?? "Contents"

The geom style for LaTeX

Silvio Levy
Geometry Center
University of Minnesota

July 1992

Contents

Bug reports and comments about the software and documentation can be sent to `levy@geom.umn.edu`. I will take them into consideration in future releases, but there is no guarantee that particular bugs will be fixed.

Section "Introduction"
geom
PostScript
historical notes
acknowledgements
Bill Thurston
Three-Dimensional Geometry and Topology
Yair Minsky
David Epstein
Warwick University
Don Knuth
Leslie Lamport
Rainer Schöpf
Frank Mittelbach
Trevor Darrell
Section "Overall Organization"
new font selection scheme
% Installation

1. Introduction

This document describes a LaTeX style called `geom`, which is loosely based on Lamport's `book` and `article` styles [Lam86], but provides a number of additional features, such as:

- inclusion of PostScript figures in the document, and of TeX text within figures;
- automatic creation of index entries and cross-reference labels where appropriate;
- no restriction on the use of macros and special characters in titles, cross-references, captions, etc.;
- greater versatility in defining theorem-like environments;
- upright digits in an italics environment (optionally); and
- proofing aids such as version numbers and a running index.

There are also purely esthetic differences, many of which can be controlled by the user.

This style had its inception in a set of LaTeX macros written for the formatting of Bill Thurston's *Three-Dimensional Geometry and Topology*, first by Yair Minsky and subsequently by me. After three years of gestation, the macros started being used intensively by some other people—notably Thurston, David Epstein and a number of students at Warwick University—and more modifications, including a complete rewrite, were made. The first book published using the `geom` style [ECL$^+$92] came out in February 1992.

The Appendix to this document describes `mathfig`, a system for typesetting TeX labels in Mathematica figures. (Mathematica is described in [Wol91]). This system was designed primarily for use with `geom`, but can also be used independently.

This release owes much to the patience of these first users and the feedback they provided, for which I am very grateful. Naturally, my debt to the people on whose work I've built—Don Knuth, Leslie Lamport, Rainer Schöpf, Frank Mittelbach, Trevor Darrell and others—is impossible to quantify. I wish to take this opportunity to thank them. I also wish to thank Al Marden and all my colleagues at the Geometry Center for making the Center the friendly place that it is, and for their generous help.

2. Overall Organization

In order to use this style, you need to be running a version of LaTeX that supports the so-called new font selection scheme (NFSS), due to Frank Mittelbach and Rainer Schöpf. To check whether your LaTeX supports NFSS, start LaTeX and type `\show\selectfont`. If the response includes

```
> \selectfont=undefined.
```

this is the wrong LaTeX. Don't despair yet; the right LaTeX may be available at your site, but under a different name—try `amslatex`, or ask your local guru. (Strictly speaking, AMS-LaTeX [Ame] is a set of LaTeX style files that emulate AMS-TeX. Since their use requires NFSS and the necessary modifications are supplied with the AMS-LaTeX style files, the name is sometimes used for LaTeX+NFSS.) If you can't find LaTeX+NFSS at all, see further instructions in Section 3.

calling geom
Section "Installation"
% Overall Organization

Unlike most other LaTeX styles, `geom` comes in several files, devoted to different capabilities such as cross-referencing, figures, indexing and so on. The files currently in the distribution are:

`dvips.chg`	driver dependencies	`geombook.sty`	changes to `book.sty`
`geom.sty`	master file	`geomenv.sty`	environments
`geomar10.sty`	changes to `art10.sty`	`geomeqns.sty`	option file
`geomar11.sty`	changes to `art11.sty`	`geomfig.sty`	high-level figure support
`geomar12.sty`	changes to `art12.sty`	`geomfnt.sty`	fonts and special symbols
`geomart.sty`	changes to `article.sty`	`geomindx.sty`	index and glossary
`geombk10.sty`	changes to `book10.sty`	`geompsfi.sty`	PostScript figure support
`geombk11.sty`	changes to `book11.sty`	`multicol.doc`	source for `multicol.sty`
`geombk12.sty`	changes to `book12.sty`	`multicol.sty`	two-column formatting

The file `geom.sty` is the master, and it is read in when `geom` is called as an option to a main document style, which can be either `book` or `article`. That is, to use the stuff described in this document, your text file should start with a line such as

```
\documentstyle[12pt,geom,...]{book}
```

When `geom.sty` is loaded in, it guesses what your main style is, and reads in some changes to that style, contained in the files `geombook.sty`, `geombk12.sty`, and so on.

