TransFig: Portable Figures for $\TeX$

Micah Beck*

TR 89-967
February 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This research was supported by NSF grant DCR 85-21394

TransFig: Portable Figures for TeX
Version 1.4-TFX Release 4

Micah Beck
Department of Computer Science
Upson Hall, Cornell University
Ithaca, New York 14853

TransFig is a mechanism for integrating figures into TeX documents. Several "graphics languages" exist which achieve such integration, but none is widely enough used to be called a standard. TransFig's goal is to maintain the portability of TeX documents across printers and operating environments. The central mechanism in TransFig is Fig code, a graphics description format which is produced by the Fig interactive graphics editor. TransFig provides an automatic and uniform way to *Trans*late *Fig* code into various graphics languages and to integrate that code into a TeX document.

1 TransFig

The TransFig package includes several translators between Fig code and other graphics languages (see figure 1). Programs shown in dashed boxes are not part of the TransFig distribution, but are compatible with it. These translators can be used directly to produce various graphics languages from Fig code, but do not in themselves provide document portability. Each graphics language requires different macro files to be loaded, or has a different command for specifying that a figure be included in the document.

TransFig allows these differences to be hidden behind a uniform TeX interface. If this interface is followed, then a `Makefile` created by the `transfig` command will translate Fig code into any of these graphics languages without changing the TeX document. To change to a different graphics language, it is necessary only to rerun `transfig`, and then `make`. `Make` can also be used to keep the translated code up to date when figures change.

·The author's Internet mail address is `beck@cs.cornell.edu`

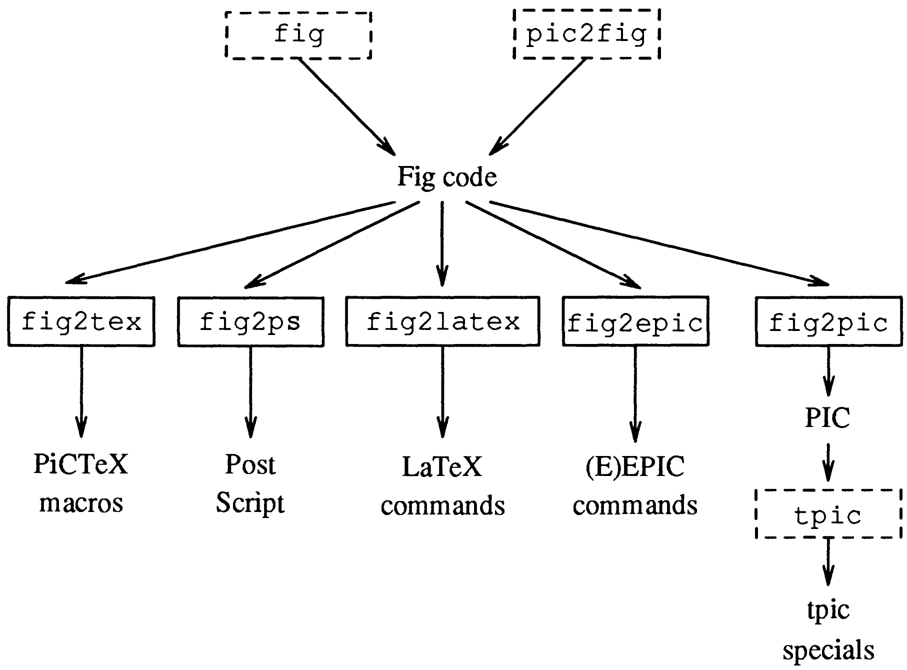

Figure 1: Fig Code Translations

The graphics in this manual were created using Fig and integrated into the document using TransFig.

1.1 File Name Conventions

Suppose that a document is to include a set of figures which are stored in Fig code form. These should be in files with the name suffix `.fig`, for instance `figure1.fig`, `figure2.fig`,...`figuren.fig`. TransFig will create files `figure1.tex`, `figure2.tex`,...`figuren.tex` for `\input` to the TEX document, and in some cases will create files with other suffixes. Additionally, TransFig creates a file named `transfig.tex` which must be `\input` at the start of the document.

```
\documentstyle{article}
\input{transfig}            TransFig macro file
\begin{document}
            ⋮
\begin{figure}
  \begin{center}
  \input{figurei}           i'th TransFig figure
  \end{center}
\end{figure}
            ⋮
```

Figure 2: Layout of a TransFig TEX Document

1.2 Transfig TEX Documents

In order to use TransFig, a TEX file must follow the format shown in figure 2. The the file `transfig.tex` must be `\input` before any TransFig figure is encountered. At the point where the ith figure `figurei` is to be inserted, the file `figurei.tex` is `\input`.

