TransFig: Portable Graphics for TEX

Micah Beck
Department of Computer Science, Cornell University, Ithaca, NY 14853
(607) 255-8597. Internet: beck@cs.cornell.edu

Alex Siegel
Department of Computer Science, Cornell University, Ithaca, NY 14853
(607)-255-1165. Internet: siegel@cs.cornell.edu

Abstract

The TransFig software package defines a portable description language for technical graphics. Translations are provided from this language to commonly used graphics description formats, which can then be included in typeset documents. TransFig includes a particularly convenient framework for including figures in LaTeX. The graphics language defined by TransFig facilitates the interchange of structured, modifiable graphics between applications. In this paper, we review our experience with TransFig to argue the need for a standard *application level* graphics language, and suggest guidelines for its design.

Fig and TransFig

The Fig graphics editor was originally developed by Supoj Sutanthavibul at the Universtiy of Texas. Fig was designed to produce output in the language of the PIC graphics preprocessor for Troff, although it uses an editable intermediate file format which is quite independent of the output language. This *Fig code* format consists of a simple dump of Fig internal data structures. Fig was distributed from the University of Texas with two translators: from Fig code to PIC and to PostScript.

TransFig. Neither of the output forms supported by Fig allowed inclusion of Fig graphics in TEX documents in the operating environment of the Computer Science Department at Cornell University. To make such inclusion possible, Micah Beck developed a translator from Fig code to PICTEX macros [Wichura]. Frank Schmuck, also at Cornell, developed a translator to LaTeX picture environment macros; the generality of this translation was restricted by limitations of the target language. These two translators, together with those developed at Texas, and a translation to the EPIC and EEPIC macro packages developed by Conrad Kwok at the University of California, Davis, were combined to create a single package for *Trans*lating *Fig* code [Beck].

TransFig was developed with two high level goals:

1. to define a useful graphics intermediate form with a clear interpretation which can be implemented in any reasonably expressive graphics language.
2. to create a framework for the convenient inclusion of figures in TEX documents with no user customization due to the choice of graphics language.

In order to create a widely used intermediate form quickly, it was decided to define a standard interpretation for the Fig intermediate format. A reference manual was developed which defines Fig code and its interpretion [Beck]. While this interpretation was derived from the Fig editor, it is independent of that implementation. The second goal was addressed in the UNIX computing environment by the Transfig program which is described in a later section.

Goals

TransFig should be evaluated in light of its specific goals; we will therefore look more closely at what TransFig does and does not attempt to achieve.

Expressiveness. The most important parameter in the design of TransFig is the class of graphical figures which is to be expressed. These figures,

which we call *technical graphics*, are combinations of *graphical primitives* with embedded text; bitmaps are not included. Primitives are simple lines and curves, with properties such as dotted or dashed lines, shading, and arrow heads; text properties include font and size.

Technical graphics are typically used to illustrate some idea or example. The content of such figures is transmitted mainly through the shape and labelling of primitives and their placement relative to one another. This should be considered in contrast to the pixel-level precision required to produce highly detailed or realistic images (For examples, see Appendix A).

Restricting our interest to technical graphics limits the possible uses for TransFig. On the other hand, it allows us to give a less precise interpretation to Fig code than is required for a general purpose graphics language such as PostScript. A less exact interpretation in turn eases the task of producing a correct implementation using a wide variety of output languages.

TransFig does not attempt to model the expressiveness of PostScript, its most flexible output form. The goal of portability leads TransFig to a level of expressiveness closer to the least common denominator of its output forms. This has led to a reluctance among some developers of Fig to maintain compatibility with the TransFig interpretation of Fig code.

Portability. Portability of graphics is a goal which underlies many other choices in the design of TransFig. We have mentioned portability as a limiting factor on the precision and expressiveness of the interpretation of Fig code; it also rules out local or non-standard interpretations. In this context, portability means that a document, including figures, can be moved between operating environments.

To illustrate this point, consider the specification of bitmap patterns for area fill. It would be possible to increase the flexibility of the area fill specification by using a local configuration file to map logical names of area fill types to actual bit patterns. This would, however, also reduce the portability of the resulting Fig code. For this reason the list of area fill patterns defined by TransFig is not locally extensible; the intent is for this list to be extended at the discretion of the developers of TransFig.