The file `multicol.sty` is part of the package files developed by Frank Mittelbach and Rainer Schöpf at the University of Mainz. It is accompanied by `multicol.doc`, which is a "source" file with documentation. To obtain more of the excellent software written by Mittelbach and Schöpf, use anonymous ftp to `ymir.claremont.edu`, and cd to `tex.inputs.latex-mainz`.

The rest of this document describes the installation of `geom` (skip if it's already installed), its major features, and how you can take advantage of them and (sometimes) modify them. Some material that is either very technical or unlikely to be needed except under unusual circumstances is printer in smaller type and introduced by an asterisk $*$.

3. Installation

You only need to read this section if you are in charge of installing `geom` at your site. Installation is very simple if your LaTeX supports NFSS and your TeX-to-PostScript driver is Tom Rokicki's `dvips` [Rok].

To check whether you have LaTeX+NFSS, see the beginning of Section 2. To install LaTeX+NFSS, get it by ftp from `e-math.ams.com` (it is currently in `ams/amslatex.tar.Z`), and follow the installation instructions in the file `amslatex/fontsel/nfssinst.tex`.

If your driver is not `dvips`, you have two choices: either install `dvips`, which is almost certainly a good idea (get it by ftp from `labrea.stanford.edu`, in the directory `pub`); or make a copy of the file `dvips.chg` in this distribution, and modify it so it writes the `\special` commands required by your driver. If you adapt `dvips.chg` for your driver, give the new file a different name, and replace with this name all occurrences of `dvips.chg` in the rest of this section. Also, please send me a copy of the file, so I can include it in future distributions.

Section "No More Fragile Commands"

As I was saying before I got sidetracked, installation is simple. If you're on a UNIX system, edit the `Makefile` to make sure the pathnames are correct for your site, and run `make install`. If you're not on a UNIX system, here are the step-by-step instructions:

Place the files listed in the previous section (all the distribution files that end in `.sty`, `.chg` or `.doc`) in the system-wide directory for TEX input files. Create a symbolic link `driver.chg` in this directory, pointing to `dvips.chg`. (If your system does not support symbolic links, copy `dvips.chg` to `driver.chg`.)

Copy the file `mathlabels.orig` to `mathlabels` and edit its first line to be the correct pathname of the perl executable on your system. (If you don't have `perl` at all, you don't know what you're missing—get it as soon as possible.) Decide on a directory where you want to install the auxiliary scripts `mathlabels` and `geomfix` (which are used by `mathfig`), and put copies of them there. Copy the file `mathfig.orig` to `mathfig` and edit the lines at the top to specify the location of the `mathlabels` and `geomfix` scripts, then copy `mathlabels` to a system-wide directory for executables.

Finally, copy `math.pro` and `illustrator.pro` into the directory where your TEX-to-PostScript driver looks for prolog files. (This assumes your driver supports the inclusion of prolog files; switch to `dvips` if it doesn't.)

4. No More Fragile Commands

In standard LATEX, commands such as `\section` have *moving arguments*, that is, arguments that migrate to an auxiliary file *jobname*`.aux` to be used later in typesetting the table of contents and so on [Lam86, 33–34]. In this process, macros in the body of the argument are expanded, so that when you type, say,

```
\section{Annoying \TeX nicalities}
```

what goes into the auxiliary file is something like

```
\@writefile{toc}{\string\contentsline\space {chapter}{\string\numberline\space
{1}Annoying T\kern -.1667em\lower .5ex\hbox {E}\kern -.125emXnicalities}{1}}
```

where the second line contains the expansion of the macro `\TeX`.

There are several problems with this. First, the expansion, and consequently the line where it occurs in the auxiliary file, can be very long, which in some implementations means that TEX cannot read the line back in. Secondly, the expansion might not make sense, or might do the wrong thing, at the time when it is read in again. Even worse, the expansion of `\section{\small foo}`, for example, actually causes an infinite loop.

