{
% First segment
\figwritelw 1,2:West(4pt) \figwritele 1,2:East(4pt)
\figwriteln 1,2:North(4pt)\figwritels 1,2:South(4pt)
% Second segment
\figwritelw 3,4:West(4pt) \figwritele 3,4:East(4pt)
\figwriteln 3,4:North(4pt)\figwritels 3,4:South(4pt)
% Third segment
\figwriteln 5,6:North(4pt,0.8)\figwritels 5,6:South(4pt,0.1)
}
\centerline{\box\figBoxA}
```

When no text is specified, the text written by default depends on the position. The text is:
- the text joined to the first end point of each segment (P_i) for west position,
- the text joined to the second end point of each segment (P_{i+1}) for east position,
- the length of the segment (in user unit) for north or south position. The number of decimals printed is 2 by default and can be changed by the rounding parameter of the \figset write macro.
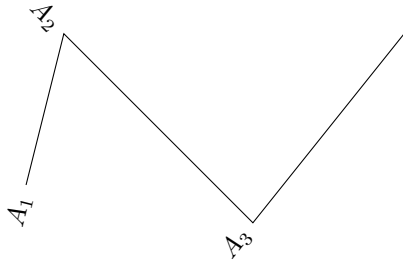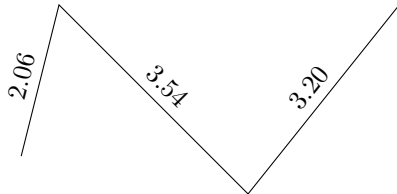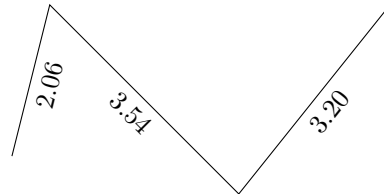
This is shown on the following figures:

Figure 13. West
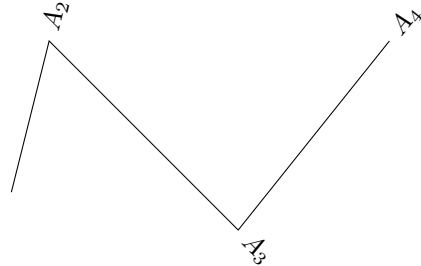
Figure 14. East

Figure 15. North

Figure 16. South

The text of the program that produces these figures is the following:

```
% 1.  Definition of characteristic points
\figinit{cm}
\figpt 1:(1,1)
\figpt 2:(1.5,3)
\figpt 3:(4,0.5)
\figpt 4:(6,3)
% 2.  Creation of the graphical file
\def\MyGrFile{brokenline.gra}
\figdrawbegin{\MyGrFile}
\figdrawline[1,2,3,4]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{{\bf Figure 13.} West}
 {\figinsert{\MyGrFile}\figwritelw 1,2,3,4:(4pt)}
\figvisu{\figBoxB}{{\bf Figure 14.} East}
 {\figinsert{\MyGrFile}\figwritele 1,2,3,4:(4pt)}
\centerline{\box\figBoxA\hfil\box\figBoxB}
%
\figvisu{\figBoxA}{{\bf Figure 15.} North}
 {\figinsert{\MyGrFile}\figwriteln 1,2,3,4:(4pt)}
\figvisu{\figBoxB}{{\bf Figure 16.} South}
 {\figinsert{\MyGrFile}\figwritels 1,2,3,4:(4pt)}
\bigskip
\centerline{\box\figBoxA\hfil\box\figBoxB}
```

The typical use of this macro is to write a legend along an axis or an arrow ; see figure 6 for an example. Here with the same line, a more realistic use would probably be:
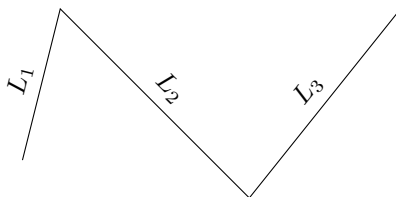
Figure 17.

which is obtained by the following code:

```
% 3.  Writing text on the figure
\figvisu{\figBoxA}{{\bf Figure 17.}}{\figinsert{\MyGrFile}
\figwriteln 1,2 :$L_1$(4pt)
\figwriteln 2,3 :$L_2$(4pt)
\figwriteln 3,4 :$L_3$(4pt)
}
\centerline{\box\figBoxA}
```

We finally mention the specific macro

```
\figwritedim Pt1, Pt2 :Text(Distance)
```

whose purpose is to write a dimension over a figure, generally a plan. Its use is similar to the macros just described that write on lines. Given a segment [Pt1, Pt2], this macro writes the text Text above or under the segment at the specified distance Distance. The text is centered with respect to the segment and is written under the segment if Distance < 0, above the segment otherwise. We could also say that if Pt1 and Pt2 represent west and east, the text is written at the south of the segment if Distance < 0, at the north otherwise. If Text is empty, the value of the distance between Pt1 and Pt2 is written. In this case, the written value can be rounded with the help of the macro **\figset write**(rounding=*NDec*) by setting the number of decimals to be printed. This feature was introduced in the **Text argument** subsection. The following figure and the associated program shows a typical use of this macro, along with its companion macro **\figdrawdim** which is described in the section **Arrow**.
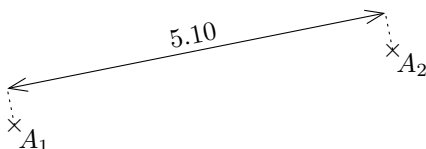
Figure 18.

```
% 1.  Definition of characteristic points
\figinit{cm}
\figpt 1:(0,0)\figpt 2:(5,1)
% 2.  Creation of the graphical file
\figdrawbegin{}
\figdrawdim 1,2(0.5) % Draw a double arrow at 0.5cm above the segment
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 18.}{%
\figset write(mark=$\times$)\figwritese 1,2:(2pt) % Show the two points
\figwritedim 1,2:(0.5)} % Write the length of the segment over the double arrow
\centerline{\box\figBoxA}
```

<center>§**6. Graphical files handling**</center>

Preliminary advice:

 The simplest way to use this macro package is to call the macro `\figdrawbegin` without any argument, as shown in most of the examples in this documentation. The figure will then be automatically inserted by the following call to the macro `\figvisu` and any modification will be taken into account on next compilation of the document, as expected.

This situation corresponds to what the user needs most of the time. In the following, we find a more detailed description of some particular features that are sometimes useful.

The way the graphical files to be included into the document are handled by Fig4TeX depends on:

 - the origin of the file: internal (created by Fig4TeX itself) or external (created by any other software),

 - the typesetting mode: DVI or PDF mode.

Definition:

 If the document is typeset with the command `tex` *or* `latex`*, the output file is a DVI file. This is called the DVI mode.*

 If the document is typeset with the command `pdftex` *or* `pdflatex`*, the output file is a PDF file. This is called the PDF mode.*

1. Internal file

The macro to use to include the file in the document is `\figinsert`.

The file name given as argument to `\figdrawbegin` is mandatory only if there are several references to the *same* figure in the document. For example, if the file is referenced twice, we'll get the following statements :

```
\figdrawbegin{MyGrFile}
 ...
\figvisu ... {\figinsert{MyGrFile}
 ...
}
 ...
\figvisu ... {\figinsert{MyGrFile}
 ...
}
```

When a file name is specified, the file is subject to the update process (see below). We recall that its name must not begin with a digit (see section **Writing text on the figure**).

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

2. External file

The macro to use to include the file in the document is `\figinsertE` (see section **Text annotation**).

	DVI mode	PDF mode
Kind of file that can be included by Fig4TeX	PostScript	JBIG2, JPEG, PDF, PNG

In PDF mode, PostScript files *cannot* be included, because they are ignored by the `pdftex` engine (except those created by MetaPost). Therefore, to be used in this mode, a PostScript file must be converted into one of the other formats. There are valuable hints about this on the pdfTeX home page (`http://www.pdftex.org`).

Automatic file name supply

When no file name is given as argument to `\figdrawbegin`, a default file name is supplied by Fig4TEX. The corresponding graphical instructions are then automatically included by the next call to `\figvisu`, without even calling `\figinsert{}`. This automatic inclusion refer to the *last* call to `\figdrawbegin{}`, which is the most common situation. Inside the group of commands of `\figvisu`, a call to `\figinsert` is taken into account only if it has an argument, since without argument the graphical file just created is automatically included.

It must be noticed that this graphical file is not subject to the update process: it will be *overwritten* on next compilation of the document, and even on next call to `\figdrawbegin{}` in PDF mode.

In PDF mode, the file used to hold the graphical information is a scratch file whose name is `\jobnameGI.anx`. One file is sufficient, even if there are several figures, because the graphical instructions are put on the fly into the final PDF output file, at compilation time. At the end of the compilation, this scratch file contains the graphical information related to the last figure created.

In DVI mode, a new file is used for each occurrence of `\figdrawbegin{}`. Indeed, since the DVI output file stores only the name of the graphical file, each of them must be kept until the DVI file is converted into PostScript or PDF. The name of the file is `\jobnameGIk.anx` for the kth figure in the document.

In addition to these files, another temporary file whose name is `\jobname.anx` is created (in fact, this is true only if at least one graphical file is created). This file is empty at the end of the compilation. All these `.anx` files *can be removed safely* since they are automatically recreated on next compilation.

Update process

By default, every graphical file is updated or recreated at each compilation of the document.

Remark: This behavior is opposite to the choice made in the previous versions of Fig4TEX, which was mainly motivated to make compilations fast. Nowadays, computers are very powerful so that extra work is acceptable. Moreover, most users expect this behavior.

However, it is sometimes desirable to avoid the re-computation of the macros between `\figdrawbegin` and `\figdrawend` and thus make the compilation of the document run faster, which may be sensible with complicated figures involving a lot of geometric constructions.

To achieve this, there are two conditions: the update process must be disabled before step 2 and a filename must be provided as an argument to `\figdrawbegin`. Thus, the typical statements are:

$$\texttt{\textbackslash figset(update=no)}$$
$$\texttt{\textbackslash figdrawbegin\{MyGrFile\}}$$

In this situation, the graphical file `MyGrFile` is never updated by subsequent compilations of the document. However, if the file happen to be removed, it is automatically recreated.

The update mode can also be disabled at the document level by writing the command `\figsetdefault(update=no)` after the loading command of Fig4TEX. Every graphical file in the document will then be preserved by the compilations, except those eventually forced locally by `\figset(update=yes)`.

This "no update" mode is obviously to be used when the figures need no more modifications. It may be useful for complicated figures that require many computations. It must be noticed that, in this case, since the points defined between `\figdrawbegin` and `\figdrawend` are not computed, they should not be used as attach points for text during step 3 (macro `\figvisu`).

Remark: The macros `\figset` and `\figsetdefault` are used to set other graphical attributes; they are described in the section **Setting graphical attributes**.

1. Introduction

Given a pre-existing figure, such as an image, it may be useful to *write some text* at a precise place upon it, or superpose some *drawing*, or even an another *image*. These actions can be done rather easily with Fig4TEX.

The graphical file containing the figure must first be included in the document, with the help of one of the macros \figinsert or \figinsertE:

- if the graphical file was created by Fig4TEX, then the macro \figinsert *must* be used ;
- if the graphical file was created by any other software, which is mainly the case we are dealing with in this section, then the macro \figinsertE *must* be used (the last letter E stands for External). This macro also allows to scale the image so that it is displayed at the appropriate dimension.

We recall that the kind of file that can be included depends on the typeset mode (see the previous section **Graphical files handling**).

2. Text annotation

Here, we answer the question: "How to write some text on an existing figure ?"
Any unit can be chosen. For the demonstration, we consider the triangle drawn on figure 1. We assume that the corresponding graphical information has been stored in the file `Fig1.gra` (we intentionally use this generic filename extension), as if we had written \figdrawbegin{Fig1.gra}. 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{\figBoxA}{\bf Figure 19}{
\figinsertE{Fig1.gra}
\figpt 1:(4.5,3)
\figwrite [1]{\vbox{\hbox{Something at the right}\hbox{of the triangle}}}
}
\centerline{\box\figBoxA}
```

Thus, we get the result shown on the figure 19:

Something at the right
of the triangle
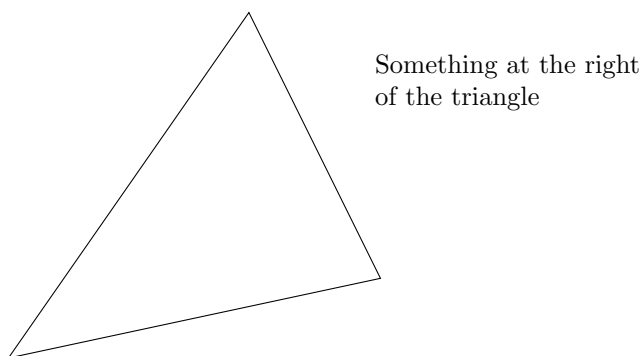
Figure 19

Notice that if the file to be included in the document is a PostScript file, it must be in fact an *encapsulated* PostScript file because the macro \figinsertE needs the bounding box of the figure to work correctly. 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 450, and `Ymin` and `Ymax` range between 0 and 700 (which corresponds roughly to maximum values for a A4 page). It must be noticed that some graphical softwares
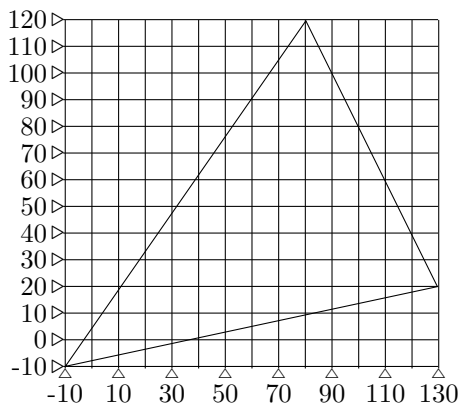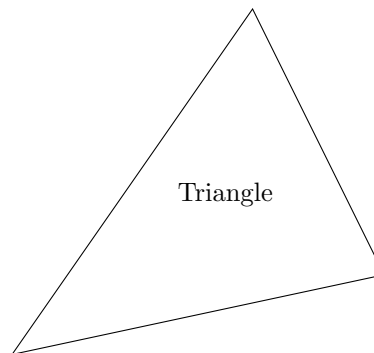
Figure 20

Figure 21

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 problem is the same as finding the coordinates of a particular point on the figure. In the previous example, the coordinates of the attach point of the text need not to be set very accurately. However, it sometimes happens that one wants to refer to a particular point on the figure. Except in the rare case where the user explicitly knows the coordinates of the point, the problem is precisely to obtain easily these coordinates.

Some previewers may help, if they show the coordinates of the cursor. This is the case for example with `ghostview`, a PostScript viewer available on some operating systems. The position of the cursor is given in PostScript unit, so the argument to use for `\figinit` must be `bp`.

Another solution, provided by Fig4TEX, which works with any graphical file and not only with a PostScript file, is to use the macro `\figscan FileName(HX,HY)` which draws a rectangular grid with horizontal step `HX` and vertical step `HY`. Numerical values corresponding to the size of the figure described in the file `FileName` are printed, at least every 20 bp horizontally and every 10 bp vertically to avoid overprinting. The values of the steps are given in PostScript unit (`bp`) and must 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 `\figinsertE`, for example to set the coordinates of the attach point or adjust the bounding box of a PostScript file.

For example, if we want to write the word "Triangle" inside the triangle, we need to determine the coordinates of the attach point. For that, we can do the following.

- First, we compile a file containing the commands (`\MyImage` is defined because it is referenced twice):

```
\def\MyImage{Fig1.gra}
\figvisu{\figBoxA}{\bf Figure 20}{
 \figinsertE{\MyImage}
 \figscan \MyImage (10,10) }
\centerline{\box\figBoxA}
```

- Then, we use a previewer to look at the result shown on the figure 20: the grid and the drawing are printed together. We are now able to fix the coordinates of the point: we choose (70,50).

- At last, we slightly modify the previous file: set the unit to `bp`, remove the call to `\figscan`, create the point and center the text on it, which leads to the following commands:

```
\figinit{bp}
\def\MyImage{Fig1.gra}
\figvisu{\figBoxA}{\bf Figure 21}{
 \figinsertE{\MyImage}
 \figpt 1:(70,50) \figwritec[1]{Triangle} }
\centerline{\box\figBoxA}
```

The result is shown on the figure 21.

28

Scaling factor

The figure to be inserted in the above example has a dimension that makes it directly useable in the document. But, in particular when an image is inserted, it should generally be scaled to fit inside the page. In order to control the scale of the figure to be inserted in the page, the macro `\figinsertE` has a second argument ; its full prototype is:

`\figinsertE{FileName, ScaleFactor}`

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

Remarks:

- The macro `\figinsert` has the same arguments and the same behaviour. But although this macro works with graphical files created by Fig4TEX, it is better to control the scale with the macro `\figinit` because the scale factor set by `\figinsert` apply to each component of the figure, e.g. the line width, which may lead to a disgracious appearance.
- When several scale factors are specified, the macro `\figscan` works properly only if it appears *immediately after* the corresponding call to `\figinsertE`.

An example of this feature is given on figure 22 (and in next section).
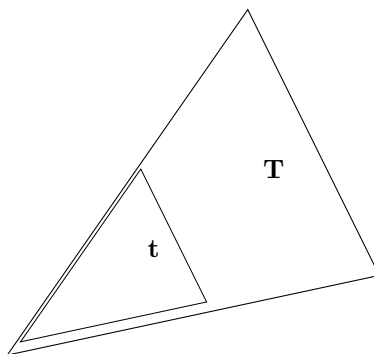
Figure 22

The same graphical file, `Fig1.gra`, 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 smaller one. The program that produces it is :

```
\figinit{bp}
\def\ScFactor{0.5}
\def\MyImage{Fig1.gra}
\figvisu{\figBoxA}{\bf Figure 22}{
 \figinsertE{\MyImage}
 \figpt 1:(90,60) \figwritec[1]{\bf T}
 % Idem after scaling:
 \figinsertE{\MyImage, \ScFactor}
 \figpt 0:(0,0) \figpthom 1:=1/0,\ScFactor/ \figwritec[1]{\bf t}
}
\centerline{\box\figBoxA}
```

3. Graphical annotation

Drawing on a pre-existing figure or an image requires to create a graphical file that will be manually inserted *after* the image. The procedure is similar to what has been explained in the previous section (see figures 20 and 21). As an example, we propose to highlight the slopes in a mountain landscape with two straight lines. The image is supposed to be stored in a file called `mountain.jpg`.