1.3 Using TransFig

The `transfig` command has the form

`transfig` [*option*]... [[*control*]... *filename*]...

Where *option* is one of the following:

- **-L** *language* to translate into the specified *language* (default `pictex`).
- **-M** *makefile* to name the output makefile *makefile* (default `Makefile`).
- **-T** *texfile* to name the output TEX macro file *texfile* (default `transfig.tex`).

The *language* specifiers `pictex`, `postscript`, `latex`, `(e)epic` and tpic, indicate translation into PiCTEX macros, PostScript, LATEX picture environment, (E)EPIC macros, or `tpic` specials, respectively. See section 2 for further details about these languages. The special *language* specifier `null` causes the figures to be replaced by empty boxes of the appropriate size.

A *control* specifier sets a parameter which governs the translation of all files to its left in the argument list, until it is overridden. A *control* specifier must be one of the following:

-m *magnification* to scale figures by *magnification* (default 1.0).

-f *font* to set the default font family (default `cmr`).

-s *size* to set the default font size (default $12 * magnification$).

Each file name specifies a Fig file, either with or without the `.fig` suffix. TransFig creates a file called `Makefile` to apply the appropriate translator to the named files, and creates an appropriate `transfig.tex` file. Thus, to create a `Makefile` which will translate all figures in a directory to LaTeXpicture environment, with Computer Modern Bold as the default font family, the command would be

```
transfig -L latex -f cmb *.fig
```

After running `transfig`, simply run `make` to create the appropriate TeX files. `Make` should be rerun whenever a Fig file is changed to recreate the corresponding TeX file. To change between graphics languages, simply run `make clean` to remove the files created by `transfig`, and then rerun `transfig`.

1.4 Text in Figures and Portability

In order to be translatable into different graphics languages, Fig code used in TransFig documents should use only those features which are supported by all of them. In particular, some graphics languages support more sophisticated processing of text which is part of the figure than others. PiCTeX, for example, allows full use of TeX commands in text strings, while PostScript does not.

The standard way to use text in TransFig figures is to use only straight text with no TeX commands. The text font is determined by the translator or the graphics language. A document which makes use of TeX commands in text can still be created as a TransFig document, but it will not be portable.

Similarly, some graphics languages have controlling commands which are not understood by all (such as PiCTEX's `\nopictures`). To create a portable TransFig document, these commands should not be used. However, if portability is not an issue (as when developing a document), then they can be used at any point after the `\input{transfig}` command.

2 Graphics Languages and Fig Translators

TransFig's goal is to provide a framework for including graphics which maintains the portability of TEX documents across printers and operating environments. The central mechanism in TransFig is Fig code, a graphics description format which is produced by the Fig interactive graphics editor. If this code is widely used as an intermediate form for figures, the builders of other graphics tools may be attracted to produce compatible output. The reference guide in appendix A describes Fig code in more detail.

2.1 Translations From Fig

TransFig currently translates Fig code into these graphics languages: PiCTEX, PostScript, LATEX picture environment, and (E)EPIC. The Fig translation programs corresponding to these languages are `fig2tex`, `fig2ps`, `fig2latex`, and `fig2epic`. The TransFig package also includes Fig2pic, which translates Fig code into the PIC graphics language. The `transfig` command supports the translation of Fig code into `tpic` specials (see below) using translation programs `fig2pic` and `tpic`, which is not part of the TransFig package. Each language may be appropriate in different operating environments or for different applications. A short description of each language is given below:

PiCTEX is a set of TEX macros which implement simple graphics objects directly in TEX. PiCTEX makes no use of pre- or post-processors; the DVI files it generates are completely standard, and can be printed or previewed in any environment where TEX is used. This result is achieved by using TEX integer arithmetic to do all plotting calculations, and by drawing the figure using the period character as a "brush". As a result PiCTEX is quite slow and requires a large internal TEX memory.