* Experienced users of TEX who want to know why this happens can try typing the following lines to plain TEX:

```
\def\foo{\let\bar\foo}
\foo                    %Nothing much happens
\immediate\write0{\foo} %Disaster strikes
```

The problem is that at the time of the `\write` no assignment is taking place, so the `\foo` at the end of the expansion is not handled as a literal.

% Theorems and Their Friends
% PostScript Figures
Section "Labels and Cross-References"
% Theorems and Their Friends

Standard LaTeX's solution to these problems is for the user to precede "fragile" macros with the `\protect` command [Lam86, 33–34]. By contrast, in the `geom` style, you don't have to worry about "fragile" commands in the following situations:

- in the arguments of the sectioning commands `\chapter`, `\section`, and so on;
- in the optional argument to the `theorem` environment and its relatives (Section 6);
- in the arguments to `\caption`, `\fig` and its relatives (Section 10);
- in the arguments to `\markright` and `\markboth`;
- in the argument to `\cite`.

This covers almost all important situations where fragile commands need to be `\protect`ed [Lam86, 151]. In future versions I hope to extend this permissiveness to the remaining situations.

Notice, however, that the `\verb|...|` construction continues to be forbidden inside arguments to most commands. This is due to reasons deeply embedded in TeX itself, and is unlikely ever to change.

* In some circumstances you might want the old LaTeX behavior of macro expansion inside the argument of `\chapter`, say. For instance, you might have a locally defined macro whose definition will be unknown at the time the table of contents is read. In this case you can use the following construction:

```
\edef\mystring{...} % where ... stands for the argument to be expanded
\expandafter\chapter\expandafter{\mystring}
```

5. Labels and Cross-References

In standard LaTeX, the argument to the `\label` command cannot contain anything but characters (after expansion). In the `geom` style, this restriction is lifted, so labels contain just about anything. For example, `\label{\TeX}` and `\label{Isometries of $\bold R^2$}` are acceptable. Macros are no longer expanded.

One advantage of this flexibility is that subdivisions can be automatically labeled by their titles. In the `geom` format, by default, every sectioning command such as `\section` generates a label with its title. If this feature causes you to run out of memory, you can turn it off by saying `\autolabelfalse` in your document's preamble.

To refer to the place where a label occurs, you can still use the `\ref` command, as in standard LaTeX. This gives a bald number—for example, `\ref{Isometries of $\bold R^2$}` will print "8.3" if Section 8.3 starts with `\section{Isometries of $\bold R^2$}` while automatic labeling is in effect.

A new feature is that you can replace `\ref` by `\fullref` or `\Fullref`, to get "section 8.3" or "Section 8.3", respectively. The way `\fullref` works is by storing in a certain register the type of environment in which a label was created. Thus, when a new section starts, this register gets the value "section". When a `\begin` command introducing one of the environments described in Section 6 is encountered inside this section, the register gets updated to the type of that environment, say "theorem". It reverts to "section" when the matching `\end` command is read.

Theorem "Erdös"
Kurdish
% Erdös
% circle
% Kurdish
Erdös
% circle
% circle
Section "Theorems and Their Friends"

Even if you have nested environments, labels placed in each of them are assigned the correct type. Consider the input

```
\newtheorem[{\par\large}{\it}]{theorem}{Theorem}[section]
\equationwith{theorem}                    % These two lines are explained
...                                       % in later sections
\begin{theorem}[Erd\"os]
A theorem both deep and profound / Is that a circle is round:
\begin{equation}
\mathrm{curvature}\,S^1=\mathrm {const}.
\label{circle}
\end{equation}
In a paper by Erd\"os / Written in Kurdish / A counterexample is found.
\label{Kurdish}
\end{theorem}
```

which gives

Theorem 5.1 (Erdös). *A theorem both deep and profound / Is that a circle is round:*

$$\mathrm{curvature}\,S^1=\mathrm{const}. \tag{5.2}$$

In a paper by Erdös / Written in Kurdish / A counterexample is found.