Ease of inclusion. One goal of TransFig is to allow the user to specify the location of a figure within a TEX document with a simple command which requires no information about the figure except the name of the Fig code file. This means that the TEX file produced by TransFig must include all the spacing information required for the proper placement of graphics relative to the surrounding text; the bounding box of the figure must be known.

The details of how to include figures described in Fig code in a document will be discussed later. The problem of calculating the bounding box points up one of the main problems of the definition of Fig code. Formatted text embedded in figures is not handled properly by TransFig, since the bounding box of the text is known only *after* it has been formatted.

Implementation

The current implementation of TransFig is a compromise; it meets some of the above goals, and meets others only partially. Further discussion of the current implementation can be found in the TransFig manual [Beck].

The Fig2dev program. All Fig code translation programs are derived from F2p, the original program written by Supoj Sutanthavibul to translate Fig code to PIC. The TransFig translators were named Fig2pic, Fig2ps, Fig2tex, Fig2latex, and Fig2epic to differentiate them from the original versions.

Recent releases of TransFig have combined these five translation programs into a single program called Fig2dev. This program consists of a common control structure which uses a standard subroutine interface to produce a specific output form. A specific translation is then implemented as a set of subroutines meeting this interface, much like an operating system device driver.

Output languages. The translations currently implemented by Fig2dev are from Fig code to the following output languages:

PICTEX, a general picture environment for TEX which uses only native TEX facilities [Wichura].

LATEX picture environment, a restricted graphics facility that uses special fonts which are a standard part of LATEX [Lamport].

EPIC (Extended Picture Environment), a more flexible extension of LATEX picture environment [Podar].

EEPIC (Extended EPIC), a generalization of EPIC which uses an extension of TEX's DVI output format [Kwok].

PostScript, a general graphics description language often proposed as an industry standard [Adobe].

PIC, a graphics language designed for the Troff typesetting program [Kernighan].

None of these output languages can be used to include any figure in all operating environments; taken together they provide a translation compatible with most environments. TEX has no native graphics facility, so each output language must strike a balance between generality and adherence to standards.

PICTEX draws lines using a text character, usually the period, as a pen. This strategy, together with the implementation of all calculations using TEX integer registers, allows graphics to be generated using only standard features. Formatting complex figures, however, is slow and can require a very large internal TEX memory.

LATEX picture environment uses special drawing fonts which are a standard part of LATEX; however, the class of figures which can be represented is quite restricted; slopes of lines are restricted to a small set, curves and area fill are not implemented at all.

EPIC is an extension of LATEX picture environment which can represent a broader, but still restricted, class of figures using the same LATEX drawing fonts.

EEPIC is a reimplementation of EPIC which uses a graphics extension of the DVI output format (tpic specials), and therefore requires non-standard software support.

PostScript is a very general graphics description language which requires non-standard software (and often hardware) support.

PIC figures require non-standard software support to be included in TEX documents (tpic specials).

The Transfig program. The goals of generality and portability are addressed by the Fig2dev program; the Transfig program provides ease of graphics inclusion, at least in the UNIX operating environment. Each figure in a document is represented by a separate Fig code file. In order to create a printable document, these figures must be translated to some TEX-compatible output language, and appropriate commands must be inserted in the TEX document. These commands will, in general, depend on the choice of output language. The Transfig program hides these details from the user by automating them.

The mechanics of including a set of figures expressed in a given graphics language can be divided into two parts: certain definitions required by all figures, and a particular set of commands for each figure. To allow the automatic generation of the initial definitions, the user must `\input` into the document the file `transfig.tex`, which will be created by Transfig. For each Fig code file named `figure.fig`, the user must input into the document the file `figure.tex`, which will also be created by Transfig.

The Transfig program takes as arguments an output language and the list of Fig code files. It creates an initial file of definitions `transfig.tex`, and it creates a `Makefile` which, when processed by the UNIX Make facility, invokes Fig2dev to translate each Fig code file into an appropriate TEX file.