PostScript (PS) is a powerful graphics language which is gaining acceptance as a standard. In an environment where DVI code is translated into PS before being printed, it is usually possible to insert a separately generated PostScript file into a document, using the TEX `\special` command. However, the resulting PS file can only be previewed using a PS previewer, and must be printed on a PS printer, such as the Apple LaserWriter.

LATEX picture environment is a restricted graphics facility implemented within LATEX. It is a standard part of every version of LATEX, is processed quickly, and does not require a large internal TEX memory. However, not every graphics object which can be described with Fig code can be drawn using the LATEX picture environment. Restrictions include a limited set of slopes at which lines can be drawn, and no ability to drawn splines.

EPIC is an enhanced version of the LATEX picture environment which removes many restrictions. It uses no facilities outside of those needed for the LATEX picture environment.

EEPIC is a further enhancement of EPIC which uses `tpic` specials to implement general graphics objects. It is subject to the same software requirements as `tpic`.

`tpic` specials are a set of `\special` commands which produce graphics instructions in the DVI file produced by TEX. However, the graphics in the resulting DVI file can only be previewed or printed using software which understands these commands.

When TEX processes the file `transfig.tex`, it will print the message "`TransFig: figures in` *language*" indicating which graphics language is in use.

2.2 The Fig Graphics Editor

The interpretation of Fig code was originally defined by the Fig graphics editor and the program `f2p`, which translates Fig code into the PIC graphics language. The Fig code reference guide in appendix A is derived from the file `FORMAT1.4` which is distributed with Fig Version 1.4 Release 2. It reflects the operation of that release of Fig, and indicates intended uses for certain fields which were not yet defined.

It is possible to extend the description of Fig code by specifying interpretations for the undefined fields. This has been done in the TransFig extension to Fig code (TFX), which is described in appendix B. TFX is supported by the TransFig translators, and by a version of the Fig graphics editor called Fig 1.4.FS.

Certain features of Fig code are specified by the file `FORMAT1.4` and so are described in appendix A, but are not implemented in Fig Release 2. Fig 1.4.FS implements these features, as well as the TFX extension. In addition, it implements various enhancements to the Fig user interface.

2.3 The PIC to Fig Translator

The PIC-to-Fig translator allows graphics to be described in PIC, the language of Brian Kernighan's graphics preprocessor for Troff. The Fig code produced by `pic2fig` is compatible with that produced by Fig, and can be edited with Fig. This translator allows users to create figures without employing a graphics editor. `Pic2fig` is a modified form of the `tpic` program, which was adapted from PIC itself.

3 Related Software

Software availability is subject to change, and this list may not be completely up to date.

EPIC is an enhancement of the LaTeX picture environment which removes many restrictions. It uses only the facilities which implement the LaTeX picture environment. EPIC was developed by Sunil Podar at the State University of New York at Stonybrook, and is available via anonymous FTP from `CS.CAYUGA.ROCHESTER.EDU`.

EEPIC is a further enhancement of EPIC which uses `tpic` specials (see below) to implement general graphics objects. It is subject to the same software requirements as `tpic`, although there is an "emulation package" which will implement most of EEPIC using the same facilities as EPIC. EEPIC was developed by Conrad Kwok at the University of California at Davis, (`kwok@IRIS.UCDAVIS.EDU`), and is available via anonymous FTP from `IRIS`.

Fig is an interactive graphics editor in the style of MacDraw which runs under the Suntools/SunView windowing system. It produces intermediate code which can be translated into a variety of graphics languages, including PIC, Postscript, and PiCTEX.

Fig was developed by Supoj Sutanthavibul at the University of Texas at Austin, (`supoj@SALLY.UTEXAS.EDU`), and is available via anonymous FTP from `SALLY`.

Fig 1.4.FS is a version of Fig Version 1.4 Release 2 which implements various enhancements to the user interface, and also uses the TFX extension to the definition of Fig code. Fig 1.4.FS was developed by Frank Schmuck at Cornell University and is supported by Micah Beck (`beck@CS.CORNELL.EDU`). It is available via anonymous FTP from `SVAX.CS.CORNELL.EDU`.