Now `\Fullref{Erd\"os}` gives "Theorem 5.1", `\Fullref{circle}` gives "Equation 5.2", and `\Fullref{Kurdish}` gives "Theorem 5.1". Furthermore, `\ref*{Erd\"os}` gives "5.1 (Erdös)", that is, it prints the (optional) tag associated with theorems and the like. Of course you can also use `\fullref*` and `\Fullref*`.

∗ If you disagree with LaTeX's idea of what the label type should be at a particular point, you can override is using `\setlabeltype`. For example, if we had `\setlabeltype{formula}` right before the `\label` command in the previous example, the result of `\Fullref{circle}` would be "Formula 5.2" instead of "Equation 5.2".

Before `\ref` actually prints the number of a cross-reference, it applies the macro `\preref` to it. By default, this macro does nothing; but you can redefine it to do whatever you want with the number. If you want the number to appear upright even if the surrounding text is in italics, you can say

```
\def\preref#1{{\normalshape#1}}
```

The output of `\pageref` and of `\cite` is also passed to `\pageref` before being printed.

6. Theorems and Their Friends

Standard LaTeX provides the `\newtheorem` command to create environments for statements like theorems, lemmas, and so on. The basic idea can be adapted for other types of text that should stand out, such an exercises, conventions, and so on. Not all of these should be treated typographically the same: for example, you might want exercises in smaller type.

% Labels and Cross-References
% Labels and Cross-References
% Erdös

The `geom` style extends LaTeX's `\newtheorem` command to allow you to do this easily. Recall from [Lam86, 58] that `\newtheorem` has two required arguments: the name of the new environment (say "theorem"), and the text used to introduce it (say "Theorem"). In standard LaTeX it also takes an optional argument, whose meaning depends on whether it comes before or after the second required argument.

In the `geom` style, you can use yet another optional argument, right after `\newtheorem`, to tell LaTeX how this environment differs from a vanilla theorem-like environment. This optional argument should consist of two sequences of commands, each surrounded by braces, like this:

```
\newtheorem[{\par\large}{\it}]{theorem}{Theorem}[section]
```

(This is the same definition used for the example in Section 5.)

The first sequence of commands in braces, `\par\large`, will be executed as soon as LaTeX sees a `\begin{theorem}`. This means the theorem will come out in larger type (see the output in Section 5). LaTeX implicitly starts a new level of braces when it encounters a `\begin` command—changes inside the environment are local, so the `\large` will cease to have effect at the end of the environment.

The second sequence of commands is inserted right after the introductory text **Theorem 5.1 (Erdös)**. It affects everything from that point till the end of the environment. Here we've used `\it`, so the statement of the theorem is printed in italics. Note that, unlike the standard `article` and `book` styles, `geom` does not italicize theorems by default. If we didn't have the `\it` in the `\newtheorem` command, the text of the theorem would be upright.

* You can also control the appearance of the introductory text itself by changing the definition of `\theoremintro`, which by default is

```
\def\theoremintro#1{\normalshape\bf#1}
```

The argument `#1` stands for the introductory text.

You need two pairs of braces in the new optional argument to `\newtheorem`, even if one is empty. If both are empty, the behavior is the same as if the optional argument weren't there.

Another enhancement to `\newtheorem` in the `geom` style is automatic labeling. Suppose you say

```
\newtheorem{definition}[theorem]{Definition}
```

(recall from [Lam86, 59] that the use of `[theorem]` causes definitions to be numbered in the same sequence as theorems, assuming the `theorem` environment has been previously defined). Now saying

```
\begin{definition}[universe]
The universe is a Hubble bubble.
\end{definition}
```

prints

Def. "universe"
% Labels and Cross-References
% universe
% Labels and Cross-References
% universe
% Labels and Cross-References
Section "The Proof Environment"

Definition 6.1 (universe). The universe is a Hubble bubble.

and implicitly calls `\label{universe}`. (As mentioned in Section 5, you can turn this feature off by saying `\autolabelfalse`.) The environment knows its type at the time the label is made, so saying `\Fullref{universe}` will print "Definition 6.1".

* The type is usually just the environment name, but it can be overridden by including `\setlabeltype` (Section 5) in the first optional argument to `\newtheorem`. Thus if the `definition` environment had been created with

```
\newtheorem[{\setlabeltype{def.}}{}]{definition}[theorem]{Definition}
```

`\Fullref{universe}` would appear as "Def. 6.1".