- First step: chose a scale factor so that the image is displayed at the wanted size and display a grid over it with the help of `\figscan`. This is achieved by the small following program, where 0.8 is chosen as scale factor. The result is shown on the figure 23.

```
% Display a grid over the image
\def\MyImage{mountain.jpg}
\figvisu{\figBoxA}{\bf Figure 23}{
 \figinsertE{\MyImage,0.8}
 \figscan{\MyImage}(20,20)
}
\centerline{\box\figBoxA}
```
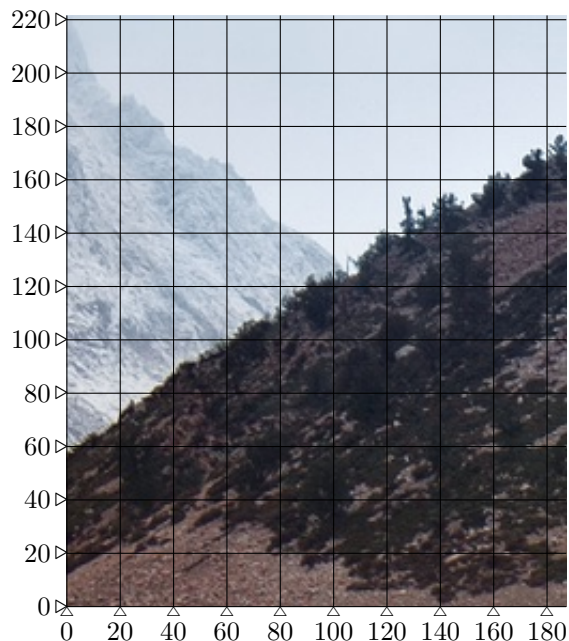

Figure 23

- Second step: chose the coordinates of the points defining the segments (or more generally any suitable drawing).
- Last step (modify the previous program): set the unit to `bp`, create the points and the graphical file (`slopes.gra`), replace the call to `\figscan` by the insertion of this file *after* the image using the macro `\figinsert` ; this leads to the following program whose result is shown on the figure 24:

```
% 1.  Definition of characteristic points
\figinit{bp}
\figpt 1:(32,80)  \figpt 2:(168,160) % red line
\figpt 3:(24,192) \figpt 4:(104,128) % blue line
% 2.  Creation of the graphical file
\def\MyGrFile{slopes.gra}
\figdrawbegin{\MyGrFile}
\figset(color=\Redrgb, width=2) \figdrawline[1,2]
\figset(color=\Bluergb, dash=2) \figdrawline[3,4]
\figdrawend
% 3.  Display annotated image
\def\MyImage{mountain.jpg}
\figvisu{\figBoxA}{\bf Figure 24}{
 \figinsertE{\MyImage,0.8}
 \figinsert{\MyGrFile}
}
\centerline{\box\figBoxA}
```


Figure 24

30

Remarks:

- Note that the graphical file created by Fig4TeX should be named (by giving an argument to `\figdraw-begin`) since it must be included *after* the image. Otherwise, the file would be automatically inserted before the image and covered with it (and thus invisible !).

- The lower left corner of the image is the origin of the axes. Thus, we could as well determine the coordinates of the points by measuring distances with a graduated ruler, without forgetting to set the appropriate unit which would probably be `mm`, `cm` or `in`, instead of `bp` (in this case, the scaling factor of the image would also probably be equal to 1).

- If, after valuable efforts to draw something over the image, the dimension of the image must be modified, the points would not be anymore at correct locations. Redefining them would be tedious, but above all unuseful since there is a quick solution: apply new scaling factors to *both* files.

  For example, if we want to diplay the image at scale 0.7 instead of 0.8 as done above, we have to apply a new scaling factor equal to 0.875 since $0.7 = 0.8 \times 0.875$, recalling that by default this factor is equal to 1. Thus, by modifying accordingly the last part of the program above:

```
\figvisu{\figBoxA}{\bf Figure 25}{
 \figinsertE{\MyImage,0.7}   % 0.7 = 0.8 x 0.875
 \figinsert{\MyGrFile,0.875} % 0.875 instead of 1 (default value)
}
\centerline{\box\figBoxA}
```

we get the following figure:

Figure 25

Interestingly, we can notice that the width of the lines is scaled, which is suitable in this case. Indeed, alternatively, we could have used the factor to modify the unit (`\figinit{0.875bp}`), which would have changed the graphical file, that should then have to be inserted without scaling (by saying `\figinsert{\MyGrFile}`) leaving the lines width unchanged and leading to a worse aspect.

As mentionned in the introduction of this chapter, displaying an image over another one used as background is a feature that is rather easy to achieve. To illustrate that, we propose to brighten this gloomy mountain lanscape by adding a yellow ink spot supposed to represent the sun. We thus prepare an small image stored in the file `sun.png`. The lower left corner of the image is the handle which will be used to locate the image over the background. On the following figure, the dashed line corresponds to the border of the image.
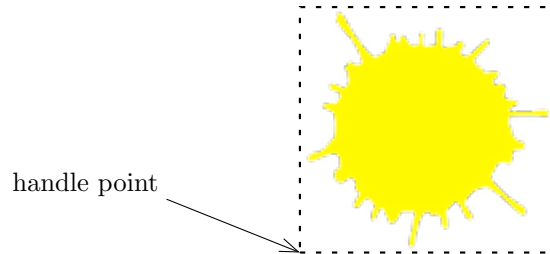
handle point

In order to display it on the background image, the procedure makes use of tools that have already been encountered: choose the coordinates of the point on the background image that will coincide with this handle point and use the `\figwrite` macro whose text argument is repaced by the call to `\figinsertE`. This leads to the following program whose result is shown on the figure 27:

```
\figinit{bp}
% Display an image over an image
\def\MyImage{mountain.jpg}
\def\MySun{sun.png}
\figvisu{\figBoxA}{{\bf Figure 27.}
Superposition of two images (PDF mode)}{
 \figinsertE{\MyImage,0.8}
 \figpt 0:(100,170)
 \figwrite [0]{\figinsertE{\MySun,0.5}}
}
\centerline{\box\figBoxA}
```
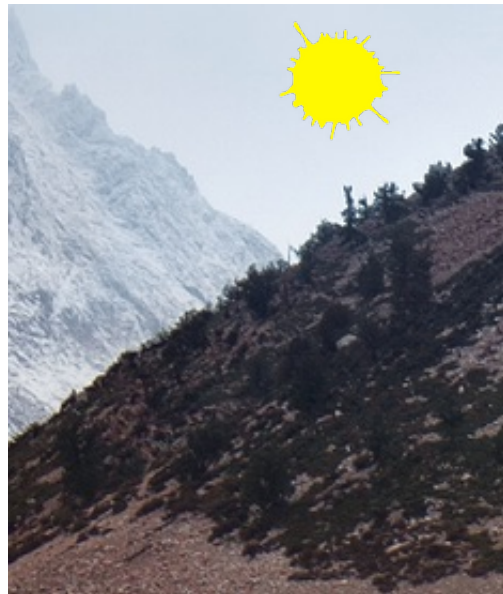
Figure 27. Superposition of two images (PDF mode)

In DVI mode, only PostScript files can be inserted, so in the previous example the yellow spot would appear inside a white square. Thus, except in the case where the two images have the same color background, this feature is most useful in PDF mode, since the transparency background of the last image allow a nice superposition. Here, in PDF mode, we use a PNG file with a transparent background.

Serious people may find this example futile. Putting a signature over an existing document would have admittedly been a more worthy example. It should be noted that this particular action can be done in a more straightforward way using the Fill & Sign tool of Adobe Acrobat Reader.

1. Introduction

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

```
\figset        keyword (attribute1=value1, attribute2=value2,...)
\figsetdefault keyword (attribute1=value1, attribute2=value2,...)
\figreset{keyword1, keyword2,...}
```

The `keyword` argument refers to attributes that are logically grouped together. There are two kinds of attributes: general attributes that may concern every drawing macro (like the color), and specific attributes that concern only a particular drawing macro (like the roundness of a curve).

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

To get back to the default value, one can use `\figset`, called with the generic value `default`, for example `\figset(width=default)`, or `\figreset` which resets to their default value all the attributes related to a particular keyword, for example `\figreset{general}` .

Some specific attributes have the same nature as general attributes (line attributes and color). Their default value is the *current* value of the corresponding general attribute. For this reason, they are called Dynamic Default Value attributes (DDV): the default value varies according to the changes made to the general attribute. Indeed, this "follow context" feature is most often convenient in practice ; however, any specific attribute can be assigned a particular value with the macro `\figset`, which "locks" this attribute and suspends the dynamic behavior.

Moreover, the macro `\figsetdefault` allows the user to change the default values (of every non DDV attribute of course). Indeed, the "factory" choice may not fit the needs of the user and this macro provides a way to customize the way the package is used without having to modify the macro file. This macro is intended to be called at the beginning of the document, after the macro package has been imported. The default values are changed up to the end of the document or up to next call to `\figsetdefault`, and are taken into account by every macro that refers to the default values, which are `\figinit`, `\figdrawbegin` and `\figreset`. For example, the following lines:

```
\input fig4tex.tex
\figsetdefault(width=1)
\figsetdefault arrowhead(angle=10)
```

now impose that, by default, the line width is 1 bp and the arrowhead half-angle is 10 degrees.

The list of available keywords is the following:
- *no* keyword or keyword = `general` which deals with the so-called general attributes, i.e. applicable unless otherwise stated in a specific context with one of the following keywords,
- `altitude` which deals with the altitude attributes (see `\figdrawaltitude`),
- `arrowhead` which deals with the arrowhead attributes (see the `\figdrawarrow*` family),
- `curve` which deals with the attributes of a curve (see `\figdrawcurve`),
- `general` which deals with the general attributes,
- `flowchart` which deals with the flow chart attributes (see the `\figdrawfc*` family),
- `mesh` which deals with the attributes of a mesh (see `\figdrawmesh`),
- `sign` which deals with the attributes of a sign or symbol (see the `\figdrawsign*` family),
- `trimesh` which deals with the attributes of a mesh (see `\figdrawtrimesh`).

Then, here follow the lists of the attributes relative to each keyword. Each attribute is followed by the possible values it can take. DDV indicates that its default value is dynamic, i.e. follows the last value given by the `general` keyword ; due to the DDV definition, such attributes should be used inside a `\figdrawbegin`–`\figdrawend` group to be effective:

- *no* keyword or keyword = `general`
    - . `alpha` = transparency level
    - . `cap` = `butt`, `round` or `square`
    - . `color` = color definition
    - . `dash` = index or dash pattern
    - . `fillmode` = `yes`/`no`
    - . `join` = `miter` ($\vee$), `round` ($\smile$) or `bevel` ($\backslash\_/$)
    - . `update` = `yes`/`no`
    - . `width` = dimension in PostScript unit
- keyword = `altitude`
    - . `blcolor` = color definition (DDV)
    - . `bldash` = index or dash pattern (DDV)
    - . `blwidth` = dimension in PostScript units (DDV)
    - . `sqcolor` = color definition (DDV)
    - . `sqdash` = index or dash pattern (DDV)
    - . `sqwidth` = dimension in PostScript units (DDV)
- keyword = `arrowhead`
    - . `angle` = real in degrees
    - . `fillmode` = `yes`/`no`
    - . `length` = real in user unit
    - . `out` = `yes`/`no`
    - . `ratio` = real in $[0, 1]$
- keyword = `curve`
    - . `roundness` = real usually in $[0, 0.5]$
- keyword = `flowchart`
    - . `arrowposition` = real in $[0, 1]$
    - . `arrowrefpt` = `start`/`end`
    - . `bgcolor` = color definition (frame background)
    - . `line` = `polygon`/`curve`
    - . `padding` = real in user unit (set both `xpadding` and `ypadding`)
    - . `radius` = positive real in user unit
    - . `shape` = `circle`, `ellipse`, `lozenge` or `rectangle`
    - . `thickcolor` = color definition (of the thickness of the frame)
    - . `thickness` = real in user unit
    - . `xpadding` = real in user unit
    - . `ypadding` = real in user unit
- keyword = `mesh`
    - . `diag` = integer in {-1, 0, 1}
    - . `color` = color definition (DDV)
    - . `dash` = index or dash pattern (DDV)
    - . `width` = dimension in PostScript units (DDV)
- keyword = `sign`
    - . `color` = color definition (DDV)
    - . `dash` = index or dash pattern
    - . `dimension` = positive real in user unit
    - . `fillmode` = `yes`/`no`
    - . `join` = miter (V), round (U) or bevel ($\backslash\_/$)
    - . `shape` = `circle`, `diamond`, `nabla`, `square`, `star` or `triangle`
    - . `width` = dimension in PostScript units
- keyword = `trimesh`
    - . `color` = color definition (DDV)
    - . `dash` = index or dash pattern (DDV)
    - . `width` = dimension in PostScript units (DDV)

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 \figsetdefault is not called, so the default values used for the attributes in all the examples are the "factory" default values.

The update attribute has been described in the **Graphical files handling** section (**Update process** subsection). Except this one, the general attributes have an action on the way a line or a patch is drawn. They are described in the next subsections. However, the fillmode attribute is presented later in the **Filled area, transparency** subsection, after some graphical macros have been introduced, in order to show its effect in a more demonstrative way. The same apply for the alpha attribute which is generally used when patches are drawn.

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 \figset(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, we get

———————————	with \figset(width=default)
———————————	with \figset(width=1)
———————————	with \figset(width=1.5)
———————————	with \figset(width=2)

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

Remark: The transparency attribute also apply for lines and is particularly relevant when the lines are thick (see **Filled area, transparency** subsection).

• Line style

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

$$\text{\figset(dash=}Index)$$
$$\text{\figset(dash=}Pattern)$$

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

———————————	with \figset(dash=default)
— — — — — — — — — —	with \figset(dash=2)
– – – – – – – – – – – – – –	with \figset(dash=3)
- - - - - - - - - - - - - - - - -	with \figset(dash=4)
··	with \figset(dash=5)
- - - - - - - - - - - - - - -	with \figset(dash=6)
- - - - - - - - - - - -	with \figset(dash=7)
- - - - - - - - - - - - - -	with \figset(dash=8)
- · - · - · - · - · - · - · ·	with \figset(dash=9)
— · - · — · - · — · - — · - -	with \figset(dash=10)

The second form refers to a pattern which is a sequence of integers separated with white spaces. The general form is $Dash_1\ Space_1 \ldots Dash_n\ Space_n\ Offset$ where each value is a dimension given in PostScript points (bp), $Dash_i$ corresponding to the length of a dash, $Space_i$ 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 \figset(dash=2) is equivalent to \figset(dash=6 2), we obtain a better look using the pattern form of the macro as shown below:

————————————— with \figset(dash=2)
————————————— with \figset(dash=6 2) (same as previous)
————————————— with \figset(dash=6 2 2)

● 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.

join=miter   join=round   join=bevel

● Line cap

This attribute establishes the shape to be put at the ends of a straight line. Rarely used, its effect is mainly visible with wide lines. The three possible values are shown on the following figure, where the theoretical end points are those of the thin white line in the middle of the black wide one. The default value is butt, meaning that the line stops at the true location. The two other possibilities allow to add a circular or squared cap. The circular cap is especially useful to smooth the rendering when several segments meet at the same point.

cap=butt   cap=round   cap=square

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 instruction:

\figset(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 *cmyk* 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 *cmyk* 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 `\figset(color=\Redrgb)` or `\figset(color=1 0 0)`, provided the macro `\Redrgb` has been defined. Note that the coordinates are separated with blank spaces and range between 0 and 1. The basic colors have been predefined in each color model, along with some other ones :

- in *cmyk* :

⬛ `\Blackcmyk,`	⬜ `\Whitecmyk,`	🟫 `\Graycmyk,`
🟦 `\Cyancmyk,`	🟪 `\Magentacmyk,`	🟨 `\Yellowcmyk,`
🟥 `\Redcmyk,`	🟩 `\Greencmyk,`	🟦 `\Bluecmyk,`
🟫 `\Browncmyk,`	🟥 `\BrickRedcmyk,`	🟩 `\ForestGreencmyk,`
🟥 `\Marooncmyk,`	🟧 `\Orangecmyk,`	🟦 `\RoyalBluecmyk,`
🟨 `\Goldenrodcmyk,`	🟪 `\Purplecmyk,`	🟪 `\Pinkcmyk,`	🟪 `\Violetcmyk,`

- in *rgb* :

⬛ `\Blackrgb,`	⬜ `\Whitergb,`	🟫 `\Grayrgb,`
🟦 `\Cyanrgb,`	🟪 `\Magentargb,`	🟨 `\Yellowrgb,`
🟥 `\Redrgb,`	🟩 `\Greenrgb,`	🟦 `\Bluergb,`
🟧 `\Chocolatergb,`	🟥 `\Firebrickrgb,`	🟩 `\ForestGreenrgb,`
🟥 `\Maroonrgb,`	🟧 `\DarkOrangergb,`	🟦 `\RoyalBluergb,`
🟨 `\Goldrgb,`	🟪 `\HotPinkrgb,`	🟪 `\Pinkrgb,`	🟫 `\DarkGoldenrodrgb.`

Remark: The color coordinates are taken from different sources and the associated names have been preserved. A color may render differently in the two color models: this is the case for `Cyan` 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. The color model is automatically deduced from the coordinates. For example, orange is defined in *cmyk* model by saying `\def\Orangecmyk{0 0.61 0.87 0}` . The suffix `cmyk`, or `rgb`, is not mandatory ; one could have chosen to call it simply `\Orange`.

Finally, as already mentionned in the previous **Introduction** subsection, some high level macros use specific attributes, among which can be found the color. For example, setting the background color of the frames in a flowchart in *gray* tone, *cmyk* or *rgb* color code can be achieved by saying `\figset flowchart(bgcolor=`*Color Definition*`)` which works the same way as `\figset(color=`*Color Definition*`)`.

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 very 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 (see the section **Text annotation**).

2. Arc

We already saw the macro \figdrawcirc 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:

> \figdrawarccirc Center ; Radius (Ang1,Ang2)
> \figdrawarcell 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 \figdrawarccirc 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 \figdrawarcell 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:

> \figdrawarccircP Center ; Radius [Pt1,Pt2]
> \figdrawarcellPP 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 \figdrawarcellPP 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 $\overrightarrow{CA_1} \times \overrightarrow{CA_2}$.