The `transfig.tex` file generally inputs style or macro files specific to a given output language. The file `figure.tex` may be a large file of graphics commands. Some output forms, notably PostScript, require the creation of an additional file, which is given an appropriate suffix such as `figure.ps`. The file `figure.tex` will then contain TEX commands which make reference to the PostScript file.

TransFig compatibility. The most powerful aspect of TransFig is that it defines a non-proprietary *application level* language for the description and transfer of technical graphics. By application level, we mean a language which describes graphics primitives at a level high enough to be edited by users or conveniently translated to other forms. In contrast, PostScript is a description level language; it is impossible to recover the higher level primitives from PostScript, particularly the text formatting commands. TransFig is non-proprietary in the sense that it is not under the exclusive control of the developer of any particular software tool. It is based on the Fig graphics editor, but has a definition and interpretation of its own.

Many application level description languages have been defined; every structured graphics editor defines an intermediate format for storage of figures, and ultimately translates it to a printable form. Since such a storage format is seen only as a utility for one graphics editor, there is generally little attention paid to its design. The definition of the format is encoded in the programs which use it, and can change with every release. These proprietary graphics formats are not useful for interchange of graphics between applications.

Fig code is derived from the proprietary graphics format of the Fig graphics editor. In fact, recent developers of Fig have defined PostScript-oriented extensions to the format which are not compatible with the standard TransFig interpretation. To distinguish the TransFig definition of Fig code, we refer to it by its version identifier TFX (for TransFig

eXtension). TFX is a language with a fixed syntax and interpretation, albeit somewhat loosely specified. This makes it appropriate as a target language for other graphics applications.

The most flexible version of the Fig graphics editor currently available is Fig 1.4.FS, or Fig-FS, which supports all TFX features. Fig-FS is a version of Fig Version 1.4 Release 2, the last release distributed from Texas, enhanced by Frank Schmuck of Cornell, and runs under the SunView windowing system.

XFig 2.0 is the most recent version of Fig which runs under the X Windowing System. XFig has been developed by several people; Brian Smith of the Lawrence Berkeley Laboratory has made the most recent improvements. XFig supports one of two Fig code dialects for use as an intermediate language, TFX and its own 2.0 format; the choice is made at compile time. Fig code 2.0 is a PostScript-oriented extension to Fig code; the PostScript driver in recent versions of the Fig2dev program supports both dialects.

Several programs are currently compatible with TransFig and with one another through their use of TFX. These include:

Gnuplot, a numerical plotting program, which can produce output in TFX;

Pic2fig, which translates PIC into TFX;

Plot2fig, which translates the UNIX plot file format to TFX.

TransFig is a flexible and widely used tool for the portable exchange and inclusion of graphics. This success has come in spite of serious shortcomings which the definition of TFX has inherited from the initial implementation of Fig. In any case, it is worthwhile asking what the most appropriate application level graphics format for technical graphics in TEX documents would be.

Intermediate Languages

Fig code has serious shortcomings as an application level graphics description language. These problems include:

1. an unreadable syntax,
2. an ad hoc integration of text with graphics,
3. limited facilities for the creation and use of composite graphics objects.

The least important problem is the syntax; it is mainly troublesome to software developers. In order to be easily parsed using the C language I/O library, Fig code consists almost solely of numbers; strings are used only to represent text objects. A more readable syntax similar to that of PostScript would be preferable. The other points are more troublesome, and the first question to address is: why not settle on PostScript as a standard graphics language for TEX?

PostScript. It is commonly held that one graphics language will suffice for all document description needs, and that PostScript is the appropriate standard. As we have pointed out, the level at which graphics are described in PostScript is too low to be useful as an application level representation. The function served by Fig code is simply different from that served by more primitive languages.

If PostScript were accepted as the standard graphics language for TEX, no higher level standard would be needed to provide portability. On the other hand, PostScript is very general and a full implementation places a substantial and unnecessary burden on users of technical graphics. A simpler extension to the DVI format would suffice for that purpose.