The preferred style of some publishing houses calls for digits and punctuation to be typeset upright in theorems and such, amidst surrounding text in italics. The control sequence `\specialdigits` achieves this. It causes each digit in text, as well as parentheses, commas, colon, semicolons and periods, to effectively be preceded by `\normalshape`. You can use `\specialdigits` after `\it` in the definition of a new environment with `\newtheorem`.

* The way `\specialdigits` works is by making digits and punctuation into active characters (changing their `\catcode`s to 13). This has two consequences. First, characters that arise from macro expansion don't get the special treatment, because their `\catcode`s are frozen beforehand; this is true, in particular, of the expansion of `\ref` commands (but see the end of Section 5).

The second consequence is that the digits and decimal point no longer work in specifying dimensions: if you write `\vskip 1in` while `\specialdigits` is in effect, you get an error message complaining that a number was expected. To get around this, you can temporarily cancel the specialness with `\regulardigits`. If you run into this problem more than once or twice, it's best to create a macro, outside the scope of `\specialdigits`, that expands to the desired command (`\def\inch{\vskip 1in}`).

7. The Proof Environment

Theorems and proofs go hand in hand. The `proof` environment in `geom`, in its most basic form, prints *Proof.* at the beginning of the proof, and an end-of-proof symbol □ at the end. With an optional argument it behaves like this:

```
\begin{proof}[Poincar\'e conjecture]
Notice first that ...
... as we wished to show.
\end{proof}
```

Proof of Poincaré conjecture. Notice first that ... as we wished to show.

Poincaré conjecture

* The formatting of the proof's introductory text is controlled by the macro `\proofintro`, whose single argument is the optional argument to `\begin{proof}`—in this case, "Poincaré conjecture". If the optional argument is not used, `\proofintro` is passed an empty argument. The default setting of `\proofintro` is

Section "Equations"
% Labels and
Cross-References

```
\def\proofintro#1{\def\@tempa{#1}%
  {\it Proof\ifx\@tempa\empty\else\ of #1\fi. }\ignorespaces}
```

You can probably figure out how it decides whether to write *Proof* or *Proof of*.... You can change its definition to obtain other effects.

Likewise, the formatting of the box at the end of the proof is controlled by the macro `\provedboxcontents`, which is entirely analogous. Its standard setting is

```
\def\provedboxcontents#1{\def\@tempa{#1}%
  \ifx\@tempa\empty$\square$\else\fbox{\small#1}\fi}
```

To identify your proofs with the number of their corresponding theorems, lemmas, corollaries, etc., use `\setprooftag` in the definition of these environments. For instance, after you say

```
\newtheorem[{\setprooftag}{\it}]{theorem}{Theorem}[section]
\newtheorem[{\setprooftag}{\it}]{corollary}[theorem]{Corollary}
```

`\begin{proof}` will behave as if you were passing it as its optional argument the number of the most recent theorem or corollary. Nested proofs keep track of their arguments correctly, so the box at the end of each gets the same tag as the *Proof of*... at the beginning.
However, certain situations are too confusing for `\setprooftag`. For example, if you state a corollary between a theorem and its proof, the proof will incorrectly get the corollary's number, which is more recent. In this situation you have to use the optional argument. Of course, it's best to use `\ref{...}` as the argument, rather than an explicit number, which might change later.

If a proof ends with a displayed formula, it is recommented that the proved box be placed next to the formula (as if it were a tag). This is done by inserting `\proved` before the closing `$$`. Likewise, if the last line of the proof is part of an `\item`, you can use `\proved` before the closing `\end{enumerate}` (or whatever) so the proved box won't be set on a line by itself. In even more difficult cases—for instance, a formula that already has a tag, or is part of a multi-line display—plop `\box\provedbox` where you want the box to be.

8. Equations

The `geom` style reinstates the plain TEX macro `\eqalign`, which is not part of standard LATEX but is nonetheless very useful. Unlike the `eqnarray` environment, `\eqalign` creates math alignments that are not treated as whole lines, and so can be combined with other elements.