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

> \figdrawarcellPA Center,PtAxis1,PtAxis2 (Ang1, Ang2)

It behaves exactly like \figdrawarcellPP except that it allows to specify the portion of the arc using the parametrization angles Ang1 and Ang2.
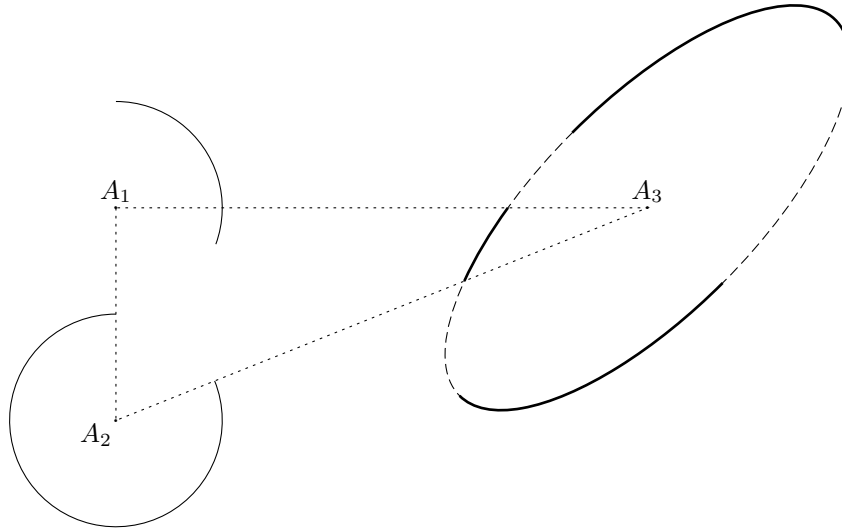
Figure 33

Figure 33 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 `\figdrawarcell`.

```
% 1.  Definition of characteristic points
\figinit{pt}
\def\haxis{100}\def\vaxis{40}
\def\startangle{-20}\def\stopangle{90}\def\inclin{45}
\figpt 1:(-10,10)\figpttraC 2:=1/0,-80/\figpttraC 3:=1/200,0/
\figptell 4::  3 ; \haxis,\vaxis( 0, \inclin) % End point of major axis
\figptell 5::  3 ; \haxis,\vaxis(90, \inclin) % End point of minor axis
% 2.  Creation of the graphical file
\figdrawbegin{}
\figdrawarccirc 1;\vaxis(\startangle,\stopangle)\figdrawarccircP 2;\vaxis[1,3]
\figset(dash=3)\figdrawarcell 3 ; \haxis,\vaxis(0,360, \inclin)
\figset(dash=default,width=1)
\figdrawarcell 3 ; \haxis,\vaxis(\startangle,\stopangle, \inclin)
\figdrawarcellPP 3,4,5[1,2]\figdrawarcellPA 3,4,5(-180,-90)
\figset(width=default,dash=5)\figdrawline[1,2,3,1]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 33}{
\figset write(mark=.)\figwriten 1,3:(2)\figwritesw 2:(2)
}
\centerline{\box\figBoxA}
```

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 $\rho = ||\overrightarrow{AC}||$. The two circles $(A; \rho)$ and $(C; R_x)$ intersect at points I' and J'. The lines

39

(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.
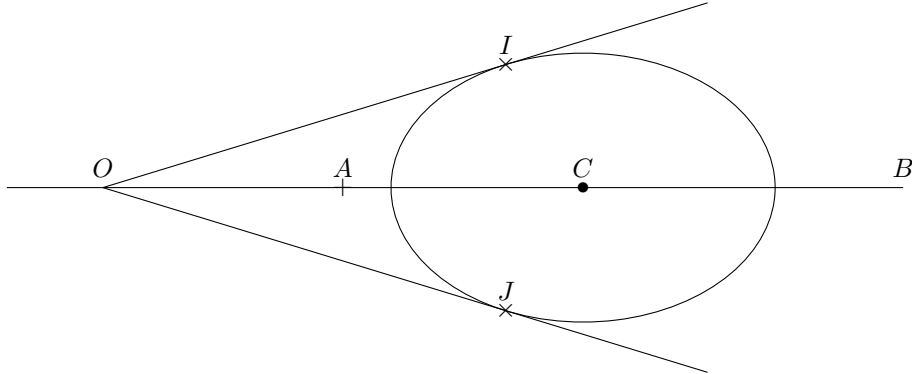

Figure 34

```
% 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]\figget distance=\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]
\figget angle=\ThetaI[1,11,3]
\figget angle=\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 graphical file
\figdrawbegin{}
\figdrawarcell 1;\Rx,\Ry (0,360,0)
\figdrawline[10,11]\figdrawline[0,13]\figdrawline[0,14]
\figdrawend
% 3.  Writing text on the figure
\def\dist{4pt}
\figvisu{\figBoxA}{\bf Figure 34}{
\figwriten 0:(\dist)\figwriten 11:(\dist)
\figset write(mark=+)\figwriten 2:(\dist)
\figset write(mark=$\figBullet$)\figwriten 1:(\dist)
\figset write(mark=$\times$)\figwriten 3,4:(\dist)
}
\centerline{\box\figBoxA}
```

4. Curve

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

$$\text{\figdrawBezier N [Pt\_1, ..., Pt\_\{3N+1\}]}$$

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 and the PDF format only provide cubic Bézier curves, 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,...,P_{3k+1},...,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.
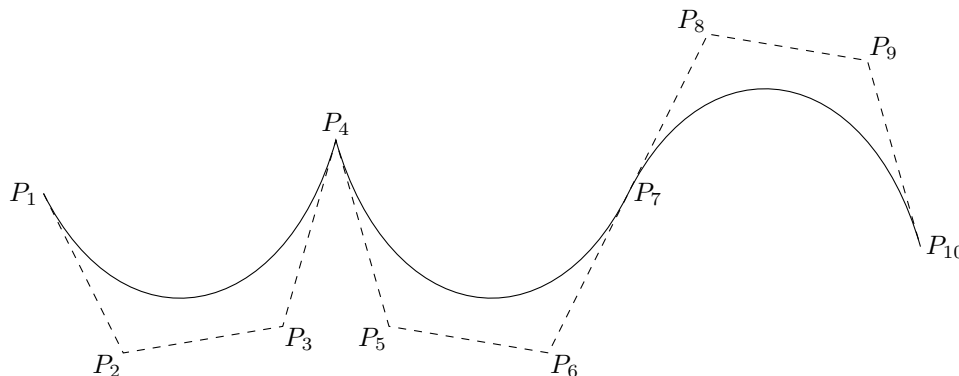

Figure 35

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 graphical file
\figdrawbegin{}
\figdrawBezier 3[1,2,3,4,5,6,7,8,9,10]
\figset(dash=8)\figdrawline[1,2,3,4,5,6,7,8,9,10]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 35}{
\figset write(ptname=$P_{#1}$)
\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\figBoxA}
```

41

It turns out that the macro \figdrawBezier 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

\figdrawcurve [Pt0, Pt1,... ,PtN, PtN+1]

It draws a C^1 curve that *interpolates* the points $P_1, P_2, \ldots, P_N$. The curve consists of $N-1$ Bézier cubic arcs. The direction of the tangent at P_i is given by the vector $\overrightarrow{P_{i-1}P_{i+1}}$, $i = 1, \ldots, N$. 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 \figset 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 \figreset{curve} or \figset curve(roundness=default). 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 \figdrawcurve 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 \figdrawcurve to draw the smooth curve.

The two following figures show what can be done with this macro.
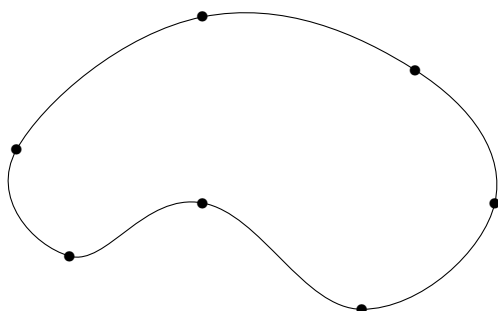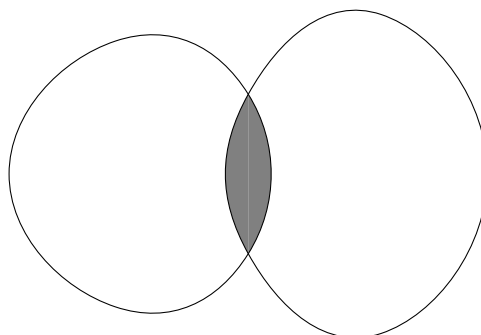
Figure 36   **Figure 37**

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 graphical file
\figdrawbegin{}
\figdrawcurve[1,2,3,4,5,6,7,1,2,3]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 36}{
\figset write(mark=$\figBullet$)
\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, transparency** 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 graphical file
\figdrawbegin{}
\figset curve(roundness=0.23) % More roundness
\figset(fillmode=yes,color=0.5)
\figdrawcurve[11,2,1,10]\figdrawcurve[20,1,2,21] % Better to do this first
\figset(fillmode=no,color=0)
\figdrawcurve[11,2,1,10,12,11,2,1]\figdrawcurve[20,1,2,21,22,20,1,2]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxB}{\bf Figure 37}{}
```

A parabola is a particular case of curve that is often needed. The macro \figdrawarcparab whose full prototype is

$$\figdrawarcparab \ [Pt1,Pt2,Pt3]$$

allows to draw an arc of parabola, defined as a quadratic Bézier curve from P_1 to P_3 whose tangent at P_1 is the line (P_2, P_1) and its tangent at P_3 is the line (P_2, P_3).

To give a short example, by adding the lines

```
\figdrawarcparab[1,2,3]
\figdrawarcparab[2,3,1]
\figdrawarcparab[3,1,2]
```

between \figdrawbegin and \figdrawend on the first example (figure 1), we get the three arcs of parabola inscribed in the triangle, as shown on the following figure.
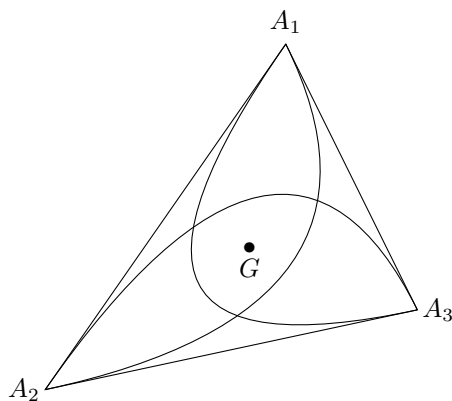

Figure 38

43

5. Filled area, transparency

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 `\figset(color=`*Color Definition*`)`. The filling algorithm is governed by PostScript and PDF own rules. This feature is relevant in particular for non convex polygons. Also, for circular or elliptic arcs, the area is limited by the arc and the corresponding chord.

The macro `\figset(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 33 and 35 and we replace the graphical file definition by

```
\figdrawbegin{}
\figset(fillmode=yes)
\figdrawarccirc 1;\vaxis(\startangle,\stopangle) % Use default color
\figset(color=\Redrgb)\figdrawarcell 3; \haxis,\vaxis(\startangle,\stopangle,\inclin)
\figset(color=\Bluergb)\figdrawarcellPA 3,4,5(-180,-90)
\figdrawend
```

in the first one and by

```
\figdrawbegin{}
\figset(fillmode=yes,color=0.7)
\figdrawBezier 3[1,2,3,4,5,6,7,8,9,10]
\figdrawend
```

in the second one. We then get the two new figures:
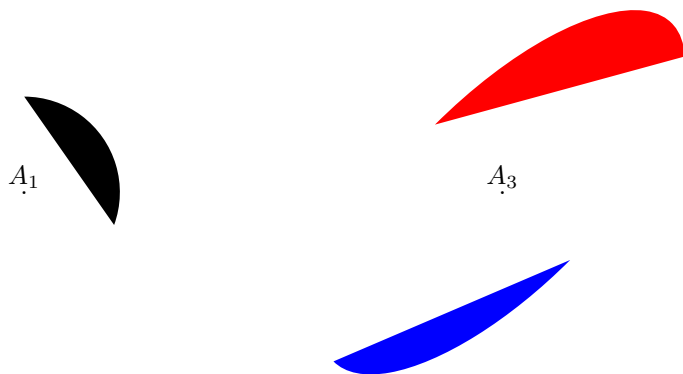
Figure 39

Figure 40

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 shrinked to 90% and 50% respectively.

To put the two figures on the same line, we just filled the box `\figBoxB` for the curve by giving it as first argument to `\figvisu`, and made the classical assembly:

$$\centerline\{\box\figBoxA\hfil\box\figBoxB\}$$

Remarks:
- When the filling mode is active, the macro `\figdrawline` gives the same result whether 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 dvi screen previewers although the result on the paper is correct.

Figure 41

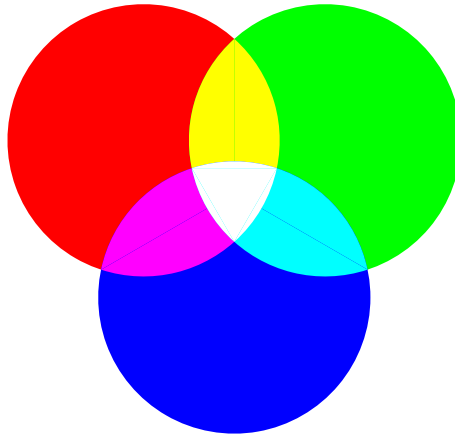The following example is a reproduction of the well known RGB color disks shown on figure 41. The data are the centers (A, B, C) of the three disks and the radius. The points (A, B, C) are the vertices of an equilateral triangle, A and B lying on the X-axis. In order to fill the different areas, we need to compute their limits given by the intersections of two circles, which is required by the macro `\figdrawarccircP`.

The program that produces the figure 41 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 and PDF images are opaque (by default), 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 graphical file
\figdrawbegin{}
\figset(fillmode=yes)
\figset(color=\Redrgb)  \figdrawcirc 1(\R)
\figset(color=\Greenrgb)\figdrawcirc 2(\R)
\figset(color=\Bluergb) \figdrawcirc 3(\R)
\figset(color=\Yellowrgb) \figdrawarccircP1;\R[4,5]  \figdrawarccircP2;\R[5,4]
\figset(color=\Magentargb)\figdrawarccircP1;\R[15,14]\figdrawarccircP3;\R[14,15]
\figset(color=\Cyanrgb)   \figdrawarccircP3;\R[25,24]\figdrawarccircP2;\R[24,25]
\figset(color=\Whitergb)
\figdrawarccircP2;\R[24,4]\figdrawarccircP1;\R[4,14]\figdrawarccircP3;\R[14,24]
\figdrawline[4,14,24]
\figdrawend
% 3.  Inserting the figure
\figvisu{\figBoxA}{\bf Figure 41}{}
\centerline{\box\figBoxA}
```

In PDF mode, the aspect of the painted areas can be modified by the *transparency* attribute. In the coloring models descriptions, this attribute is generally called *alpha*. It is a real number lying between 0 and 1, specifying the transparency level from totally transparent or invisible (0) to opaque (1). This attribute can be set by saying `\figset(alpha=`*value*`)` . The default value is 1.

45

The following figure shows the effect of the transparency attribute on a triangle:
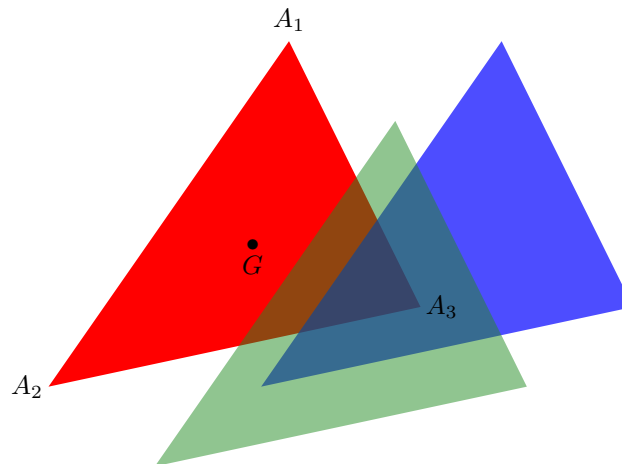

Figure 42. Effect of transparency (in PDF mode only).

Again, we start from the first example (figure 1) and change the lines between \figdrawbegin and \figdrawend by:

```
% 1st triangle
\figset (color=\Redrgb, fill=yes)
\figdrawline[1,2,3,1]
% 2nd triangle
\figvectC 10(80,0)
\figptstra 11=1,2,3/1,10/
\figset (color=\Bluergb, alpha=0.7)
\figdrawline[11,12,13,11]
% 3rd triangle
\figvectC 10(-40,-30)
\figptstra 11=11,12,13/1,10/
\figset (color=\ForestGreenrgb, alpha=0.5)
\figdrawline[11,12,13,11]
```

The original triangle (1,2,3) is first drawn and painted in red (opaque by default). Then, the three points 1, 2, 3 are horizontally translated as points 11, 12, 13 and the second triangle (11, 12, 13) is drawn and filled in blue color with a transparency level of 0.7. At last, the three points 11, 12, 13 are translated and the third triangle (11, 12, 13) is drawn and filled in green color with a transparency level of 0.5.

The legend written of the figure is linked to the first three points and is thus unchanged if we compare with the figure 1. The legend written on the figure is always visible (and opaque). On the contrary, we can observe the effect of the variation of the transparency level on the triangles, leaving the overlapping areas still visible.

The transparency attribute has also an effect on lines (remember to inactivate the filling mode to draw lines only). This is usefull if the lines are thick enough (see line width in section **Line attributes**).
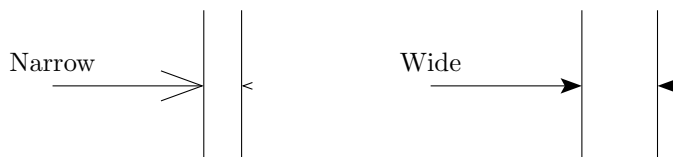
6. Arrow

Straight arrow

Figure 43

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)
\def\VTrans{20}\figvectC \VTrans(0.5,0)
\figptstra 4=1,2,3/1,\VTrans/
\figptcopy10:/0/
\def\LTrans{10}\figpttra 11:=0/\LTrans,\VTrans/
% 2.  Creation of the graphical file
\figdrawbegin{}
\figdrawline[2,3]\figdrawline[5,6]
\figset arrowhead(ratio=0.3)\figdrawarrow [0,1]
\figset arrowhead(out=yes)\figdrawarrowhead [1,4]
\figptstra 0=0,1,2,3,4,5,6/\LTrans,\VTrans/\figptstra 4=4,5,6/1,\VTrans/
\figdrawline[2,3]\figdrawline[5,6]
\figreset{arrowhead}
\figset arrowhead(fillmode=yes,length=0.3)\figdrawarrow[0,1]
\figset arrowhead(out=yes)\figdrawarrowhead[1,4]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 43}{
\figwriten 10:Narrow(0.2)\figwriten 11:Wide(0.2)
}
\centerline{\box\figBoxA}
```

The two main macros are:
$$\texttt{\textbackslash figdrawarrow [Pt1,Pt2]}$$
$$\texttt{\textbackslash figdrawarrowhead [Pt1,Pt2]}$$

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

The appearance of the arrowhead depends on four attributes: the opening angle, the length, the drawing mode and the position. Each of them can be modified using the macro `\figset` (see below). To reset at once all the attributes to their default values, the simplest is just to say `\figreset{arrowhead}`.