A simple interface designed to meet a specific purpose can insulate a software system from changes in technology. This is an important function of non-proprietary document description languages like Fig code or the TEX DVI format. It is possible that PostScript will be superceded by another popular page description language; the TEX community should not be tied to a single low level interface.

Note that the choice of PostScript or some other language of equivalent power is not important for our purposes. The complexity of technical graphics does not change rapidly; like technical prose, figures which convey ideas are best expressed with simple constructs. The full flexibility of PostScript is not required, and may in fact distract the technical writer.

Embedded text. Integrating text into graphics turns out to be more of a problem than integrating graphical figures into text documents. The problem is that a graphical interpreter may not be able to deduce what the size and shape of the text will be after formatting. On the other hand, it is necessary to allow formatting of text, in order to give unity to the appearance of the text and figures, and because technical graphics often require complex equations to be embedded.

Because of this problem, the TransFig interpretation of Fig code does not allow any embedding of formatting commands in text objects. This strict interpretation is, however, unacceptable to users, and so formatting of text objects is a necessity. To illustrate the problem posed by mixing of formatted

and unformatted text, consider how the curly brace character ({), which has a special significance to TEX, is handled when it appears in a text object.

When the curly brace character appears in unformatted text, it must be escaped with a backslash (\{) to indicate that it is not to be interpreted as a control character by TEX. In formatted text, however, control characters are not escaped; the translation program must know which type of text is being handled.

Since Fig code does not provide a means for making the distinction, TransFig takes a heuristic approach: text that has either the font or size field set to a non-default value is assumed not to include formatting commands; a text object which has both text properties set to the special default value is allowed to include such commands. A better way to deal with this problem is by differentiating between three distinct types of text: plain, formatted, and special.

Plain text contains no formatting commands. Paroperties such as font and size are specified as properties of the text object, but do not appear in the text itself. This treatment of text is the most common in graphics editors.

Formatted text contains simple formatting commands in a language which is part of the definition of the intermediate language. An appropriate choice might be a subset of LATEX.

Special text can be expresssed in any formatting language; it is not interpreted but is passed through to the output language unchanged. The specific formatter used can be specified as a property, or omitted.

The properties of plain or formatted text must be part of the definition of the language. Furthermore, a reasonably powerful graphics editor should be able to display formatted text. A very sophisticated editor might actually invoke a formatting program to generate output for special text. To allow sufficient space to be left in the containing document, however, the bounding box of special text must be explicitly specified as a property of the text object.

Intermediate language design goals. The design of an intermediate graphics representation is a complex matter, with many trade-offs to consider. We can, however, list a number of design goals for such a language:

Fast load and store: since this language will be used by an editor as the main form of graphics storage, it must be very efficiently loaded from and stored to a file.

Fast display: graphics previewers and editors must be able to interpret the language with minimal computational overhead.

Easy conversion to other formats: translation of this language to other languages such as PostScript will be very common. Unusual drawing primitives can be a major impediment to this conversion.

Extensibility: there must be a well defined facility for efficiently adding primitives or attributes to the language. It must be possible to parse extensions to the language without necessarily interpreting the extension.

User readability: users must be able to understand and edit the language. This is valuable for software debugging and for making manual adjustments to pictures.

Intuitive interpretation: there must be a direct correspondence between intuitive concepts and language constructs. Unusual constructs lead to bugs, misunderstandings, and lengthy documentation.

Density: pictures will be archived for long periods of time in this language. The most common constructs should be expressed with as few wasted characters as possible. The judicious use of macros and abreviations is very helpful in this respect.

Composition: it must be convenient to create compound objects from more primitive objects, and to manipulate these compound objects.

The most important shortcomings of Fig code and PostScript can be understood in terms of these goals. For example, Fig is unreadable and not conveniently extensible; PostScript suffers from high compuational overhead due to high language complexity and low density due a verbose style of punctuation.

ApGraph

Experience with Fig code has demonstrated the usefulness of an application level graphics description language to the TEX community. In spite the shortcomings of Fig code, the use of Fig and TransFig is increasing along with the popularity of the applications which use it. This is the appropriate time to stop and redesign TransFig; code based on preexisting designs and minimal resources is not a sound basis for future development.