F2p, F2ps translate from Fig code to the PIC graphics language and to PostScript, respectively. These programs are distributed along with some versions of Fig, but updated versions (Fig2pic and Fig2ps) are available as part of the TransFig package.

Fig2tex, Fig2ps, Fig2latex, Fig2epic, Fig2pic translate from Fig code to PiCTEX macros, Postscript, LATEX picture environment commands, (E)EPIC macros, and the PIC graphics language, respectively. They are part of the TransFig package, and support the TFX extension to Fig code. Fig2pic and Fig2ps are updated versions of F2p and F2ps respectively.

LATEX is a standard macro package used for describing documents in TEX. Part of this package is the LATEX picture environment, a restricted graphics facility. The capabilities of this facility are described in section 5.5 of LaTeX, *A Document Preparation System* by Leslie Lamport.

Pic2fig is a version of Brian Kernighan's PIC graphics preprocessor for Troff. Pic2fig, which is a modified form of `tpic` (see below), has been altered to produce Fig code. For Pic2fig distribution information, contact Micah Beck of Cornell University (`beck@CS.CORNELL.EDU`).

PiCTeX is a set of macros for describing graphics in TeX documents. PiCTeX is implemented entirely within standard TeX, and requires no pre- or post processing programs or special fonts. The main problem in using PiCTeX is its slow operation (all calculations are done using TeX's integer arithmetic) and large memory requirements. Many PiCTeX users have turned to C implementations of TeX in order to obtain memory sizes larger than are possible using the standard Web/Pascal version.

PiCTeX was developed by Michael Wichura at the University of Chicago (`wichura@GALTON.UCHICAGO.EDU`), and is available via anonymous FTP from `A.CS.UIUC.EDU`. It is also included as contributed software with the Unix TeX distribution.

`tpic` is a version of Brian Kernighan's PIC graphics preprocessor for Troff. `tpic` has been altered to produce TeX `\special` commands which are understood by some DVI print drivers and previewers. For information about distribution of `tpic`, contact Tim Morgan of the University of California at Irvine (`morgan@ICS.UCI.EDU`).

TransFig was developed by Micah Beck at Cornell University (`beck@CS.CORNELL.EDU`), with major contributions by Frank Schmuck, now of IBM, and Conrad Kwok of UC Davis. It is available via anonymous FTP from `SVAX.CS.CORNELL.EDU`.

Xfig is a version of the Fig graphics editor which can be compiled for either the Suntools or X Windows Version 11 windowing systems. Xfig was developed by Ken Yap at Rochester University (`ken@CAYUGA.CS.ROCHESTER.EDU`) and is available via anonymous FTP from `CAYUGA`.

A Fig Code Reference Guide

This guide describes the code produced by Fig version 1.4, which is not compatible with the code produced by Fig version 1.3. A Fig 1.4 code file has the following structure:

```
#FIG 1.4
```
global parameters
object description
object description

⋮

A.1 Comment Lines

The very first line is a comment line containing the version of the Fig format. Programs which interpret Fig code verify compatibility by checking the first line for this comment. All other lines which contain the character # in the first column are treated as comments and are ignored.

A.2 Global Parameters

The first non-comment line consists of two global parameters:

```
fig_resolution coordinate_system
```

Fields in a line of a Fig file are separated by blanks or tabs; newlines terminate object descriptions. The fields of lines in Fig files are described throughout this guide by tables like the one below. The fields must appear in the order given in the table.

Type	Field	Units (values)
int	`fig_resolution`	pixels/inch Fig value: 80
int	`coordinate_system`	1: origin at lower left corner 2: origin at upper left corner Fig value: 2

The *Type* column specifies the type of the field, and is either int(eger), float, or string. The notation + following the type indicates that the values 0 or -1 are interpreted as *default* values in this field. The rightmost column of this table either defines the units in which the field is expressed, or lists the possible values which the field can take. The notation `DEFAULT` in this column indicates that no value other than the default values are allowed. It is intended that future versions of Fig will define other values for these fields, but that the default values will remain legal, thus providing backward compatibility. These *defaulted fields* are discussed further in section A.5

The basic unit of position in Fig files is the pixel. While figures in a Fig file are described at this resolution, the figure can be drawn at a higher or lower resolution. Pixels are square, and so `fig_resolution` represents position resolution in both the x and y dimensions.