• opening angle: `\figset arrowhead(angle=`*value*`)`
  sets the arrowhead half-angle to *value* (in degrees). The default value is 20 degrees, that can be set by saying `\figset arrowhead(angle=default)`.

• length: Precisely, we speak of the length of each of the two edges of the arrowhead. This attribute can be set by one of the two following sequences which are mutually exclusive; the default is to use the length.

  `\figset arrowhead(length=`*value*`)`
  sets the arrowhead length to *value* (in user unit). The default value is equivalent to a 8.0 pt length, that can be set by saying `\figset arrowhead(length=default)`.

47

`\figset arrowhead(ratio=`*`value`*`)`

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

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

  sets the arrowhead 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 color.

- position: `\figset arrowhead(out=`*`switch`*`)`

  sets the arrowhead "outside" switch to `yes` or `no`. The default value is `no`. If `yes` is chosen, the effect is to draw the arrowhead 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 `\figsetdefault`. However, the length attribute is a particular case because when `\figsetdefault` 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: `\figsetdefault arrowhead(length=4mm)`

The figure 43 shows how to use these macros. The text of the program that produces it is given below it.

Non straight arrow
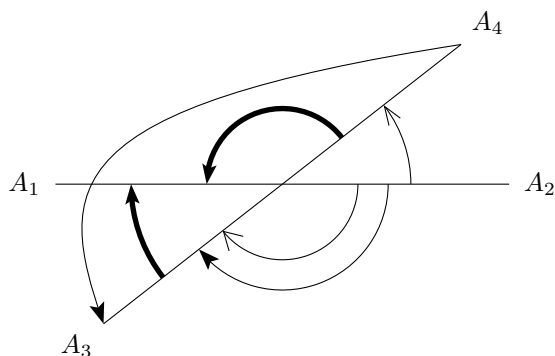
Figure 44

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 graphical file
\figdrawbegin{}
\figdrawline[1,2]\figdrawline[3,4]
\figdrawarrowcirc 0;1.7(0,\rotangle)\figset arrowhead(angle=30)
\figdrawarrowcircP 0;-1[2,3]
\figset arrowhead(angle=default,fillmode=yes)\figset(width=2)
\figdrawarrowcircP 0;1[4,1]\figdrawarrowcircP 0;-2[3,1]
\figset(width=default)\figset arrowhead(length=0.3)
\figdrawarrowcircP 0;-1.4[2,3]\figdrawarrowBezier[4,5,1,3]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 44}{
\figwritew 1:(0.2)\figwritee 2:(0.2)\figwritesw 3:(0.2)\figwritene 4:(0.2)}
\centerline{\box\figBoxA}
```

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

$$\text{\textbackslash figdrawarrowcirc Center ; Radius (Ang1,Ang2)}$$
$$\text{\textbackslash figdrawarrowcircP Center ; Radius [Pt1,Pt2]}$$

The macro `\figdrawarrowcirc` 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 arrowhead is drawn according to the `\figdrawarrowhead` macro settings.

The macro `\figdrawarrowcircP` 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`.

A free form arrow can be drawn with the help of the macro

$$\text{\textbackslash figdrawarrowBezier [Pt1,Pt2,Pt3,Pt4]}$$

It draws the arrow that consists of the cubic Bézier curve defined by the four control points `Pt1`, `Pt2`, `Pt3` and `Pt4`, and the arrowhead at point `Pt4` (see the section **Curve** for more information on Bézier curves).

The figure 44 shows how to use these macros. The text of the program that produces it is given below it.

Double arrow

We finally mention the specific macro

$$\text{\textbackslash figdrawdim Pt1, Pt2 (Distance)}$$

which draws a double arrow at the distance `Distance` from the segment [`Pt1`, `Pt2`]. This macro is intended to show the dimension between the points `Pt1` and `Pt2`. If we consider that `Pt1` represents the west position and `Pt2` the east position, this sets an orientation for the segment. In these conditions, the double arrow is drawn at the south of the segment if `Distance < 0`, and drawn at the north otherwise. To highlight the distance between the two points, the endpoints of the arrows are connected to `Pt1` and `Pt2` with a dashed line. The appearance of the two arrow-heads can be modified with the help of attributes set by the macro `\figset arrowhead(...)`. The distance between the points `Pt1` and `Pt2` can be printed by the companion macro `\figwritedim` with the same arguments. This writing macro has an additionnal text argument and is described at the end of the section **Writing text on the figure**.

The figure 45 shows how to use these macros. The text of the program that produces it is given below it.
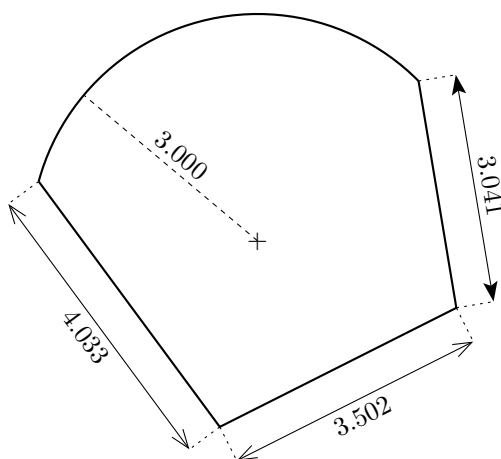

Figure 45

49

```
% 1.  Definition of characteristic points
\figinit{cm}
\figpt 0:(0,0)
\figpt 1:(-0.5, -2.46)
\figpt 2:(-2.8952, 0.786)
\figptrot 3:=2/0,-120/
\figpttraC 4:= 3 /0.5,-3/
% 2.  Creation of the graphical file
\figdrawbegin{}
\figset (width=0.8)
% Polygonal part of the figure
\figdrawline[3,4,1,2]
% Circular arc
\figget distance=\Radius[0,2]
\figdrawarccircP 0 ; \Radius [3,2]
%
\figset (width=default)
% Double arrows to show the dimensions
\figdrawdim 2,1(-0.5)
\figdrawdim 1,4(-0.5)
\figset arrowhead(fill=yes)
\figdrawdim 3,4(0.5)
% Dashed line to show the radius of the arc
\figset (dash=4)
\figptcirc 5 ::  0 ; \Radius (140)
\figdrawline[0,5]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 45}{%
\figset write (rounding=3)
% Dimensions of the edges of the polygon
\figwritedim 2,1:(-0.5)
\figwritedim 1,4:(-0.5)
\figwritedim 3,4:(0.5)
% Center of the arc and its radius
\figwritec[0]{+}
\figwritedim 5,0:\figround{\Radius}(0)
}
\centerline{\box\figBoxA}
```

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:

```
\figdrawaxes Origin(X1,X2, Y1,Y2)
\figdrawaxes 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 unit: 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 `\figdrawaxes Origin(0,L, 0,L)` and is intended to draw equal axes in a straightforward way, with two orthogonal arrows starting from the point `Origin`.

Remark: The graphical attributes of the arrows can be modified by the macro `\figset` 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 `\figdrawaxes` 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 `\figdrawaxes`, 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 A_1, A_2, A_3, A_4. It is made of three Bézier arcs, each of them being a parametric curve with the parameter value ranging in $[0, 1]$. The tangent at A_1 has the direction of $\overrightarrow{A_1 A_2}$; the tangent at A_4 has the direction of $\overrightarrow{A_3 A_4}$.

Let us notice that some points are computed during step 2 and are still needed during step 3. If a file name was given as argument to `\figdrawbegin`, it would be necessary to enforce the update mode just before by saying `\figset(update=yes)`.

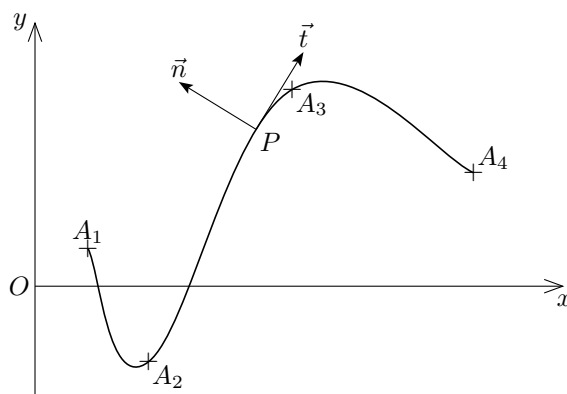The text of the program that produces this figure is given below it.

Figure 46 Frenet basis at point P

51

Corresponding program :

```
% 1.  Definition of characteristic points
\figinit{cm}
\def\ORIG{0} \figpt \ORIG:$O$(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 graphical file
\figdrawbegin{}
% Draw the axes and the curve that interpolates the data points
 \figdrawaxes \ORIG(\Xone,\Xtwo, \Yone,\Ytwo)
 \figset (width=0.6)\figdrawcurve [1,1,2,3,4,4]\figset (width=default)
% 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
 \figset arrowhead(length=0.2,fillmode=yes)
 \def\LAMBDA{1.2}
 \figpttra 26:$\vec{t}$=25/\LAMBDA,21/\figdrawarrow [25,26]
 \figvectNV 22[21]\figpttra 27:$\vec{n}$=25/\LAMBDA,22/\figdrawarrow [25,27]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{{\bf Figure 46}\\Frenet basis at point \PointName}{
% Write the name of the points
\figset write(mark=$+$)\figwriten 1:(2pt)\figwritese 2,3:(2pt)\figwritene 4:(2pt)
\figset write(mark=)\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\figBoxA}
```

8. Triangle related macros

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

The macro, whose prototype is

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

builds the altitude drawn from `Pt1` in the triangle (`Pt1`, `Pt2`, `Pt3`). `Dim` is the dimension (in user unit) 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 so-called "base line" is drawn to support the square. The attributes of this line are controlled by the settings described in the **Setting graphical attributes** section: the line width, the line style and its color can be modified with the macro `\figset` with `altitude` as keyword and respectively `blwidth`, `bldash` and `blcolor` as attributes. In the same way, the line settings of the square can be modified by the `sqwidth`, `sqdash` and `sqcolor` attributes. These attributes are all DDV attributes and thus should be used inside a `\figdrawbegin`–`\figdrawend` group to be effective.

On Figure 47, we can see the two altitudes drawn from P_2 and P_3, with respective end points H_2 and H_3. If the base line style had not been changed, the default line style would have been used and thus a solid line would have been drawn between P_3 and H_2.

The macro, whose prototype is

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

draws the so-called exterior normal, i.e. the vector of length `Length` which is normal to the oriented segment $[\texttt{Pt1},\texttt{Pt2}] = [P_1, P_2]$. 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 $\{(P_1; 1 - \lambda), (P_2; \lambda)\}$ where λ is a real number whose value is given by `Lambda`. Thus, the values 0, 1 and 0.5 correspond respectively to P_1, P_2 and the midpoint of $[P_1, P_2]$. 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 arrowhead.

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

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

computes the end point `NewPt` of the "exterior normal" to the oriented segment $[\texttt{Pt1}, \texttt{Pt2}]$. The last four arguments have the same meaning as those of the macro `\figdrawnormal` described just above. This `NewPt` can then be used as an attach point of a text to be written on the figure. On Figure 47, we can see the normal vector $\vec{n}_2$ to the segment $[P_1, P_3]$.
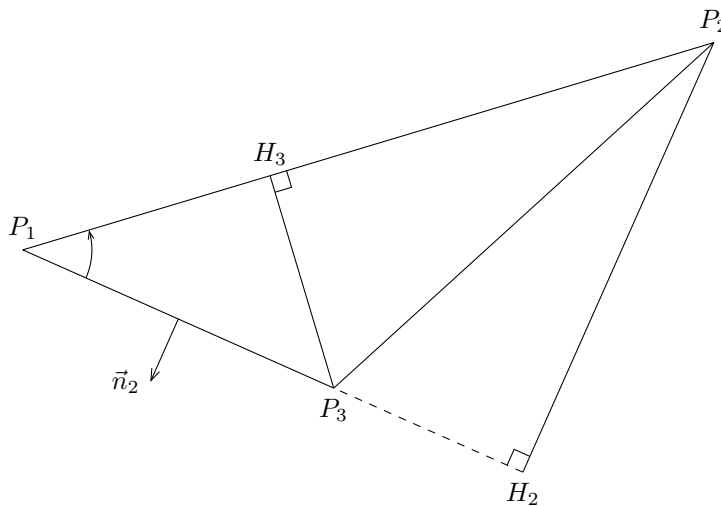

Figure 47

The text of the program that produces the figure 47 is:

```
% 1.  Definition of characteristic points
\figinit{1.3pt}
\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 graphical file
\figdrawbegin{}
\figset(join=round)\figdrawline[1,2,3,1]\figset(join=default)
\figdrawaltitude 5[3,1,2]
\figset altitude(bldash=8)\figdrawaltitude 5[2,3,1]
\figset arrowhead(ratio=0.2)\figdrawnormal 20,0.5 [1,3]
\figdrawarrowcircP 1;20 [3,2]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 47}{
\figset write(ptname=$P_{#1}$)
\figwriten 1,2,12:(4pt)
\figwrites 3,13:(4pt)
\figptendnormal 4 ::  20,0.5 [1,3]\figwritew 4:$\vec{n}_2$(4pt)
}
\centerline{\box\figBoxA}
```

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, also called isobarycenter,
- the intersection of the interior angle bisectors, which is the center of the inscribed circle or incenter,
- the intersection of one interior and two exterior angle bisectors, which is the center of one 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 straigthforward since it is the barycenter of (P_1, P_2, P_3) bearing each the coefficient 1/3. The incenter and excenters 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]      (excenter opposite to Pt1)
\figptincenter NewPt :Text[Pt1,Pt2,Pt3]
\figptcircumcenter NewPt :Text[Pt1,Pt2,Pt3]
\figptorthocenter NewPt :Text[Pt1,Pt2,Pt3]
```

The figure 48 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 graphical file
\figdrawbegin{}
\figdrawline[1,2,3,1]\figdrawline[10,11]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{{\bf Figure 47.}\ Euler's line in a triangle}{
\figwritenw 1:(2)\figwritew 2:(2)\figwritee 3:(2)
\figset write(mark=$\times$)
\figwritee 5,8:(4)\figwritew 6,7:(4)
}
\centerline{\box\figBoxA}
```
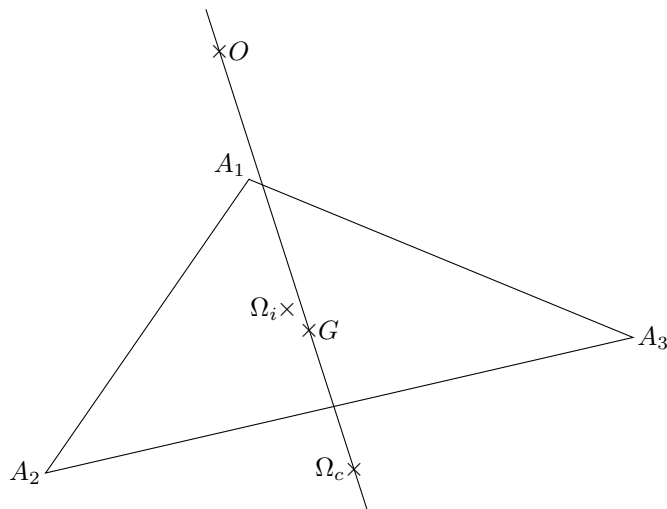
Figure 48. Euler's line in a triangle

9. Grid

The following macro comes from the approximation theory. Its prototype is

$$\figdrawtrimesh\ Type\ [Pt1,Pt2,Pt3]$$

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 internal mesh are controlled by the settings described in the **Setting graphical attributes** section: the line width, the line style and its color can be modified with the macro `\figset` with `trimesh` as keyword and respectively `width`, `dash` and `color` as attributes. Figure 49 shows some examples.
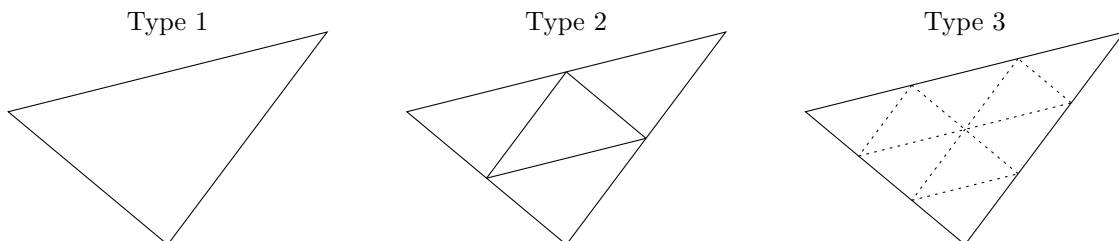
Figure 49

55

The text of the program that produces the figure 49 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 graphical file
\figdrawbegin{}
\figdrawtrimesh 1 [1,2,3]
\figptstra 1=1,2,3/1,10/\figdrawtrimesh 2 [1,2,3]
\figset trimesh(dash=5)
\figptstra 1=1,2,3/1,10/\figdrawtrimesh 3 [1,2,3]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 49}{
\figwritec[11]{Type 1}\figwritec[12]{Type 2}\figwritec[13]{Type 3}
}
\centerline{\box\figBoxA}
```

Drawing grids is also often necessary. The macro

$$\text{\figdrawmesh N1,N2 [Pt1,Pt2,Pt3,Pt4]}$$

draws a mesh of N1×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 (width, style, color) used to draw the internal mesh, are controlled by the settings described in the **Setting graphical attributes** section.

Moreover, with the help of the macro

$$\text{\figset mesh(diag=}Index\text{)}$$

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:

$$\text{\figset mesh(diag=default)}$$

Notice that the orientation of the diagonal depends on the order of the four given points. Figure 50 shows an example where the four points are the vertices of a rectangle and of a general quadrangle.
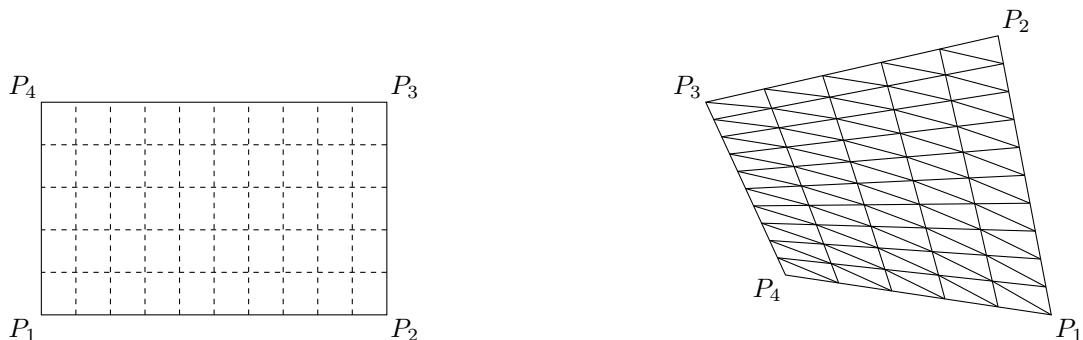

Figure 50

56

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 graphical file
\figdrawbegin{}
\figset mesh(dash=4)\figdrawmesh 10,5 [1,2,3,4]
\figset mesh(dash=default, diag=1)\figdrawmesh 10,5 [11,12,13,14]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 50}{
\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\figBoxA}
```

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, centered at the node. To create such a flow chart, one has to do the following actions:

1. Define points that are the locations of the nodes (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`, etc).

2. Create the graphical file containing the frames and the connections using the macros `\figdrawfcconnect` and `\figdrawfcnode`, *in this order*. Their prototypes are:

   `\figdrawfcconnect[Pt1,Pt2,...  ,PtN]`
   `\figdrawfcnode[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 `\figset` with the keyword made equal to `flowchart`. Moreover, the keywords `general`, `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 `\figdrawfcnode` 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 `\figdrawfcnode` 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 `\figdrawfcconnect` 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

```
\figset flowchart(line = type)
```
where *type* can take the value `polygon` or `curve`. The default value is `polygon`. When the line is polygonal, the corners can be rounded: they are replaced with circular arcs whose radius is chosen by the user by saying
```
\figset flowchart(radius = value)
```
where *value* is a dimension to be given in user unit. The default value is 0.0 . A 0 value indicates no rounding.