The example in Section 5 has `\equationwith{theorem}` after the `theorem` environment is created. This causes equations made with the `equation` and the `eqnarray` environments to be numbered in the same sequence as theorems. You can replace `theorem` with any environment that was defined with `\newtheorem` and that has a counter with its name (the environment has a counter with its name if there was no optional argument between the first and second required arguments to `\newtheorem`).

If you load the optional style `geomeqns`—that is, if your document starts with

```
\documentstyle[geom,geomeqns,...
```

Section "The Figure and Table environments"
% Proofing Aids

the tags in the `equation` and the `eqnarray` environments are printed on the left, in boldface. This makes them visually similar to the tags that introduce theorems, so it is generally appropriate to use this style if you're numbering equations and theorems in the same sequence.

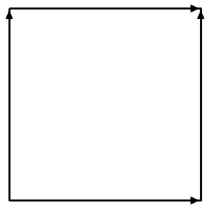
9. The Figure and Table environments

The `geom` style supports the `figure` and `table` environments of standard LaTeX, and, as mentioned above, if you use the `\caption` command you don't have to worry about fragile macros inside its argument. However, I recommend that you use instead the new `Figure` and `Table` environments, which provide some bells and whistles.

These environments take three arguments: a cross-reference label, a title and a legend (or explanatory caption). The command

```
\begin{Figure}{torus}{The square torus}
{A torus can be obtained, topologically,
by gluing together parallel sides of a square.}
$$
\begin{picture}(1,1)
...     % lots of \put commands
\end{picture}
$$
\end{Figure}
```

prints



torus

Figure 1. The square torus. A torus can be obtained, topologically, by gluing together parallel sides of a square.

and in addition it creates a cross-reference label `torus` and an entry in the table of contents with the text "The square torus". As you can see, the figure number and the title appear in boldface, the legend in a lighter face. In proofing mode (Section 13), the file name also appears under the figure, in tiny letters. If you don't want a legend, just write `{}` after the first two arguments. (In other words, the legend is *not* an optional argument—it is simply an argument that sometimes happens to be empty.) There is an optional argument to the `\Figure` environment, consisting of a subset of the letters `htbp` and indicating a location where the figure may be placed [Lam86, 176]; it should come before all other arguments. All of this applies to the `Table` environment as well.

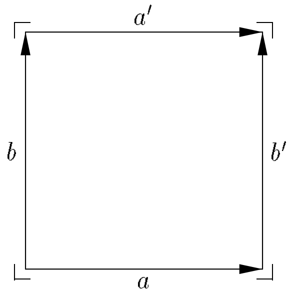
Section “PostScript Figures”
Psfig/TEX
Psfig/TEX
Stephan Bechtolsheim
% PStorus

10. PostScript Figures

Most often your picture won't be made with LATEX commands; it will exist in a separate PostScript file. Inclusion of PostScript figures is still a bit of a bugaboo for many TEX users, in spite of the existence for many years now of Trevor Darrell's excellent macro package Psfig/TEX [Dar] and others.

The `geom` style automatically loads the file `geompsfi.sty`, a modified version of Psfig/TEX (incorporating some improvements by Stephan Bechtolsheim and others of my own). It allows you to include PostScript figures very simply:

```
\fig{PStorus}{The square torus}{A torus can be obtained, topologically,
by gluing together parallel sides of a square.}
```



PStorus

Figure 2. The square torus. A torus can be obtained, topologically, by gluing together parallel sides of a square.

The `\fig` macro takes the same three arguments (plus an optional one) as the `Figure` environment. But the first argument, in addition to being used to make a label, is also the name of the file where the PostScript figure is stored (minus the `.ps` suffix). Thus Figure 2 is stored in the file `PStorus.ps`.

To get two or three figures side by side, sharing the same caption, use `\doublefig` or `\triplefig`. The PostScript file names for `\doublefig{foo}...` would be in this case would be `foo-1.ps` and `foo-2.ps`, and likewise for `\triplefig`. There is no similar mechanism for more than three figures.

The directory where the figures are to be found is governed by the macro `\picpath`. For example, if the directory is called `/u/levy/pictures`, you should say

```
\def\picpath{/u/levy/pictures/}
```

at the beginning of your document. (You need the `/` at the end.)