At Cornell, Alex Siegel is developing ApGraph, an intermediate language for *Ap*plication *Graph*ics, which is intended to replace Fig code eventually. A new graphics editor based on the X Windowing System will support ApGraph as well as TFX, and

TransFig will continue to support both. Further development based on Fig code will, however, cease.

The TEX community has much to gain by adopting some application level graphics language as a standard. We hope that ApGraph will be an attractive option, as it is a simple language oriented towards the technical graphics most needed by the TEX community. Of course, other standards are possible, including the ANSI/ISO standard Computer Graphics Metafile (CGM) format. The best choice of language requires further study and consideration.

Conclusions

We have defined a set of goals for an application level intermediate form for technical graphics in TEX documents. We have seen how the definition of a standard interpretation for Fig code has allowed it to serve the function of an intermediate graphics language. The TransFig package, which implements this standard interpretation of Fig code, is gaining increasing popularity and acceptance in the TEX community, in spite of Fig code's serious shortcomings as a graphics language. We have argued the value to the TEX community of adopting a standard language for application level description of technical graphics, and outlined some design requirements of such a language. ApGraph is being developed as Cornell in the hopes of influencing the definition of such a standard.

Software Availability

Most of the software described in this article is available without charge from the archive server at Clarkson University (Internet: `sun.soe.clarkson.edu`). Access is through anonymous FTP or by mail. Many packages, including the most recent version of TransFig, are also avaiable for FTP from Cornell University (Internet: `svax.cs.cornell.edu`). FTP sites for packages not available from Clarkson are listed below. This information is subject to change.

GnuPlot is available from `duke.cs.duke.edu`.

Pic2fig is not available for anonymous FTP. Contact author Micah Beck for distribution.

Plot2fig is available from `qed.rice.edu`.

Xfig is available from `expo.lcs.mit.edu` as a contributed client.

Acknowledgements

The development of TransFig has been made possible by the collaboration of a group of individuals on the network too numerous to list here. The most important individual contributions were made by Frank Schmuck and Conrad Kwok. Ajei Gopal first introduced Micah Beck to Fig, and Alex Aiken patiently tested the Fig-to-PICTEX translator while writing his thesis. A special acknowledgement goes to Supoj Sutanthavibul, whose original implementation of Fig has lived on in many incarnations and from which all versions of Fig and TransFig are derived.

Bibliography

Adobe Systems Incorporated. *PostScript Language Reference Manual.* Reading, Mass.: Addison-Wesley, 1985.

Beck, Micah. "TransFig: Portable Figures for TEX" Cornell University Dept. of Computer Science Technical Report #89-967, (February 1989)

Kernighan, B. W. "PIC—A Language for Typesetting Graphics", *Software Practice and Experience*, 12(1), pp. 1-21 (January 1982)

Kwok, Conrad. "Extensions to EPIC and LATEX Picture Environment." Software documentation. University of California, Davis, Dept. of Computer Science. (July 1988)

Lamport, Leslie. *LATEX: A Document Preparation System.* Reading, Mass.: Addison-Wesley, 1986.

Podar, Sunil. "Enhancements to the Picture Environment of LATEX." State University of New York at Stony Brook, Dept. of Computer Science Technical Report #86-17 (July 1986)

Wichura, Michael. "The PICTEX Manual." Software Documentation. The University of Chicago. (November 1986) Second printing, by The TEX Users Group, as *TEXniques*, Number 6, 1987.