An arrowhead is drawn along the path, oriented from the node lying at `Pt1` towards the node lying at `PtN`. The attributes of the arrowhead are specified by the macro `\figset` with `arrowhead` as keyword (see section **Arrow**). Moreover, the position of the arrowhead along the path can be modified by saying
```
\figset 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, and makes the arrowhead to be drawn half way along the path. To be more precise, the position of the arrowhead is defined with respect to its start point or its end point. This reference point is chosen by saying
```
\figset flowchart(arrowrefpt = switch)
```
where *switch* can take the value `start` or `end`. The default value is `start`.

Remarks:
- If no arrowheads are wanted, the best is to set the arrowhead length to 0 by saying: `\figset arrowhead(length=0)` . An alternative solution is to use directly `\figdrawline` or `\figdrawcurve` instead of `\figdrawfcconnect`.
- The `arrowposition` attribute is especially useful when the line path is polygonal because an arrowhead 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 arrowhead position along the path.

Once the connections are settled, the frames can be drawn using the macro `\figdrawfcnode`. Several shapes are available. This attribute can be specified by saying
```
\figset flowchart(shape = value)
```
where *value* can be one out of `circle`, `ellipse`, `lozenge` or `rectangle`. The default value is `rectangle`. The dimensions of the frame are determinated by the text associated with the node. If the text argument of `\figdrawfcnode` 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
```
\figset flowchart(xpadding = value1, ypadding = value2)
```
where *value1* and *value2* are positive or negative dimensions to be given in user unit. One can also simply say
```
\figset flowchart(padding = value)
```
which gives the same padding *value* horizontally and vertically. This last key should logically be used when the shape is a circle (otherwise, the maximum of the two values is used). The default values are 0.0 and 0.0 .

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 unit by saying
```
\figset flowchart(thickness = value)
```
The default value is `0.0` .

Remarks:
- The color, the line width or style of a frame border can be modified by the general attributes. The color of the thickness is governed by the `thickcolor` attribute by saying
```
\figset flowchart(thickcolor = New Color) .
```
  The color of the background of the frame is the `bgcolor` color attribute, which can be set by saying
```
\figset flowchart(bgcolor = New Color) .
```
- If no border is wanted, one has to make it invisible (white) by saying `\figset(color=1)` for example before calling `\figdrawfcnode`, and reset the general color to its previous value afterwards if needed.
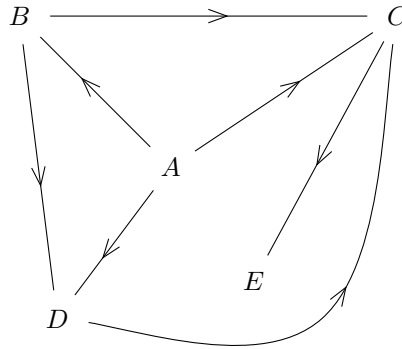
Figure 51

On figure 51, 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}
% Nodes 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 graphical file
\figdrawbegin{}
% Draw the connections between nodes
\figdrawfcconnect[1,2]\figdrawfcconnect[1,3]\figdrawfcconnect[1,4]
\figdrawfcconnect[2,3]\figdrawfcconnect[2,4]
\figdrawfcconnect[3,5]
\figset flowchart(line=curve)\figdrawfcconnect[4,0,3]
% Draw the frames at each node, without border
\figset (color=1)\figset flowchart(shape=ellipse)
\figdrawfcnode[1,2,3,4,5]{}
\figdrawend
%
% 3.  Writing text on the figure
\figvisu{\figBoxA}{{\bf Figure 51}}{
\figwritec[1,2,3,4,5]{}
}
\centerline{\box\figBoxA}
```

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{3.4cm}
\figpt 10:\centertext{4.1cm}{\bf\FigforTeX\macro package\break
  instructions diagram}(0,0)
\figpttra 20:\lefttext{\FrameWidth}{\tt\char`\\input \figforTeXfile\break
  \char`\\figsetdefault...}=10/3,2/
```

```
\figpttra 30:\lefttext{\FrameWidth}{\tt\char`\\figinit$\{$...$\}$\break
  \char`\\figpt...\break
    ...}=20/2.2,2/
\figpttra 40:\lefttext{\FrameWidth}{\tt\char`\\figdrawbegin$\{$...$\}$\break
  ...\break
  \char`\\figdrawend}=30/2.5,2/
\figpttra 50:\centertext{3.2cm}{Another\break
  graphical 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 graphical file
\figdrawbegin{}
% Draw the connections between nodes
\figdrawfcconnect[20,30]\figdrawfcconnect[30,40]
\figset flowchart(arrowpos=0.4)\figdrawfcconnect[40,50]\figreset{flowchart}
\figdrawfcconnect[50,60]\figdrawfcconnect[60,70]\figdrawfcconnect[70,80]
\figset arrowhead(fillmode = yes)\figset flowchart(radius=0.5)
\figpttra 41:=40/3,1/\figdrawfcconnect[50,51,41,40]
\figset flowchart(radius=1)
\figpttra 31:=30/5,1/\figdrawfcconnect[80,81,31,30]
% Draw the frames at each node
\figdrawfcnode[20,30,40,60]{}
\figset flowchart(bgcolor=0.9)
\figdrawfcnode[70]{}
\figset flowchart(shape=lozenge,xpadding=-0.6)
\figdrawfcnode[50]{}
\figset flowchart(xpadding=default)
\figdrawfcnode[80]{}
\figset flowchart(shape=ellipse, thickness=0.1, thickcolor=0.4)
\figset flowchart(bgcolor=default)\figdrawfcnode[10]{}
\figdrawend
%
% 3.  Writing text on the figure
\def\Xoffset{7}\def\CommentWidth{4cm}
\figvisu{\figBoxA}{{\bf Figure 51}}{
\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
  graphical 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\figBoxA}
```

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 in the `fig4tex.tex` file and thus can be used in any other context.
- 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 rigth in order to make the legend be centered under the diagram. The point #30 is translated to create the point #0, which is then "written" by a call to `\figwritep` (and thus taken into account to center the figure), but since by default the point marker is unspecified, the point is invisible (as wanted here).

11. Signs or symbols

In the section **Writing text on the figure**, we learn how to write TEX material over the figure using the location of attach points defined for this purpose. We also learn that those points can be made visible by writing a so-called *point marker* (section **Text argument**). In practice, this often reduces to write some particular characters (e.g. +, * or .) or special symbols (e.g. ● or ○) available through TEX's own capabilities. But since these characters depend on fonts, we cannot easily change their attributes, in particular their dimension.

Dealing with these signs graphically allow more flexibility. For this purpose, the following set of macros have been designed:

```
\figdrawsign[Pt1,Pt2,...  ,PtN]
\figdrawsignC(X1 Y1, X2 Y2,..., XN YN)
\figdrawsignF{FileName}
```

Their behavior is analogous to the `\figdrawline*` set of macros, but instead of drawing lines, they draw a sign (or symbol) at each point given by their number or their coordinates, explicitly provided (`\figdrawsignC`) or read into a file (`\figdrawsignF`). The syntax is the same as for the `\figdrawline*` macros ; it is explained in section **Basic drawing macros** where these macros are introduced.

Several predefined symbols are available, the default being a circle. This choice can be changed by saying `\figset sign(shape = ` *name*`)`. The following figure shows the different kinds of symbols available along with their names:

| circle | diamond | square | star | triangle | nabla | Fill mode on (default) |

Fill mode off

Figure 52

The shapes are designed so that they are inscribed in the circle of the same characteristic dimension. This parameter can be changed by saying `\figset sign(dim = ` *value*`)`, where *value* is a dimension to be given in user unit. Here, the characteristic dimension is set to 3mm to show the shapes clearly.

The attributes of the line used to draw the sign are controlled by the settings described in the **Setting graphical attributes** section: the line width, the line style, the line join and its color can be modified with the macro `\figset` with `sign` as keyword and respectively `width`, `dash`, `join` and `color` as attributes. The color is a DDV attribute, which means that the default color is dynamic, i.e. it follows the current general color, as in this example where the blue color is set as a general setting ; it can be "locked" to a specific value by saying `\figset sign(color = ` *value*`)`.

At last, by saying `\figset sign(fillmode = yes)`, the interior of the sign is painted with the chosen color ; this is the default, as shown on the upper line on figure **52**, and in this case, the other line attributes have no effect. Chosing `fillmode = no` draws only the contour of the sign and in this case, the other line attributes are relevant.

61

The text of the program that produces the figure **52** is the following:

```
% 1.  Definition of characteristic points
\figinit{cm}
\figpt 1:(1,0)
\figptstraC 2=1,2,3,4,5/2,0/        % upper line: points  1 to  6
\figptstraC 11=1,2,3,4,5,6/0,-1.5/ % lower line: points 11 to 16
% 2.  Creation of the graphical file
\figdrawbegin{}
\figset (color=\Bluergb)
\figset sign(dim=0.3)
\figdrawsign[1] % Default shape
\figset sign(shape=diamond) \figdrawsign[2]
\figset sign(shape=square)  \figdrawsign[3]
\figset sign(shape=star)    \figdrawsign[4]
\figset sign(shape=triangle)\figdrawsign[5]
\figset sign(shape=nabla)   \figdrawsign[6]
%
\figset sign(fill=no)
\figset sign(shape=circle)  \figdrawsign[11]
\figset sign(shape=diamond) \figdrawsign[12]
\figset sign(shape=square)  \figdrawsign[13]
\figset sign(shape=star)    \figdrawsign[14]
\figset sign(shape=triangle)\figdrawsign[15]
\figset sign(shape=nabla)   \figdrawsign[16]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{{\bf Figure 52}}{%
\def\Dist{0.8}
\figwritebs 1:{\tt circle}(\Dist)
\figwritebs 2:{\tt diamond}(\Dist)
\figwritebs 3:{\tt square}(\Dist)
\figwritebs 4:{\tt star}(\Dist)
\figwritebs 5:{\tt triangle}(\Dist)
\figwritebs 6:{\tt nabla}(\Dist)
\figwritee 6:{Fill mode on (default)}(1cm)
\figwritee 16:Fill mode off(1cm)
}
\centerline{\box\figBoxA}
```

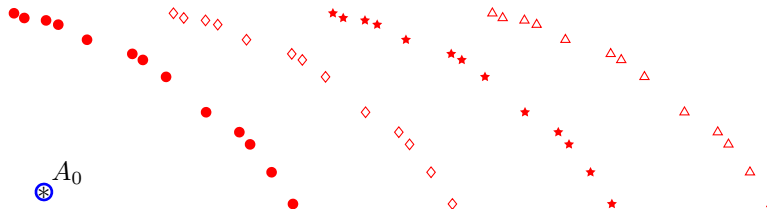On the following figure, we show a typical use of this feature:

Figure 53

The data points are first drawn with circles. For the show, they are then drawn with diamonds, stars and triangles after having been translated 3 times by the same amount towards the right. The signs are drawn with their default dimension. By looking at the program that produces this figure (see below), we

can notice that the general color setting (blue) is only used at the end (when the sign color is set back to its default value) since the specific setting (red) takes priority. The lower left point A_0 is apart from the other points and is first drawn as a "big" thick blue circle using the graphical tool \figdrawsign, and a second time as a TEX asterisk using \figwritene, after this character has been made the point marker. As expected, the two symbols show the same geometric location.

The text of the program that produces the figure **53** is the following:

```
% 1.  Definition of characteristic points
\figinit{2pt}
\figpt 0:(20,45) % Lower left point
\figpt 1:(42.9815, 66.6578)
\figpt 2:(50.5119, 60.0045)
\figpt 3:(66.852, 42.7482)
\figpt 4:(16.3086, 77.7367)
\figpt 5:(62.8328, 48.7145)
\figpt 6:(28.1114, 73.6432)
\figpt 7:(58.7876, 53.9602)
\figpt 8:(22.7012, 76.4839)
\figpt 9:(38.6159, 69.8438)
\figpt 10:(36.628, 70.9847)
\figpt 11:(56.7705, 56.2551)
\figpt 12:(14.369, 78.623)
\figpt 13:(20.4024, 77.2818)
% 2.  Creation of the graphical file
\figdrawbegin{}
\figset (color=\Bluergb)
\figset sign(color=\Redrgb)
\figdrawsign[1,2,3,4,5,6,7,8,9,10,11,12,13]
%
\figvectC 100(30,0) % Translation vector
\figptstra 1=1,2,3,4,5,6,7,8,9,10,11,12,13/1,100/
\figset sign(shape=diamond, fillmode=no)
\figdrawsign[1,2,3,4,5,6,7,8,9,10,11,12,13]
%
\figptstra 1=1,2,3,4,5,6,7,8,9,10,11,12,13/1,100/
\figset sign(shape=star, fillmode=default)
\figdrawsign[1,2,3,4,5,6,7,8,9,10,11,12,13]
%
\figptstra 1=1,2,3,4,5,6,7,8,9,10,11,12,13/1,100/
\figset sign(shape=triangle, fillmode=no)
\figdrawsign[1,2,3,4,5,6,7,8,9,10,11,12,13]
%
\figset sign(dim=1.5, width=1, color=default, shape=default)
\figdrawsign[0]
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 53}{%
\figset write(mark=$\figAst$)
\figwritene 0:(2)
}
\centerline{\box\figBoxA}
```

§10. Helpers

This section gathers some tools designed to help the user during the design phase of a figure.

`\figshowpts`

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 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). To avoid this, simply call this macro several times with shorter intervals. 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 (see section **A first look**).

`\figshowsettings`

We have seen that the user can modify some parameters such as those concerning the arrowhead, 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. An output example is shown in **3D examples**.

`\figscan`

We also recall that one can easily insert an existing graphical file with the macro `\figinsertE`. 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 **Text annotation** for more information.

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, \figptsinter-linell, \figdrawarcell, \figdrawdim, \figdrawnormal, \figdrawfcconnect, \figdrawfcnode.
  They exist because either the list of arguments is specific to the 2D (which is the case for example for \figptell), or they are not relevant in a 3D context. 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 graphical 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 projection 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 54):

- 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 λ usually chosen between 0 and 1, which is the depth reduction coefficient of the perspective.

The angle ψ can be changed by saying \figset projection(psi=*value*). The default value is 40 degrees. Values of ψ 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 on the figure. However, we may find that the objects seem too elongated. This is why the default value is $\lambda = 0.5$, which corresponds to the so-called *cabinet* projection and leads to a more realistic result. The coefficient λ can be changed by saying \figset projection(lambda=*value*).
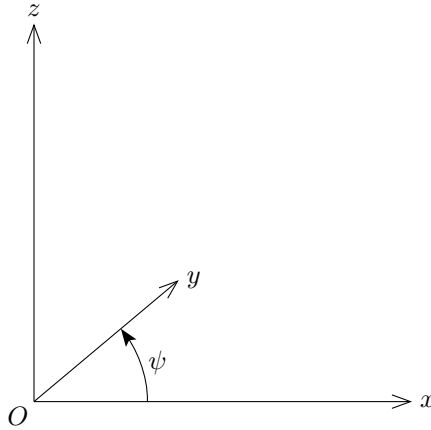
Figure 54. Definition of the cavalier projection

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 of the projection are used.