∗ You can also use a relative pathname, such as `pictures/`, or even the empty string (or `./`) for the current directory. However, the program that sends your `.dvi` file to the printer may not find the pictures since it is likely to be invoked from a different directory.

If a pathname includes characters that are special to TEX, you need to make then unspecial in order to define `\picpath`. Here is an example:

```
\begingroup     % start a group so the change in _ is local
\catcode`\_=12  % make _ behave like an ordinary character
\gdef\picpath{/home/tampa_a/fac/wpt/pictures/} % this def is global
\endgroup       % now _ reverts to its usual meaning
```

% torus
% PostScript Figures
% PostScript Figures
% PostScript Figures
% PStorus
% Proofing Aids
% PStorus
% PStorus

If you have a long document with lots of pictures, it is better to organize them into sections and chapters. In that case you should leave `\picpath` unset, and set the macro `\masterpicpath` to the master picture directory. If I were using this scheme in this document, I might say

```
\def\masterpicpath{/home/levy/pictures/}
```

and put Figure 1 in file `/home/levy/pictures/10/PStorus.ps`. (With the `book` style, which has chapters, I would need `chap3/10` instead of just `10`.)

The way `\fig` works is by calling the command `\psfig` [Dar], which creates a TEX `\vbox` of the right size and containing instructions for the figure to be included at the time of printing. You normally won't use `\psfig` directly, since `\fig` calls it automaticaly.

∗ However, occasionally you many want to use `\psfig` directly to take advantage of this macro's many options, such as resizing. See the Psfig/TEX manual [Dar] for details; but some of the newer options to Psfig/TEX are not supported here.

In order for the `\psfig` command to know what size box to make, it needs to find your PostScript figure file, and the file must have near the top a line like this:

```
%%BoundingBox: 0 30 98 128
```

The first two numbers represent the low x- and y-coordinates of the figure's bounding box, in points, and the last two represent the high x- and y-coordinates. In the example just given, which corresponds to Figure 2, the bounding box goes from 0 to 98 pt in the x-direction and from 30 to 128 pt in the y-direction, so its height and width are both 98 pt, or approximately 1.35 inches. The corners of the bounding box appear on the page in proof mode (Section 13).

Note that `\psfig` only cares about the height and the width of the bounding box, not about the actual values of the high and low coordinates. The latter are relevant only in reference to the coordinates of the objects inside the picture, about which `\psfig` knows nothing. (Thus the bottom left corner of the square in Figure 2 has coordinates $(4, 34)$, which is a little bit above and to the right of the bounding box's bottom left corner $(0, 30)$.) If one added the same number to the x-coordinates of the bounding box and of every object in the picture, the result would be exactly the same.

Normally, the `%%BoundingBox` line should have been placed in the PostScript file automatically, by whatever program you used to create the figure. If the program did not write a line like that, you have to do it by hand, and you should consider switching to another program.

Notice that the labels a and b in Figure 2 appear in the same font as they would in the text. In fact the labels were typeset by TEX itself, and do not come from the PostScript file, where it would be hard to specify them satisfactorily—especially details such as the placement of the primes and other subscripts and superscripts. The information for these labels is contained in a file with the suffix `.lab`, which normally is generated by the same program that creates the PostScript file, or by a postprocessor. If you want to include *Mathematica* graphics, for example, you should use the `mathfig` utility, described in the Appendix.

% PStorus

∗ The average user has no need to know what the `.lab` file looks like, but if you want to write a new postprocessor in order to implement this labeling scheme in conjunction with your favorite drawing program, here are the details. Each line of the file yields one label, and contains the `\setlabel` command followed by five arguments: the text of the label, to be typeset in math mode; the x- and y-coordinates of the label's *reference point*, in points; and the *relative* x- and y-coordinates of the reference point. A value of 1 for the last two arguments means the reference point of the label it at the top or right of the label's bounding box; a value of -1 means bottom or left, a value of 0 means middle, and so on.

For example, here is the file `PStorus.lab` corresponding to Figure 2:

```
\setlabel{a}{48.600000}{34.162104}{0}{1}
\setlabel{b}{4.162104}{78.600000}{1}{0}
\setlabel{a'}{48.600000}{123.037896}{0}{-1}
\setlabel{b'}{93.037896}{78.600000}{-1}{0}
```

For the label a, the reference point is at the top middle of the label's bounding box, so the label itself will be centered under the point with coordinates $(48.6, 34.1)$.