Some values are expressed as symbols and their numerical values are also listed. These symbols are defined in the header file `object.h`.

A.3 Object Descriptions

The rest of the file contains objects descriptions, having one of six types:

1. Ellipse.

2. Polyline, including Polygons and Boxes.

3. Spline, including Closed/Open Control/Interpolated Splines.

4. Text.

5. Circular Arc.

6. Compound object which is composed of one or more objects.

The following group of common fields appear in several object descriptions, and so the are described here, and later are simply referred to by the indicator *common fields.*

Type	Field	Units (values)	
int	line_style	SOLID_LINE	0
		DASH_LINE	1
		DOTTED_LINE	2
int	line_thickness	pixels	
int +	color	DEFAULT	
int +	depth	DEFAULT	
int +	pen	DEFAULT	
int +	area_fill	DEFAULT	
float	style_val	pixels	

For the dashed line style, the `style_val` specifies the length of a dash. For dotted lines it indicates the gap between consecutive dots.

Arrow lines are used to describe optional arrows at the ends of Arc, Polyline, and Spline objects. If an object has a forward arrow, then an arrow line describing it follows the object description. If an object has a backward arrow, then an arrow line describing it follows the object description and the forward arrow description, if there is one.

An arrow line consists of the following fields

Type	Field	Units (values)
int +	arrow_type	DEFAULT
int +	arrow_style	DEFAULT
int +	arrow_thickness	DEFAULT
int	arrow_width	pixels
int	arrow_height	pixels

A.3.1 Ellipse Objects

Type	Field	Units (values)	
int	object_code	O_ELLIPSE	1
int	sub_type	T_ELLIPSE_BY_RAD	1
		T_ELLIPSE_BY_DIA	2
		T_CIRCLE_BY_RAD	3
		T_CIRCLE_BY_DIA	4
	common fields		
int	direction	1	
float	angle	radians	
int	center_x, center_y	pixels	
int	radius_x, radius_y	pixels	
int	start_x, start_y	pixels	
int	end_x, end_y	pixels	

The Ellipse object describes an ellipse (or circle) centered at the point (center_x, center_y) with radii radius_x and radius_y, and whose x-axis is rotated by angle from the horizontal. If the object describes a circle, then radius_x and radius_y must be equal.

The fields start_x, start_y, end_x and end_y are used only by Fig, and are not used in drawing the object. If the ellipse is specified by radius, then (start_x, start_y) is (center_x, center_y), and (end_x, end_y) is a corner of a box which bounds the ellipse. If the ellipse is specified by diameter, then (start_x, start_y) and (end_x, end_y) are the two corners of the box which bound the ellipse.

A.3.2 Polyline Objects

Type	Field	Units (values)	
int	object_code	O_POLYLINE	2
int	sub_type	T_POLYLINE	1
		T_BOX	2
		T_POLYGON	3
	common fields		
int	forward_arrow, backward_arrow	0: no arrow 1: arrow	

The Polyline object description has an addition *points line* following any arrow lines. The line consists of a sequence of coordinate pairs followed by the pair 9999 9999 which marks the end of the line.

x_1 y_1 x_2 $y_2 \ldots$ x_n y_n 9999 9999

Type	Field	Units (values)
int	x_i, y_i	pixels

The Polyline object describes a piecewise linear curve starting at the point (x_1, y_1) and passing through each point (x_i, y_i) for $i = 2 \ldots n$. If sub_type is T_BOX or T_POLYGON then (x_1, y_1) and (x_n, y_n) must be identical. If sub_type is T_BOX, then the line segments must all be a vertically oriented rectangle.