```
% 1.  Definition of characteristic points
\figinit{5cm, 3D}
\figpt 0:$O$(0,0)
\figptsaxes 1:0(1)
\def\Rpsi{0.3}\figptcirc 4:$\psi$:0,1,3;\Rpsi(20)
% 2.  Creation of the graphical file
\figdrawbegin{}
\figdrawaxes 0(1)
\figset arrowhead(length=0.05,fillmode=yes)
\figdrawarrowcirc 0,1,3;\Rpsi(0,40)
\figdrawend
% 3.  Writing text on the figure
\def\dist{3pt}
\figvisu{\figBoxA}{{\bf Figure 54.}\ \ Definition of the cavalier projection}{
\figwritesw 0:(\dist)\figwritee 1,2,4:(\dist)\figwriten 3:(\dist)}
\centerline{\box\figBoxA}
```

The **realistic projection** is defined by a target point T and an observation point E, also called eye point (see figure 55). 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 projection by saying \figset projection(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 projection(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 $\psi = (\vec{x}, \overrightarrow{TH})$ and the latitude $\theta = (\overrightarrow{TH}, \overrightarrow{TE})$, angles to be given in degrees, that can be set by saying \figset projection(psi=*value1*, theta=*value2*) . The default values are $\psi = 40°$ and $\theta = 25°$.
  H is the orthogonal projection of E onto the plane (T, x, y).

The text of the program that produces the figure 55 is given below it.
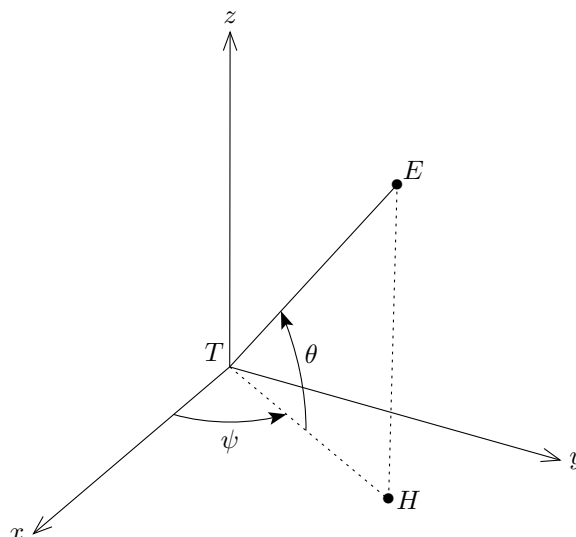
66

Figure 55. 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 graphical file
\figset projection(psi=30,theta=30)
\figdrawbegin{}
\figdrawaxes 0(1)
\figdrawline[0,4]
\figset arrowhead(length=0.05,fillmode=yes)
\figdrawarrowcircP 0;\Rpsi[1,5,2]\figdrawarrowcircP 0;\Rtheta[5,4,3]
\figset(dash=5)\figdrawline[0,5,4]
\figdrawend
% 3.  Writing text on the figure
\def\dist{3pt}
\figvisu{\figBoxA}{{\bf Figure 55.}\ \ Definition of the perspective projection}{
\figwritenw 0:(\dist)\figwritew 1:(\dist)\figwritee 2:(\dist)\figwriten 3:(\dist)
\figwrites 6:(\dist)\figwritene 7:(\dist)
\figset write(mark=$\bullet$)\figwritene 4:(\dist)\figwritee 5:(\dist)}
\centerline{\box\figBoxA}
```

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 ψ and θ (see figure 55), which are set with the macro \figset projection().

67

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:

$$\text{\texttt{\textbackslash figinit} \{\texttt{ScaleFactorUnit}, \mathit{Dim/Proj}\}}$$

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:

$$\texttt{\textbackslash figvectC NewVect (X,Y,Z)}$$

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

The macro

$$\texttt{\textbackslash figvectN NewVect [Pt1,Pt2,Pt3]}$$

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 $\vec{V} = \dfrac{\overrightarrow{P_1 P_2}}{||\overrightarrow{P_1 P_2}||} \times \dfrac{\overrightarrow{P_1 P_3}}{||\overrightarrow{P_1 P_3}||}$. Thus, $(\overrightarrow{P_1 P_2}, \overrightarrow{P_1 P_3}, \vec{V})$ is a positively oriented basis.

The macro

$$\texttt{\textbackslash figvectNV NewVect [Vector1, Vector2]}$$

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 $\vec{V} = \dfrac{\vec{V_1}}{||\vec{V_1}||} \times \dfrac{\vec{V_2}}{||\vec{V_2}||}$. Thus, $(\vec{V_1}, \vec{V_2}, \vec{V})$ is a positively oriented basis.

Basic macros

The macro

$$\texttt{\textbackslash figpt NewPt :Text(X,Y,Z)}$$

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

$$\texttt{\textbackslash figget angle OP value = \textbackslash Result[Center,Pt1,Pt2,Pt3]}$$

computes the value (in degrees) of the oriented angle $(\overrightarrow{CP_1}, \overrightarrow{CP_2})$ and stores it in the macro `\Result` 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 $(C, P1)$. Provided this is verified,

P_3 may be equal to P_2. If "OP value" is present, then the angle is modified by the specified arithmetical operation (see the **Symbolic constants** subsection).

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 $\overrightarrow{V}$ 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 $(C, \overrightarrow{V})$. The sense of rotation is such that $(\overrightarrow{CP}, \overrightarrow{CP'}, \overrightarrow{V})$ 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 $\overrightarrow{IP'} = A \; \overrightarrow{IP}$, where A is the matrix whose coefficients are `Aij`, $1 \leq \mathtt{i} \leq 3$, $1 \leq \mathtt{j} \leq 3$ 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 $\overrightarrow{CP_1} \times \overrightarrow{CP_2}$, 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
$$\verb|\figptsintercirc NewPt1 [Center1,Radius1 ; Center2,Radius2 ; PlanePt]|$$
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
$$\verb|\figptsaxes NewPt1 : Origin(X1,X2, Y1,Y2, Z1,Z2)|$$
computes the end points of the axes corresponding to the arrows drawn by the calling statement:
$$\verb|\figdrawaxes Origin(X1,X2, Y1,Y2, Z1,Z2)|.$$
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

The "coordinate" version of `\figdrawline` and `\figdrawsign` have their 3D equivalent, whose prototype are the following:
$$\verb|\figdrawlineC(X1 Y1 Z1, X2 Y2 Z2,..., XN YN ZN)|$$
$$\verb|\figdrawsignC(X1 Y1 Z1, X2 Y2 Z2,..., XN YN ZN)|$$
which corresponds to the 2D one with the third coordinate of each point added within each group of data. The same applies for the "file" version of these macros (`\figdrawlineF` and `\figdrawsignF`), which expect every group of data read in the file, made of three numbers.

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:
$$\verb|\figdrawcirc Center,Pt1,Pt2 (Radius)|$$
which draws the entire circle,
$$\verb|\figdrawarccirc Center,Pt1,Pt2 ; Radius (Ang1,Ang2)|$$
which draws the circular arc limited by the angles `Ang1` and `Ang2` given in degrees and measured counter-clockwise around the vector $\overrightarrow{CP_1} \times \overrightarrow{CP_2}$, starting from the half-line (C, P_1),
$$\verb|\figdrawarrowcirc Center,Pt1,Pt2 ; Radius (Ang1,Ang2)|$$
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 arrowhead is drawn according to the `\figdrawarrowhead` macro settings.

The last two macros have a so-called "point" version. Let P_3 stand for `Pt3` and $\overrightarrow{N} = \overrightarrow{CP_1} \times \overrightarrow{CP_3}$. 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 $(C, P1)$.

These macros are:
$$\verb|\figdrawarccircP Center ; Radius [Pt1,Pt2,Pt3]|$$
which draws the circular arc of radius `Radius`, from P_1 towards P_2 turning counterclockwise around the axis $(C, \overrightarrow{N})$,
$$\verb|\figdrawarrowcircP Center ; Radius [Pt1,Pt2,Pt3]|$$

which draws the circular arrow of radius |Radius|, from P_1 towards P_2 turning around the axis $(C, \overrightarrow{N})$:
    . counterclockwise if the radius is positive,
    . clockwise if the radius is negative.

Remarks:
- An arrow (or an arrowhead) 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
$$\text{\textbackslash figdrawaxes Origin(X1,X2, Y1,Y2, Z1,Z2)}$$
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 unit: 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: \figdrawaxes Origin(L) . It is equivalent to \figdrawaxes 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, ... 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/
\figset write(ptname=$P_#1$)
```

As we will need to draw the cube several times, we define the macro `\DrawCube` to create the graphical file. The background lines are dashed; obviously, they depend on the observation direction.

```
\def\DrawCube{
% 2.  Creation of the graphical file
\figdrawbegin{}
\figdrawline[1,2,3]\figdrawline[5,6,7,8,5]
\figdrawline[1,5]\figdrawline[2,6]\figdrawline[3,7]
\figset(dash=4)\figdrawline[1,4,3]\figdrawline[4,8]
\figdrawend}
```

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. The number of the figure is available in the document with the help of the expandable macro `\figforTeXFigno` (see **Writing text on the figure**).

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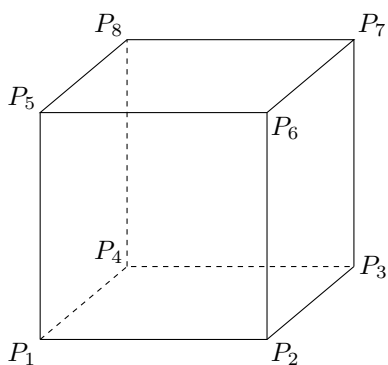

Figure 56. Default settings
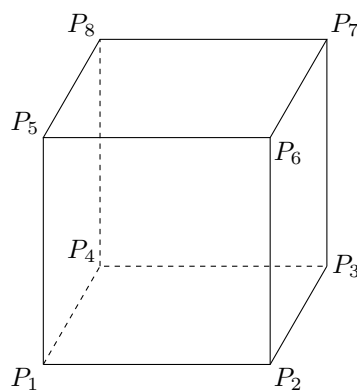
Figure 57. $\psi = 60$

Thus, the figures 56 and 57 are obtained by simply writing:

```
\DrawCube\WriteOnCube{\figBoxA}{Default settings}
\def\Valpsi{60} \figset projection(psi=\Valpsi)
\DrawCube\WriteOnCube{\figBoxB}{$\psi = \Valpsi$}
\centerline{\box\figBoxA\hfil\box\figBoxB}
```

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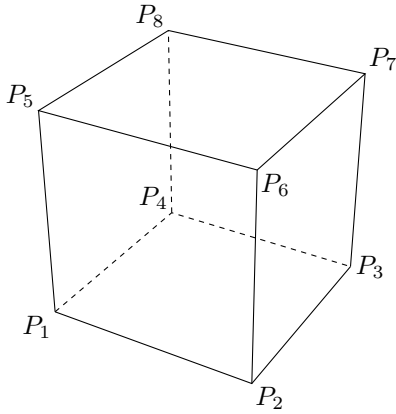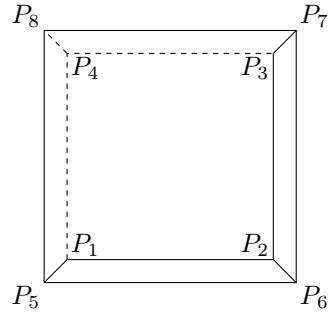

Figure 58. $(\psi, \theta) = (-60, 30)$
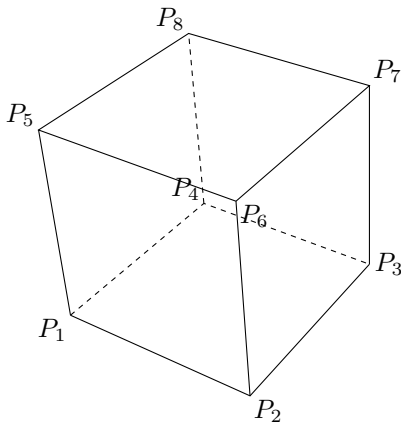

Figure 59. $(\psi, \theta) = (-90, 90)$

Figure 60. $(\psi, \theta) = (-60, 30),\ T = (1, 1, 1)$
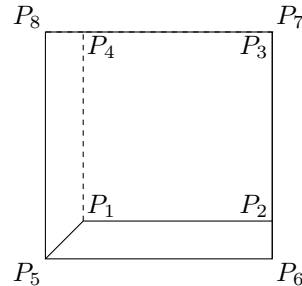
Figure 61. $(\psi, \theta) = (-90, 90),\ T = (1, 1, 1)$

We recall that the observation direction is defined by the angles (ψ, θ). 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}
...
\figset write(ptname=$P_#1$)
\def\Valpsi{-60}\def\Valtheta{30} \figset projection(psi=\Valpsi, theta=\Valtheta)
\DrawCube\WriteOnCube{\figBoxA}{$(\psi, \theta)=(\Valpsi, \Valtheta)$}
\def\Valpsi{-90}\def\Valtheta{90} \figset projection(psi=\Valpsi, theta=\Valtheta)
\DrawCube\WriteOnCubeB{\figBoxB}{$(\psi, \theta)=(\Valpsi, \Valtheta)$}
\centerline{\box\figBoxA\hfil\box\figBoxB}
```

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

```
================================================================================
 -> DDV means "with dynamic default value" (i.e.  following the context),
    PS means PostScript.
Current settings are:
 --- GENERAL ---
Scale factor and Unit = 3cm (85.35826pt)  -> \figinit{ScaleFactorUnit}
Update mode = yes -> \figset(update=yes/no) or \figsetdefault(update=yes/no)
 --- WRITING ---
Implicit point name = $P_i$  -> \figset write(ptname = {Name})
Point marker =  -> \figset write(mark = Mark)
Number of decimals to print = 2 -> \figset write(rounding = NDec/yes/no)
 --- GRAPHICAL (general) ---
Color = 0 -> \figset(color = ColorDefinition)
Filling mode = no -> \figset(fillmode = yes/no)
Line cap = butt -> \figset(cap = butt/round/square)
Line join = miter -> \figset(join = miter/round/bevel)
Line style = 4 -> \figset(dash = Index/Pattern)
Line width = 0.4 -> \figset(width = real in PS unit)
Transparency = 1 -> \figset(alpha = real in [0,1]=[transparent, opaque])
 --- GRAPHICAL (specific) ---
Altitude (all the following attributes are DDV):
 Base line color = general color -> \figset altitude(blcolor=ColorDefinition)
 Base line style = general style -> \figset altitude(bldash=Index/Pattern)
 Base line width = general width -> \figset altitude(blwidth=real in PS unit)
 Square line color = general color -> \figset altitude(sqcolor=ColorDefinition)
 Square line style = general style -> \figset altitude(sqdash=Index/Pattern)
 Square line width = general width -> \figset altitude(sqwidth=real in PS unit)
Arrowhead:
 (half-)Angle = 20 -> \figset arrowhead(angle = real in degrees)
 Filling mode = no -> \figset arrowhead(fillmode = yes/no)
 "Outside" = no -> \figset arrowhead(out = yes/no)
 Length = 0.09363  (active) -> \figset arrowhead(length = real in user unit)
 Ratio = 0.1 (not active) -> \figset arrowhead(ratio = real in [0,1])
Curve:
 Roundness = 0.2 -> \figset curve(roundness = real in [0,0.5])
Flow chart:
 Arrow position = 0.5 -> \figset flowchart(arrowposition = real in [0,1])
 Arrow reference point = start -> \figset flowchart(arrowrefpt = start/end)
 Background color = 1 -> \figset flowchart(bgcolor = ColorDefinition)
 Line type = polygon -> \figset flowchart(line = polygon/curve)
 Padding = (0.0 , 0.0 )  -> \figset flowchart(padding = real in user unit)
    (or \figset flowchart(xpadding = real, ypadding = real) )
 Radius = 0.0  -> \figset flowchart(radius = positive real in user unit)
 Shape = rectangle -> \figset flowchart(shape = circle, ellipse, lozenge or rec
tangle)
 Thickness color (DDV) = general color -> \figset flowchart(thickcolor=ColorDef
inition)
 Thickness = 0.0  -> \figset flowchart(thickness = real in user unit)
Mesh:
 Diagonal = 0 -> \figset mesh(diag = integer in {-1,0,1})
 Lines color (DDV) = general color -> \figset mesh(color = ColorDefinition)
 Lines style (DDV) = general style -> \figset mesh(dash = Index/Pattern)
 Lines width (DDV) = general width -> \figset mesh(width = real in PS unit)
```

74

```
 Sign:
  Sign color (DDV) = general color -> \figset sign(color = ColorDefinition)
  Style = 1 -> \figset sign(dash = Index/Pattern)
  Dimension = 0.0234  -> \figset sign(dimension = real in user unit)
  Filling mode = yes -> \figset sign(fillmode = yes/no)
  Join mode = miter -> \figset sign(join = miter/round/bevel)
  Shape = circle -> \figset sign(shape = circle, diamond, nabla, square, star or
  triangle)
  Lines width = 0.4 -> \figset sign(width = real in PS unit)
 Trimesh:
  Lines color (DDV) = general color -> \figset trimesh(color = ColorDefinition)
  Lines style (DDV) = general style -> \figset trimesh(dash = Index/Pattern)
  Lines width (DDV) = general width -> \figset trimesh(width = real in PS unit)
  --- 3D to 2D PROJECTION ---
 Projection :  realistic -> \figinit{ScaleFactorUnit, ProjType}
 Longitude (psi) = -90 -> \figset projection(psi = real in degrees)
 Latitude (theta)  = 90 -> \figset projection(theta = real in degrees)
 Observation distance = 4.99493  -> \figset projection(dist = real in user unit)
 Target point = CenterBoundBox -> \figset projection(targetpt = point 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 projection(targetpt=7) and then execute the same program, then we obtain the last two figures 60 and 61.

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

$$\text{\textbackslash figset projection(distance=10)}$$

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{2cm, 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 projection(psi=\Valpsi, theta=\Valtheta)
```

The left-hand side figure 62 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 graphical file
\figdrawbegin{}
\figdrawline[1,2,3,4,1]\figdrawline[11,12,13,14,11]\figdrawline[5,6]
\figdrawend
% 3a.  Writing text on the figure
\figvisu{\figBoxA}{\bf Figure 62}{}
```
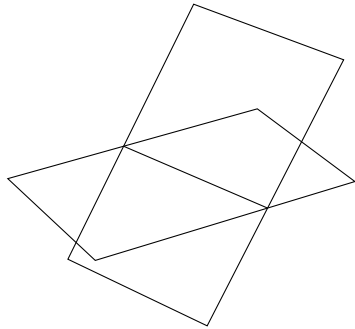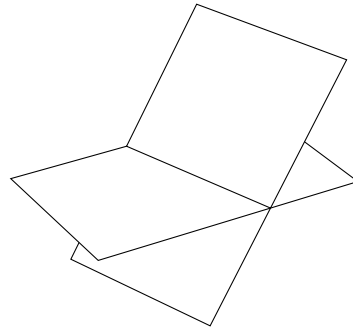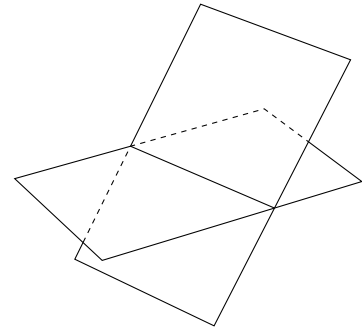
75

Figure 62   **Figure 63**   **Figure 64**

A solution to this is proposed on the figure 63 (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 θ 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 graphical file
\figdrawbegin{}
\def\white{1}\def\black{0}
\figset(fillmode=yes,color=\white)\figdrawline[11,14,6,5,11]
\figset(fillmode=no, color=\black)\figdrawline[11,14,6,5,11]
\figset(fillmode=yes,color=\white)\figdrawline[1,2,3,4,1]
\figset(fillmode=no, color=\black)\figdrawline[1,2,3,4,1]
\figset(fillmode=yes,color=\white)\figdrawline[5,6,13,12,5]
\figset(fillmode=no, color=\black)\figdrawline[5,6,13,12,5]
\figdrawend
% 3b.  Writing text on the figure
\figvisu{\figBoxB}{\bf Figure 63}{}
```