Appendix

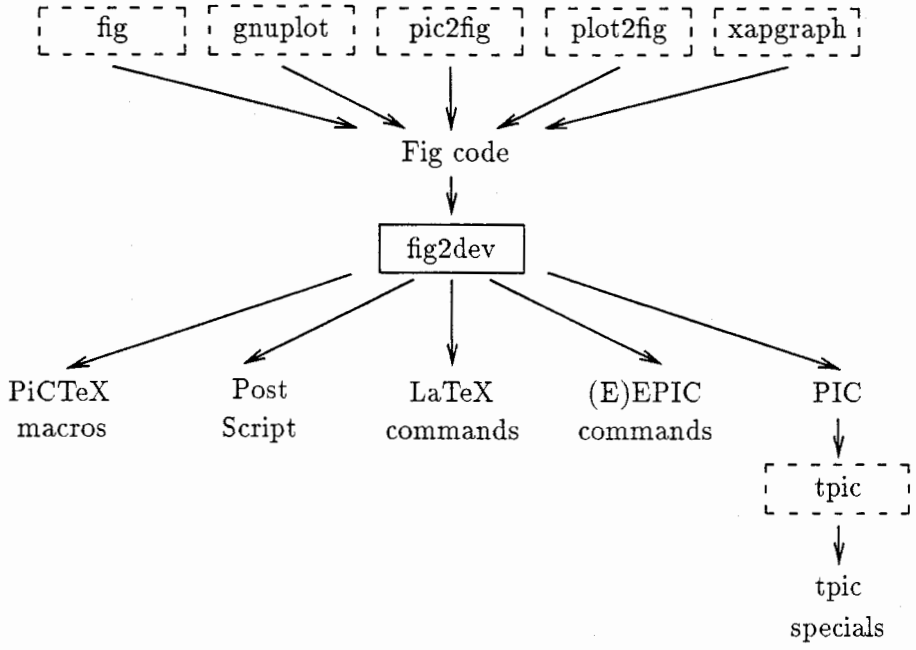
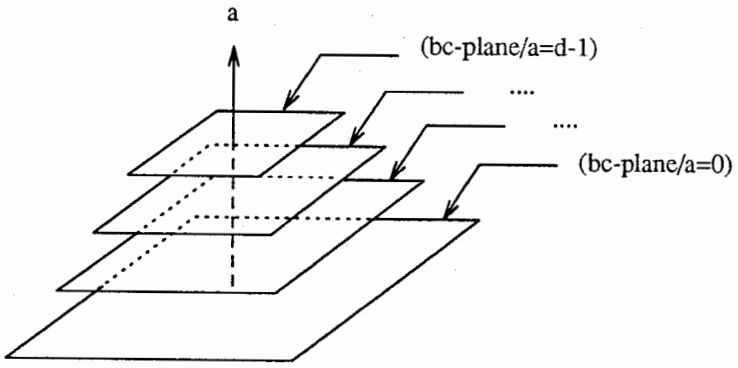
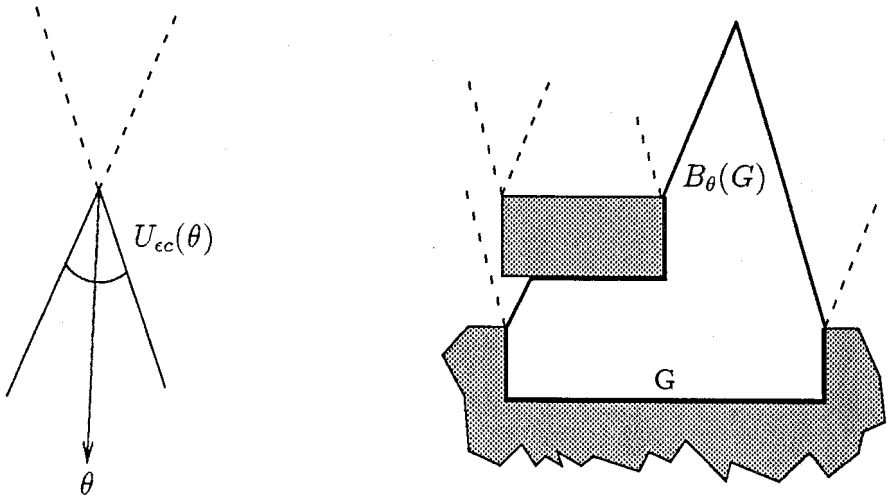
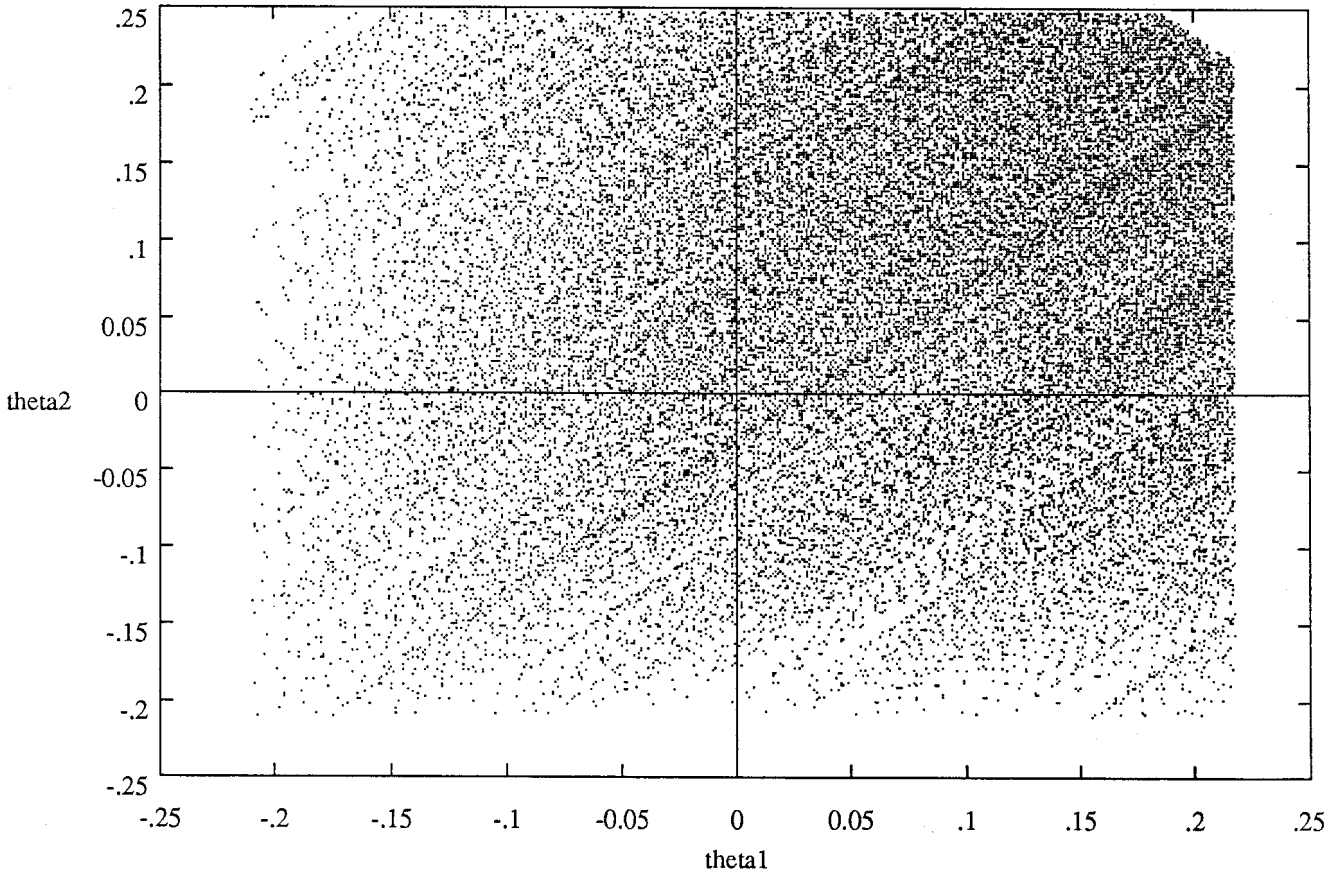


fig
gnuplot
pic2fig
plot2fig
xapgraph
Fig code
fig2dev
PiCTeX
macros
Post
Script
LaTeX
commands
(E)EPIC
commands
PIC
tpic
tpic
specials


a
(bc-plane/a=d-1)
....
....
(bc-plane/a=0)


$U_{cc}(\theta)$
θ
$B_\theta(G)$
G


.25
.2
.15
.1
0.05
0
-0.05
-.1
-.15
-.2
-.25
theta2
-.25
-.2
-.15
-.1
-0.05
0
0.05
.1
.15
.2
.25
theta1