A.3.3 Spline Objects

Type	Field	Units (values)	
int	object_code	O_SPLINE	3
int	sub_type	T_OPEN_NORMAL	0
		T_CLOSED_NORMAL	1
		T_OPEN_INTERPOLATED	2
		T_CLOSED_INTERPOLATED	3
		common fields	
int	forward_arrow, backward_arrow	0: no arrow 1: arrow	

The Spline object description has a *points line* following any arrow line which has the same format as described above for the Polyline object description. If the sub_type of the spline is T_OPEN_INTERPOLATED or T_CLOSED_INTERPOLATED, then an additional *control points line* follows the points line. The line consists of a sequence of coordinate pairs, two coordinate pairs for each point in the points line.

lx_1 ly_1 rx_1 ry_1 lx_2 ly_2 rx_2 $ry_2 \ldots$ lx_n ly_n rx_n ry_n

Type	Field	Units (values)
float	lx_i, ly_i, rx_i, ry_i	pixels

The interpretation of Spline objects is more complex than of other object descriptions, and is discussed in section A.4.

A.3.4 Text Objects

Type	Field	Units (values)	
int	object_type	O_TEXT	4
int	sub_type	T_LEFT_JUSTIFIED	0
		T_CENTER_JUSTIFIED	1
		T_RIGHT_JUSTIFIED	2
int +	font	DEFAULT	
int +	font_size	DEFAULT	
int +	pen	DEFAULT	
int +	color	DEFAULT	
int +	depth	DEFAULT	
float	angle	radians	
int +	font_style	DEFAULT	
int	height, length	pixels	
int	x, y	pixels	
string	string		

The positioning of the string is specified by the `sub_type`. The values `T_LEFT_JUSTIFIED`, `T_CENTER_JUSTIFIED`, and `T_RIGHT_JUSTIFIED` specify that (`x`, `y`) is the left end, center and right end of the baseline, respectively. The `height` and `length` fields specify the box that the text fits into. These specifications are accurate only for the fonts used by Fig.

The `string` field is an ascii string terminated by the character `'\01'`. This terminating character is not a part of the string. Note that the string may contain the new-line character `'\n'`.

A.3.5 Arc Objects

Type	Field	Units (values)
int	`object_code`	`O_ARC` 5
int	`sub_type`	`T_3_POINT_ARC` 1
	common fields	
int	`direction`	0: clockwise 1: counter
int	`forward_arrow`, `backward_arrow`	0: no arrow 1: arrow
float	`center_x`, `center_y`	pixels
int	$x_1, y_1, x_2, y_2, x_3, y_3$	pixels

The Arc object describes a circular arc centered at the point (`center_x`, `center_y`), starting at (x_1, y_1), passing through (x_2, y_2), and ending at (x_3, y_3). It is drawn either clockwise of counter-clockwise as specified by `direction`. Note that this description is quite overdetermined, as the center and direction of the arc can be deduced from the three points of the arc which are specified.

A.3.6 Compound Objects

Type	Field	Units (values)
int	`object_type`	`O_COMPOUND` 6
int	`upperright_corner_x`	pixels
int	`upperright_corner_y`	
int	`lowerleft_corner_x`	
int	`lowerleft_corner_y`	

The Compound object description describes a compound object bounded by the rectangle determined by the points

(`upperright_corner_x`, `upperright_corner_y`)
(`lowerleft_corner_x`, `lowerleft_corner_y`)

It consists of all the objects following it until an object whose `object_type` field is `O_END_COMPOUND` (-6) is encountered. Compound objects may be nested.

A.4 Splines

Specifying the interpretation of a Spline object description is more problematic than other graphics objects. A graphics object description can be viewed as having two parts: an abstract description of the locus of points which make up the object; and a set of appearance parameters which specify how the abstract object is to be represented. For example, a circular arc has a very precise and well understood abstract definition, independent of the width of the line used to draw it. Unfortunately, the abstract specification of splines is more complex. The following descriptions come at second hand; the author of this guide is not versed in spline algorithms, and so may have garbled them. Hopefully, they will give the knowledgeable reader some idea of the intended meaning of Spline objects.

Fig splines come in two major varieties: B-splines and Interpolated splines. Each of these is available in open or closed versions. If the `sub_type` field has the values `T_OPEN_NORMAL` or `T_CLOSED_NORMAL` then it describes a B-spline. In these cases, the points line which follows contains the `control points` for the spline. The spline does not actually pass through these points, but they determine where it will pass, which is generally quite close to the control points. B-splines are quite smooth.