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 ψ and θ 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 graphical file
\figdrawbegin{}
\figptvisilimSL 7:[2,3 ; 6,13]
\figptvisilimSL 15:[5,11 ; 1,4]
\figdrawline[5,1,4,3,7]\figdrawline[5,12,13,14,11,15]\figdrawline[5,6]
\figset(dash=4)\figdrawline[15,5,2,7]
\figdrawend
% 3c.  Writing text on the figure
\figvisu{\figBoxC}{\bf Figure 64}{}
```

The three figures are then displayed by saying:

```
\centerline{\box\figBoxA\hfil\box\figBoxB\hfil\box\figBoxC}
```

● **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 an orthogonal 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. 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, from which is deduced the third vector $\vec{u}_2$. This also gives the sail a neutral position in the plane $(P; \vec{u}_1, \vec{u}_3)$. 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`).

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.
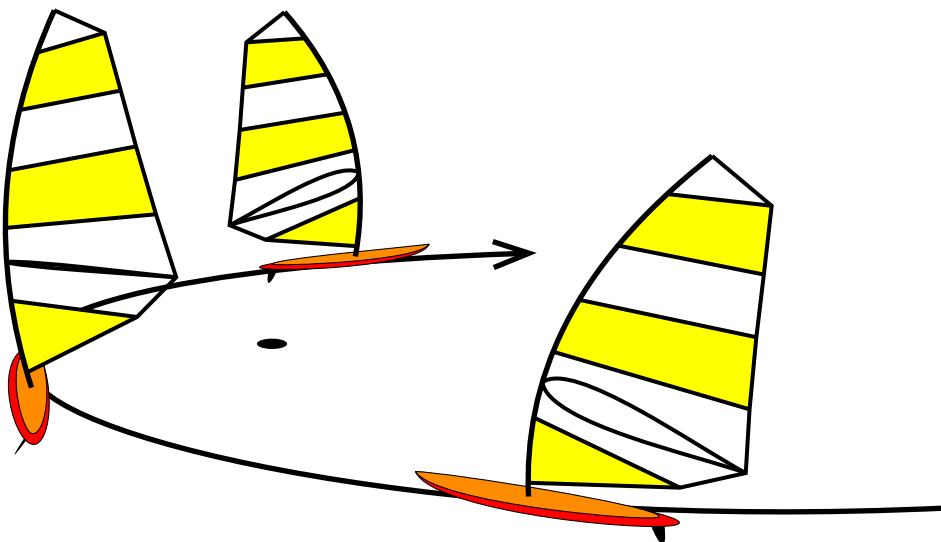

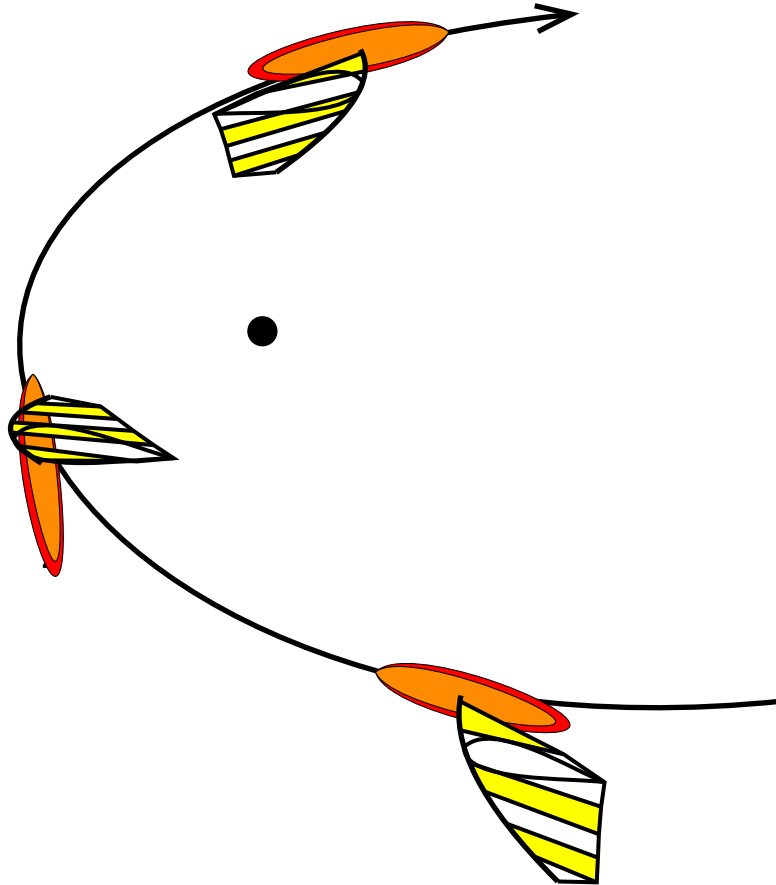**Figure 65.** $(\psi, \theta) = (-35, 20)$

77

Figure 66. $(\psi, \theta) = (-35, 90)$

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/
\figset(fillmode=yes,color=\Blackrgb)\figdrawBezier 1[20,21,22,23]
```

78

```
% Compute the symmetric part of the board and draw it
\figptssym 20=2,3/1,12/
\figset(width=default,color=\Redrgb)\figdrawBezier 2[1,2,3,4,21,20,1]
\figset(fillmode=no,color=\Blackrgb)\figdrawBezier 2[1,2,3,4,21,20,1]
\figptssym 20=5,6/1,12/
\figset(fillmode=yes,color=\DarkOrangergb)\figdrawBezier 2[1,5,6,7,21,20,1]
\figset(fillmode=no,color=\Blackrgb)\figdrawBezier 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
\figset(fillmode=yes,color=\Yellowrgb)
\figdrawline[40,31,41]\figdrawline[33,34,44,43]\figdrawline[35,36,46,45]
\figset(fillmode=no,width=1.5,color=\Blackrgb)
\figdrawline[31,41]\figdrawline[33,43]\figdrawline[34,44]
\figdrawline[35,45]\figdrawline[36,46]
\figdrawline[40,41,42,43,44,45,46,23]
% Draw the wishbone
\figdrawBezier 1[32,37,38,42]\figptssym 37=37,38/8,12/\figdrawBezier 1[32,37,38,42]
% Draw the mast
\figset(width=2)\figdrawBezier 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
\figset(width=2)\figset arrowhead(length=0.6)\figdrawarrowBezier [15,16,17,18]
\figpttraC 1:=19/1,0,0/\figpttraC 2:=19/0,1,0/
\figset(fillmode=yes,color=\Blackrgb)\figdrawcirc 19,1,2(0.2)}
%--------------
\def\newfig{% Creates a new graphical file and a new figure
% 2.  Creation of the graphical file
\figdrawbegin{}
\trajectory\jibe
\figdrawend
% 3.  Writing text on the figure
\figvisu{\figBoxA}{{\bf Figure \figforTeXFigno.}\\$(\psi, \theta)=(\Valpsi, \Valtheta)$}{}
\vskip0.8cm
\centerline{\box\figBoxA}}
%--------------
% 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 projection(targetpt=19, distance=20)
\def\Valpsi{-35}\def\Valtheta{20}\figset projection(psi=\Valpsi, theta=\Valtheta)
\newfig
% Second figure
\def\Valtheta{90}\figset projection(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 shows 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]
```

§12. Using this macro package

This macro package is ready to be used with plain TeX and LaTeX under any Unix system (including Linux and Mac OS X) and Windows.

Remark (probably no longer relevant): 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.

Prerequisite knowledge about TeX

In order to fully benefit from the macros described in this documentation, it may be useful to know of two fundamental TeX features, namely box and macro definition.

A box is a basic object in TeX; everything is box in TeX, from the character to the page. Thus, Fig4TeX is designed so that the useful result (produced by `\figvisu`) is a box. To be very short, a box can be referred to by a name, e.g. `\myBox`, and the contents of the box are displayed by saying `\box\myBox`. Afterwards, the box is empty and can be reused and filled again with other material.

The second important feature is the so-called macro definition. The simplest syntactic form is similar to a variable assignment: `\def\Macro{`*definition*`}`, where `\Macro` is a name chosen by the user and *definition* is any text or material allowed by TeX. Any subsequent definition of `\Macro` will override this one. The macro is then expanded by just saying `\Macro`, which means that the *definition* is displayed in this place.

Arguments can also be given in a macro definition. In the calling sequence, the arguments follow the macro name and are enclosed with braces {}. In order to make usage of the macros more intuitive, the arguments can be separated by specific characters, called delimiters, which are mandatory in the calling sequence. Most of the macros of Fig4TeX are built according this schema. For example, the coordinates of a point are always given as a comma separated list of numbers surrounded by parentheses: `(x, y)` in 2D or `(x, y, z)` in 3D.

Nota: Instead of `\def`, LaTeX users may prefer to use `\newcommand` and `\renewcommand`. The `\def` construction has the same effect as `\renewcommand` in that the macro `\Macro` is overridden. It can be secure to use a group to make a definition local (to the group) and thus not affect the entire document. A group is simply delimited by a couple of braces {}.

Fig4TeX and LaTeX

To use Fig4TeX with LaTeX, there are two ways:
- write `\usepackage{fig4tex}` before `\begin{document}`,
- or write `\input fig4tex.tex` after `\begin{document}`.

Some users sometimes wonder if the LaTeX environments can be used together with Fig4TeX, especially the `figure` environment and the labelling tools.

The answer is: "Yes". Here follows a template showing a typical usage:

```
\documentclass[12pt]{article}
\usepackage{fig4tex}

\begin{document}
Some text before the figure \ref{ItsOK}.

\figinit{...}
 ...
\figvisu{\figBoxA}{}{
 ...
}
\begin{figure}[h]
 \centerline{\box\figBoxA}
\caption{The figure}
\label{ItsOK}
\end{figure}

Some text after the figure.
\end{document}
```

The command `\centerline{\box\figBoxA}` can be replaced with:

```
\begin{center}
 \begin{tabular}{c}
   \box\figBoxA
 \end{tabular}
\end{center}
```

To display to figures side by side, we can just say (see **Writing text on the figure**):

`\centerline{\box\figBoxA\hfil\box\figBoxB}` or equivalently:

```
\begin{center}
 \begin{tabular}{cc}
   \box\figBoxA & \box\figBoxB
 \end{tabular}
\end{center}
```

Compatibility with other packages

Problems are to be expected if a macro is defined in two packages with different meanings. In order to avoid such conflicts, FigTEX warns by issuing a message when such redefinitions are detected, in which case it is wise to change the conflicting macro names in either package or the other.

FigTEX proves to be compatible with most common packages. However, some packages change the behaviour of particular characters (such as colon or semicolon), by changing their category code, to be precise. This is especially the case with packages dealing with language specificities, namely French. This may cause trouble to FigTEX, but an easy workaround is to load FigTEX in last place, which leads to the following structure:

TEX	LATEX
`\input frencha4.tex`	`\documentclass[12pt]{article} % for example`
`\input fig4tex.tex % <- LAST`	`\usepackage[french]{babel}`
	`\usepackage{fig4tex} % <- LAST`
`   ...`	
`\bye`	`\begin{document}`
	`  ...`
	`\end{document}`

Bug report

The author would appreciate to receive back any information about trouble in using this macro package. Please send your message to `Yvon.Lafranche@univ-rennes1.fr`.

§13. List of user macros

An HTML version of this list is available at:
        https://perso.univ-rennes1.fr/yvon.lafranche/fig4tex/ReferenceGuide.html

```
                 ------------------------------------
                | 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{} or \figinsert{FileName} or \figinsert{FileName, ScaleFactor}
  Insertion of the file FileName in the page, scaled by ScaleFactor which by default
  equals to 1.  This macro must be used to include a graphical file created by Fig4TeX.
  If the argument is empty or blank, the internal default file name is used, i.e.  the
  included file is the last file created by the command \figdrawbegin{}.  Otherwise,
  the first argument is assumed to be the valid file name of an existing file.
  If given, the file name must not begin with a digit.
  See also \figinsertE.
\figinsertE{FileName} or \figinsertE{FileName, ScaleFactor}
  Insertion of the external file FileName in the page, scaled by ScaleFactor
  which by default equals to 1.  In PDF mode, this macro allows to include
  so-called "external" files such as JBIG2, JPEG, PDF or PNG files.
  In DVI mode, this macro allows to include PostScript files only.
\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 projection (attribute1=value1, attribute2=value2,...)
  Setting the parameters governing the 3D -> 2D projection.
  The general calling sequence is
          \figset keyword (attribute1=value1, attribute2=value2,...)
  Here, the required keyword is "projection".  See the macro \figset in the section
  "Macros for graphical generation" for the other keywords.  The attributes are
  given below, along with the possible values and their definition.
  The current values can be obtained with the help of \figshowsettings.
  - keyword = projection
   . depth (or lambda) = real in [0,1] (default = 0.5):  depth reduction coefficient
```

83

```
              (for the cavalier projection).
        .   distance = real in user unit (default = 5 x the 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 (\figdrawbegin) 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.
     - other keywords :  see "Control macros" in section "Macros for graphical generation".
\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.  Commands and Caption can be void.
  Three boxes are preallocated and are available to the user:  \figBoxA, \figBoxB
  and \figBoxC (a single one is generally sufficient for a whole document since
  it can be reused several times).
  The numbers of the current figure and of the next one are also available via
  the expandable macros \figforTeXFigno and \figforTeXnextFigno.


       --------- 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 \figset write(ptname={...})).
\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.
\figget keyword = \Result [Arg1,...ArgN]
\figget keyword OP value = \Result [Arg1,...ArgN]
  Computation of geometrical parameters specified by a keyword.
  The simplified calling sequence is
         \figget keyword = \Result [Arg1,...ArgN]
  where \Result is a macro that holds the expected result and Argi are the arguments
  needed to compute this value.  \Result is a macro whose name is chosen by the user
  and can then be used as a symbolic numerical constant.
  The computed value can be modified by an arithmetic operation by using the general
  calling sequence
         \figget keyword OP value = \Result [Arg1,...ArgN]
  where OP is one of *, / or +, and value is a numerical constant or a symbolic numerical
```

constant (i.e. stored in a macro). Thus, if e denotes the neutral element of OP, the
simplified calling sequence above is equivalent to
          \figget keyword OP e = \Result [Arg1,...ArgN]
For each keyword, the definition of the associated parameter is given below along with
the simplified calling sequence:
- keyword = angle
  Calling sequence in 2D: \figget angle = \Result [Center,Pt1,Pt2]
  Calling sequence in 3D: \figget angle = \Result [Center,Pt1,Pt2,Pt3]
  Computation of the value \Result (in degrees) of the oriented angle (CP1, CP2), where
  C=Center, P1=Pt1, P2=Pt2.
  In 3D, the plane is oriented so that the angle is measured counterclockwise around the
  vector CP1 x CP3, where P3=Pt3. Notice that C, P1, P2 and P3 must be coplanar, but P3
  must not lie on the line (C,P1).
- keyword = distance
  Calling sequence: \figget distance = \Result [Pt1,Pt2]
  Computation of the euclidian distance \Result between the points Pt1 and Pt2.
  The distance is expressed in user units.
\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|| x 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 Vector1 / ||Vector1|| x Vector2 / ||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) ---
\figptmap NewPt :Text= Pt /InvPt/A11,A12 ; A21,A22/
\figptmap 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

```
\figptinv NewPt :Text= Pt /Center, Ratio/
  Image NewPt of point Pt by the inversion of center Center and of ratio Ratio
  with a joined Text
\figptrot NewPt :Text= Pt /Center, Angle/
\figptrot 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.
\figptsym NewPt :Text= Pt /LinePt1, LinePt2/
\figptsym 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.
\figpttra NewPt :Text= Pt /Lambda, Vector/
  Image NewPt of point Pt by the translation of vector Lambda * Vector
  with a joined Text.
\figpttraC NewPt :Text= Pt /X,Y/
\figpttraC 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.
\figptorthoprojline NewPt :Text= Pt /LinePt1, LinePt2/
  Image NewPt of point Pt by the orthogonal projection onto the line (LinePt1,LinePt2),
  with a joined Text.
\figptorthoprojplane 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) ---
\figptsmap NewPt1 = Pt1, Pt2, ..., PtN /InvPt/A11,A12 ; A21,A22/
\figptsmap 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 \figptstra.
\figptshom NewPt1 = Pt1, Pt2, ..., PtN /Center, Ratio/
  Images NewPt1, NewPt1+1,..., NewPt1+(N-1) of points Pt1, Pt2, ..., PtN
  by the homothety of center Center and of ratio Ratio.
  See following information at \figptstra.
\figptsinv NewPt1 = Pt1, Pt2, ..., PtN /Center, Ratio/
  Images NewPt1, NewPt1+1,..., NewPt1+(N-1) of points Pt1, Pt2, ..., PtN
  by the inversion of center Center and of ratio Ratio.
  See following information at \figptstra.
\figptsrot NewPt1 = Pt1, Pt2, ..., PtN /Center, Angle/
\figptsrot NewPt1 = Pt1, Pt2, ..., PtN /Center, Angle, Vector/
  Images NewPt1, NewPt1+1,..., NewPt1+(N-1) of points Pt1, Pt2, ..., 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
```

```
    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=Center, P=Pti, P'=NewPti.
  See following information at \figptstra.
\figptssym NewPt1 = Pt1, Pt2, ..., PtN /LinePt1, LinePt2/
\figptssym NewPt1 = Pt1, Pt2, ..., PtN /PlanePt, NormalVector/
  Images NewPt1, NewPt1+1,..., NewPt1+(N-1) of points Pt1, Pt2, ..., 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 NewPt1 = Pt1, Pt2, ..., PtN /Lambda, Vector/
  Images NewPt1, NewPt1+1,..., NewPt1+(N-1) of points Pt1, Pt2, ..., PtN
  by the translation of vector Lambda * Vector
  - Common information to all multiple results macros -
  Text eventually previously associated with the result points is lost.
  The result points NewPti have successive numbers.
  The data points Pti can be given in any order.
  WARNING : Let Ir (resp.  Id) the set of the result points numbers (resp.  the set
  of the given points numbers).  Let J the intersection of Ir and Id.
  1) Ir = Id, or J is empty, or 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, NewPt1 belongs to J, and the given points are
      taken into account sequentially, beginning from the first element in the list.
      -> Take note that one can take advantage of this feature to create a sequence
      of points such that P_{n+1} is the image of P_{n}.  For example, given point 1,
      the statement  \figptsXXX 2=1,2/.../    will create point 2 image of point 1,
      and point 3 image of point 2.
\figptstraC NewPt1 = Pt1, Pt2, ..., PtN /X,Y/
\figptstraC NewPt1 = Pt1, Pt2, ..., PtN /X,Y,Z/
  Images NewPt1, NewPt1+1,..., NewPt1+(N-1) of points Pt1, Pt2, ..., PtN
  by the translation of vector (X,Y) in 2D, (X,Y,Z) in 3D
  See following information at \figptstra.
\figptsorthoprojline NewPt1 = Pt1, Pt2, ..., PtN /LinePt1, LinePt2/
  Images NewPt1, NewPt1+1,..., NewPt1+(N-1) of points Pt1, Pt2, ..., PtN
  by the orthogonal projection onto the line (LinePt1, LinePt2).
\figptsorthoprojplane NewPt1 = Pt1, Pt2, ..., PtN /PlanePt, NormalVector/
  Images NewPt1, NewPt1+1,..., NewPt1+(N-1) of points Pt1, Pt2, ..., 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 NewPt1, NewPt1+1 (and NewPt1+2 in 3D) of the arrows corresponding to
  the axes, drawn by the macro \figdrawaxes.  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 \figdrawaxes.
\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 abscissa in [0,1] of the
  origin of the normal with respect to the segment [Pt1, Pt2], 0 corresponding to Pt1
  and 1 to Pt2. This sets the position of the normal 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
  \figdrawcurve when it is invoqued by  \figdrawcurve [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
   \figdrawBezier \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 roundness.
\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.
```

```
                  ------------------------------------
                  | Writing macros - Non-mute macros |
                  ------------------------------------
```

     --------- 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 or the non-mute macros \figwrit* .
  Only NDec decimals are printed, after rounding. If NDec is absent, then the number
  of decimals to print is obtained from last call to \figset write (rounding=...)).
  The default value is 2.
\figround{RealNumber}
  Rounds a numerical value. To be used in the joined text argument of the macros
  that create points and the non-mute macros \figwrit* . The number of decimals
  displayed is set by \figset write (rounding=...). The value to be rounded is
  generally a computed value, e.g. by \figget.
\figset write(attribute1=value1, attribute2=value2,...)
  Setting the attributes the \figwriteXX macros depend on.
  The general calling sequence is
        \figset keyword (attribute1=value1, attribute2=value2,...)
  Here, the required keyword is "write". See the macro \figset in the section
  "Macros for graphical generation" for the other keywords. The attributes are
  given below, along with the possible values and their definition.
  The current values can be obtained with the help of \figshowsettings.
  - keyword = write
   . mark = symbol : controls how the point position is shown ; the point marker
              to be written can be a point (.) or $\bullet$ for example.
              By default, nothing is written.
   . ptname = text : controls the definition of the point name ; the default name
              for the point i is A_i: \figset write(ptname={$X^{(#1)}$})
              changes it to $X^{(i)}$ for example.
   . rounding = Nd : sets Nd as the number of decimals printed with \figcoord or
              \figround, after rounding.
  - other keywords : see "Control macros" in section "Macros for graphical generation".
\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)
`\figwritegce` 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.

`\figwritelw` Pt1, Pt2, ..., PtN :Text(Distance)
`\figwritele` Pt1, Pt2, ..., PtN :Text(Distance)
`\figwriteln` Pt1, Pt2, ..., PtN :Text(Distance)
`\figwritels` Pt1, Pt2, ..., PtN :Text(Distance)
`\figwriteln` Pt1, Pt2, ..., PtN :Text(Distance,lambda)
`\figwritels` Pt1, Pt2, ..., PtN :Text(Distance,lambda)

Writing the Text placed at a given Distance from the reference point associated
to each segment, towards the west, the east, the north or the south.
A segment [Pti, Pt{i+1}] is defined by two consecutive points in the list.
The reference point used to write the text is defined as follows:
 - when writing towards the west (resp. the east), the reference point is Pti
   (resp. Pt{i+1}),
 - when writing towards the north or the south, the reference point is Qi, the
   barycenter of (Pti, 1-lambda) and (Pt{i+1}, lambda). Thus, lambda = 0
   corresponds to Pti and lambda = 1 to Pt{i+1}. By default, lambda = 1/2.
The writing direction is parallel to each segment.
The argument Distance measures the shortest path between each reference point
(Qi, Pti or Pt{i+1}) and the bounding box of the Text.
By default, the text written is:
 - the text associated to point Pti (resp. Pt{i+1}), if X=w (resp. X=e),

```
      - the length (in user unit) of the segment, if X=n or X=s.
\figwritedim Pt1, Pt2 :Text(Distance)
  Write a Text (assumed to denote a dimension) at the distance Distance of the
  midpoint of the segment [Pt1, Pt2], oriented from west to east.  If Distance<0,
  the text is written at the south of the segment, at the north otherwise.
  If Text is empty, the value of the distance between Pt1 and Pt2 is written.
  In this case, the written value can be rounded with the help of the macro
  \figset write(rounding=...).
  This macro is the companion macro of \figdrawdim so that the arguments Pt1, Pt2
  and Distance are generally the same.
                    ---------------------------------
                    | Macros for graphical generation |
                    ---------------------------------


      --------- Control macros
\figdrawbegin{} or \figdrawbegin{FileName}
  Starting the creation of a graphical file whose name is FileName.
  If the argument FileName is empty or blank, an internal default file name is used,
  which is always updated, regardless the update mode.
  See \figinsert for details.
\figdrawend
  End of the current graphical file.
\figreset{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:
   - NO keyword or keyword = general to reset the general (or top-level) attributes,
   - altitude to reset the altitude attributes,
   - arrowhead to reset the arrowhead attributes,
   - curve to reset the (\figdraw)curve attributes,
   - general to reset the general (or top-level) attributes,
   - flowchart to reset the flow chart attributes,
   - mesh to reset the (\figdraw)mesh attributes,
   - sign to reset the (\figdraw)sign attributes,
   - trimesh to reset the (\figdraw)trimesh attributes.
\figset keyword (attribute1=value1, attribute2=value2,...)
  Setting graphical attributes or parameters acting on the appearance of the figure,
  which are to be given as pairs of the form  attribute = value.
  The general calling sequence is
          \figset keyword (attribute1=value1, attribute2=value2,...)
  For each keyword, the attributes are given below, along with the possible values
  and their definition.
  The current values can be obtained with the help of \figshowsettings.
  - NO keyword or keyword = general
    . alpha = transparency level :  makes lines and filled areas more or less transparent
      (real number between 0 (transparent, invisible) and 1 (opaque).
    . cap = butt, round or square :  line cap parameter that establishes the shape to be
      put at the ends of a straight line.  It may be usefull when wide lines are used.
    . color = color definition :  line or area color in cmyk or rgb color code,
      or in gray tone (real number between 0 (black) and 1 (white).
    . dash = index or dash pattern :  line style given by Index (Index = 1 to 10,
      1 meaning "solid line") or by Pattern.
    . fillmode = yes/no :  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.
```

92

. join = miter (V), round (U) or bevel (_/) : 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 : setting the update mode to "yes" (resp. to "no") before
  \figdrawbegin enforces (resp. avoids) the graphical file to be recreated at
  each compilation.
. width = dimension in PostScript unit : line width.
  Note : has no effect on a straight line after \figset (fillmode=yes).
- keyword = altitude
 All attributes are DDV (they have a dynamic default value).
 The following attributes affect the aspect of the extension of the base line drawn
 when the foot of the altitude lies outside the triangle:
. blcolor = color definition : controls the color of the base line,
. bldash = index or dash pattern : controls the style of the base line,
. blwidth = dimension in PostScript unit : controls the width of the base line.
 The following attributes affect the aspect of the square of the altitude:
. sqcolor = color definition : controls the color of the square,
. sqdash = index or dash pattern : controls the style of the square,
. sqwidth = dimension in PostScript unit : controls the width of the square.
- keyword = arrowhead
. angle = real in degrees : arrowhead opening half-angle.
. fillmode = yes/no : arrowhead filling switch, which tells whether the interior
  of the arrowhead must be filled or not.
. length = real in user unit : length of each edge of the arrowhead.
. out = yes/no : arrowhead "outside" switch, which tells whether the arrowhead
  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] : arrowhead ratio defining the arrowhead length so that
  it is equal to ratio * (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] : roundness of a C1 curve.
- keyword = flowchart
. arrowposition = real in [0,1] : arrowhead position along the path,
  0 corresponding to the beginning and 1 to the end of the path.
. arrowrefpt = start/end : reference point of the arrowhead to its start point or
  to its end point ; the arrowhead position is computed with respect to this point.
. bgcolor = color definition : controls the color of the frame background.
. line = polygon/curve : tells \figdrawfcconnect to draw the path between two nodes
  using polygonal lines or smooth curved lines.
. padding = real in user unit (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 unit : radius of the circular arcs to be
  drawn at the corners of the path when the line is polygonal.
. shape = circle, ellipse, lozenge or rectangle : shape of the frames.
. thickcolor = color definition : controls the color of the thickness of the frames.
. thickness = real in user unit : thickness of the frames.
. xpadding = real in user unit : horizontal space around the text inside a frame.
. ypadding = real in user unit : vertical space around the text inside a frame.
 Notice that the frame border color is the general color.
- keyword = mesh

```
              .  diag = integer in {-1,0,1} :  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.
             The following DDV attributes affect the aspect of the lines used to draw the mesh
             inside the quadrangle:
              .  color = color definition :  controls the color of the lines,
              .  dash = index or dash pattern :  controls the style of the lines,
              .  width = dimension in PostScript unit :  controls the width of the lines.
           - keyword = projection :  see "Control macros" in section "Geometrical macros - Mute macros".
           - keyword = sign
              .  dimension = positive real in user unit :  characteristic dimension of the sign,
              .  shape = circle, diamond, square, star, triangle :  shape of the sign.
             The following attributes affect the aspect of the lines used to draw the sign:
              .  color = color definition (DDV attribute) :  controls the color of the contour lines
                or their interior,
              .  dash = index or dash pattern :  controls the style of the contour lines,
              .  fillmode = yes/no :  sign filling switch, which tells whether the interior
                of the sign must be filled or not.
              .  join = miter (V), round (U) or bevel (\_/) :  line join parameter of the contour lines
              .  width = dimension in PostScript unit (DDV attribute) :  width of the contour lines.
           - keyword = trimesh
             The following DDV attributes affect the aspect of the lines used to draw the mesh
             inside the triangle:
              .  color = color definition :  controls the color of the lines,
              .  dash = index or dash pattern :  controls the style of the lines,
              .  width = dimension in PostScript unit :  controls the width of the lines.
           - keyword = write :  see "Control macros" in section "Writing macros - Non-mute macros".
\figsetdefault 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.  This is not applicable to DDV attributes.
  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 \figset for their definition.
  For dimension attributes, a unit should be specified, e.g.
   \figsetdefault arrowhead(length=10pt).
  - NO keyword or
    keyword = general:  transparency, cap, color, dash, fillmode, join, update, width,
  - keyword = arrowhead:  angle, fillmode, length, out, ratio,
  - keyword = curve:  roundness,
  - keyword = flowchart:  arrowposition, arrowrefpt, bgcolor, line, padding, radius,
                shape, thickness, xpadding, ypadding,
  - keyword = mesh:  diag,
  - keyword = sign:  dimension, shape.


           --------- Basic drawing macros
\figdrawcirc Center (Radius)
\figdrawcirc 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.
\figdrawline[Pt1,Pt2,...  ,PtN]
  Line defined by N points, closed if the last point number equals the first one.
\figdrawlineC(X1 Y1, X2 Y2,..., XN YN)
\figdrawlineC(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.
\figdrawlineF{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 ---
\figdrawarccirc Center ; Radius (Ang1,Ang2)
\figdrawarccirc 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 CP1 x CP2, starting
  from the half-line (C, P1), where C=Center, P1=Pt1, P2=Pt2.
\figdrawarccircP Center ; Radius [Pt1,Pt2]
\figdrawarccircP Center ; Radius [Pt1,Pt2,Pt3]
  Point version of \figdrawarccirc
  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 x Y.
  In 3D, N = CP1 x CP3, 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).
\figdrawarcell 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.
\figdrawarcellPA 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.
\figdrawarcellPP 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.
\figdrawarcparab [Pt1,Pt2,Pt3]
  Arc of parabola between Pt1 and Pt3.
  The point Pt2 defines the tangents (Pt2,Pt1) at Pt1 and (Pt2,Pt3) at Pt3.

  --- Arrow ---
\figdrawarrow [Pt1,Pt2]
  Arrow from Pt1 to Pt2.
  The arrowhead is drawn according to the \figdrawarrowhead macro settings.
\figdrawarrowBezier [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 arrowhead at point Pt4.

```
\figdrawarrowcirc Center ; Radius (Ang1,Ang2)
\figdrawarrowcirc 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 arrowhead is drawn according to the \figdrawarrowhead 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.
\figdrawarrowcircP Center ; Radius [Pt1,Pt2]
\figdrawarrowcircP Center ; Radius [Pt1,Pt2,Pt3]
  Point version of \figdrawarrowcirc
  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 x Y.
  In 3D, N = CP1 x 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).
\figdrawarrowhead [Pt1,Pt2]
  Arrowhead of the arrow from Pt1 to Pt2.  The segment [Pt1, Pt2] (body) is not drawn.
  The appearance of the arrowhead can be modified with the help of attributes set by
  the macro \figset arrowhead(...).
\figdrawaxes Origin(X1,X2, Y1,Y2)
\figdrawaxes Origin(X1,X2, Y1,Y2, Z1,Z2)
\figdrawaxes 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 \figdrawaxes Origin(L) is equivalent to \figdrawaxes Origin(0,L, 0,L)
  in 2D and \figdrawaxes Origin(0,L, 0,L, 0,L) in 3D.
  The end points of the axes are given by \figptsaxes.
\figdrawdim Pt1, Pt2 (Distance)
  Double arrow intended to show the dimension between the points Pt1 and Pt2.
  Assuming the segment [Pt1, Pt2] is oriented from west to east, the arrow is
  drawn at the distance Distance at the south of the segment if Distance<0,
  at the north otherwise.
  The endpoints of the arrows are connected to Pt1 and Pt2 with a dashed line.
  The appearance of the two arrow-heads can be modified with the help of
  attributes set by the macro \figset arrowhead(...).
  The distance between the points Pt1 and Pt2 can be printed by the companion
  macro \figwritedim.


  --- Curve ---
\figdrawBezier N [Pt_1, ..., Pt_{3N+1}]
  Bezier curve defined by N cubic arcs.  The arc number i is defined by the four
  control points P_{j}, P_{j+1}, P_{j+2}, P_{j+3} with j=3i-2.
  The curve interpolates each 3 points beginning with the first, i.e. at P_{3i-2},
  i=1,...N+1.  At these points, the curve is only C0.  To obtain G1 continuity at P_j,
  the points P_{j-1}, P_j and P_{j+1} must be aligned.
  The total number of points must be 3N+1 and is not checked.
```

\figdrawcurve [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 \figset curve(...). The best values for this coefficient are
  in the interval [0.15, 0.3] (0 gives a polygonal line).

  --- Flow chart ---
\figdrawfcconnect[Pt1,Pt2,... ,PtN]
  Connections between nodes in a flow chart. Each connection path can be a polygonal
  line or a curved line. An arrowhead 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 \figset flowchart(...).
  The frames at the nodes are drawn with the help of the macro \figdrawfcnode.
\figdrawfcnode[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 \figset flowchart(...).
  The connections between the nodes are usually drawn by the macro \figdrawfcconnect.
  If no arrows are wanted, the best is to say \figset arrowhead(length=0) ; the macro
  \figdrawline or \figdrawcurve can also be used.

  --- Grid ---
\figdrawmesh 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
  \figset mesh(diag=...) allows to draw a diagonal line inside each cell. Other
  graphical settings govern the appearance of the internal mesh (see \figset mesh).
  The border line attributes are the general ones.
\figdrawtrimesh Type [Pt1,Pt2,Pt3]
  Mesh of the type Type triangle in the triangle (Pt1, Pt2, Pt3). The internal
  mesh is drawn according to specific graphical settings (see \figset trimesh).
  The border line attributes are the general ones.

  --- Sign or symbol ---
\figdrawsign[Pt1,Pt2,... ,PtN]
  Draw the N points given by their numbers.
  The position of a point is displayed by the mean of a marker ; a characteristic
  dimension is associated to this marker. By default, the marker is a circle and
  its characteristic dimension is its radius. Other possible markers are diamond,
  square, star and triangle. The marker and its dimension can be set or changed
  by the macro \figset sign(...). Other attributes (color, fillmode...) can be
  modified by the macro \figset (...).
\figdrawsignC(X1 Y1, X2 Y2,..., XN YN)
\figdrawsignC(X1 Y1 Z1, X2 Y2 Z2,..., XN YN ZN)
  Draw the N points given by their coordinates, each of them separated by a white
  space, each group of coordinates separated by a comma. They are assumed
  to be given as decimal integers or fixed floating point numbers ; scientific
  notation (e.g. 1.23e+2) is not allowed.
  See \figdrawsign for more information about signs.

\figdrawsignF{FileName}
  Draw the 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. They are assumed
  to be given as decimal integers or fixed floating point numbers ; scientific
  notation (e.g. 1.23e+2) is not allowed.
  See \figdrawsign for more information about signs.

  --- Triangle related macros ---
\figdrawaltitude Dim [Pt1,Pt2,Pt3]
  Altitude from Pt1 in the triangle (Pt1, Pt2, Pt3).
  Dim is the dimension of the square at the foot H of the altitude which is drawn on
  either side of the altitude according to the order of the 2 points Pt2 and Pt3.
  If H is outside the segment [Pt2, Pt3], a line is drawn to support the square.
  The graphical attributes of this line and the square are set by \figset altitude(...).
\figdrawnormal Length,Lambda [Pt1,Pt2]
  Exterior normal of length Length to the segment [Pt1, Pt2]. Lambda is the abscissa
  in [0,1] of the origin of the normal with respect to the segment [Pt1, Pt2],
  0 corresponding to Pt1 and 1 to Pt2. This sets the position of the normal vector
  along [Pt1, Pt2]. In 3D-mode, it works only in the plane Z=0.

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.