If the `sub_type` field has the values `T_OPEN_INTERPOLATED` or `T_CLOSED_INTERPOLATED` then it describes an interpolated spline. In these cases, the points line which follows contains the *interpolation points* through which the spline will pass. In addition, a *control points* line follows the points line, which specifies two control points $(\mathtt{lx}_i, \mathtt{ly}_i)$ and $(\mathtt{rx}_i, \mathtt{ry}_i)$ for each interpolation point. The `i'th` section of the interpolated spline is drawn using the Bezier cubic with the four points $(\mathtt{x}_i, \mathtt{y}_i)$, $(\mathtt{rx}_i, \mathtt{rx}_i)$, $(\mathtt{lx}_{i+1}, \mathtt{ly}_{i+1})$, and $(\mathtt{x}_{i+1}, \mathtt{y}_{i+1})$. Interpolated splines are not as smooth as B-splines.

For either type of closed splines, the first and last points on the point line $(\mathtt{x}_1, \mathtt{y}_1)$ and $(\mathtt{x}_n, \mathtt{y}_n)$ are identical. For closed interpolated splines, the last pair of control points on the control points line, $(\mathtt{lx}_n, \mathtt{ly}_n)$ and $(\mathtt{rx}_n, \mathtt{ry}_n)$ are the same as $(\mathtt{lx}_1, \mathtt{ly}_1)$ and $(\mathtt{rx}_1, \mathtt{ry}_1)$ respectively.

A.5 Defaulted Fields

Certain fields can only take a *default* value; the values 0 and -1 are both used as defaults. They are identified in the field tables by the notation `DEFAULT` in the the rightmost column. It is intended that future versions of Fig code will define other values for these fields, but that the default value will remain legal, thus providing backward compatibility. One extension to Fig code is TFX, discussed in appendix B, which defines fields for text font specification and area fill.

The intended use of some of these fields is discussed below:

`depth` This value adds a half dimension to Fig. It is useful when we have overlapping filled objects and we want one to obliterate another. An object can have only one depth (including compound objects). An object that is in less depth can obscure the one with greater depth if they overlap.

`area_fill` The stipple pattern (which will be aligned) for filling object internals. For example, a filled arc will look like a piece of pie.

`pen` This will define the shape of pen used in drawing objects. It also includes the the stipple pattern for line filling. The default pen is a circular pen with black filling.

`color` The color used to draw an object and its area fill.

B TransFig Extension to Fig Code (TFX)

Section A.5 discusses fields which are not used in Fig code version 1.4. These fields have intended uses, but the interpretation of values in these fields are not defined. This section describes an extension to the interpretation of Fig code which gives an interpretation to some of these fields. Because this extension is implemented by the Fig code translators in the TransFig package, it is called the TransFig eXtension to Fig code, abbreviated TFX.

A TFX Fig code file will be correctly processed by Fig code translators which do not implement TFX; the fields defined by the extension are ignored.

Conversely, since TFX extends the definition of Fig code, non-TFX Fig files are correctly processed by TFX translators.

The first line of a TFX Fig Code file is modified to indicate that the extension is used.

```
#FIG 1.4-TFX
```

The following Fig code fields are defined by TFX.

Type	Field	Units (values)	
int	`depth`	no units	
int +	`area_fill`	`DEFAULT`	
		`BLACK_FILL`	1
		`DARK_GRAY_FILL`	2
		`MEDIUM_GRAY_FILL`	3
		`LIGHT_GRAY_FILL`	4
		`WHITE_FILL`	5

Type	Field	Units (values)	
`int +`	`font`	`DEFAULT`	
		`ROMAN`	1
		`BOLD`	2
		`ITALICS`	3
		`MODERN`	4
		`TYPEWRITER`	5
`int +`	`font_size`	points	

Depth, discussed in section A.5, determines which filled objects will obscure other objects. Many graphics languages cannot fully implement area fill. If the `font` (or `font_size`) field holds a default value, then the object is assigned a font (or size) which approximates the appearance of the fonts used by the Fig graphics editor. The `font_style` field is not used.