If you want to typeset a label outside math mode, use LaTeX's `\mbox` command (or TeX's `\hbox`). The font used by default in this case is the same as for captions. Thus, to get the label `foo` you'll probably want to type the string `\hbox{foo}` into your figure-making program.

Sometimes the fact that the source `\hbox{foo}` has many more characters than the output causes problems—for example, the figure-making program might leave too much space for the label. The macro `\hyperactivelabels` provides a way around this. This macro is called every time a label is about to be typeset, and you can define it at will, to allow abbreviations, say. Thus, after the commands

```
\def\hyperactivelabels{\mathcode`:="8000}
\def\activecolon#1:{\hbox{#1}}
{\catcode`\:=\active\global\let:=\activecolon}
```

the construction `:foo:` has exactly the same effect as `\hbox{foo}`—but only inside labels. (The command ``\mathcode`:="8000`` causes the colon to be treated as an active character in math mode; the next two lines say, in a roundabout way, what this active character should do.)

Utilities (programs) that generate PostScript graphics files generally start the file with a "signature" (a comment line stating the utility's name and version) and a chunk of PostScript code—call it a header—establishing abbreviations, subroutines, fonts and the like. The header is always the same, so if you have dozens of figures generated by the same utility it's good to factor out this commonality, if possible. With certain utilities, you can save your files without the header. Then they won't print on their own; but when `\psfig` looks into them, if it can recognize the utility's signature, it puts into the dvi file a command that causes the inclusion of the header at printing time. This way the header is included only once for all figures, saving memory and time. The utilities that `\psfig` currently knows about are Mathematica [Wol91] and Adobe Illustrator [S$^+$88]. With Illustrator, saving without the header is accomplished by clicking on the appropriate entry in the "Save As" dialog box. For Mathematica, see the Appendix.

∗ For the signature to be seen by `\psfig`, it must appear near the top of the file; the actual number of lines that `\psfig` scans is controlled by the counter `\maxheaderlines`. The `%%BoundingBox` line must also lie in that same initial stretch. The default value of `\maxheaderlines` is 100, which is almost always adequate.

Section “Fonts”
% Overall Organization
Section “The Index, the Table of Contents and the Glossary”
% Labels and Cross-References
% Theorems and Their Friends
% Proofing Aids

11. Fonts

As mentioned in Section 2, the `geom` style requires NFSS, a rational font management system that makes the use of new fonts easier than under traditional LaTeX.

Under this scheme, unlike the old, if you say `\bf...` `\small...`, the `\small` has no effect on the boldness of the current font—size, boldness, style and so on are independent attributes. This is generally good, but it does create backward incompatibilities. For example, if you have an old LaTeX document where you've relied on `\small` to reset the weight of the font, you're in for a surprise. Also, `\rm` should generally be replaced by `\normalshape`. For a list of all such differences, see [Ame].

Another case of conflicting ideas on what font-change commands should do is inside math mode. Under NFSS, `\bf` has no effect inside math mode—you need to say `$\bold{xyz}$`, for example. The `geom` style will print a warning on your terminal if it sees `\bf` in math mode.

In addition to the math-mode commands `\bold`, `\mathrm` and `\mathit`, the `geom` style defines `\sans` and `\type` as the math mode counterparts of `\sf` (sans-serif) and `\tt` (typewriter type).

Support for Fraktur ("gothic") and blackboard-bold fonts is provided, as well as for the fonts `msam` and `msbm`, which contain a plethora of mathematical symbols. This is all thanks to the AMS, and you should turn to the AMS-LaTeX distribution [Ame] if your site lacks the fonts themselves or the prerequisite auxiliary files `amsfonts.sty` and `amssymb.sty`.

12. The Index, the Table of Contents and the Glossary

The basic command for putting something into the index is `\index`, as in standard LaTeX. But `\index` does not print its argument. To index a word that actually appears in the text say `\ix{...}`. To index mathematical notation say `\ixn$...$` or `\ixninv$...$`, depending on whether the contents should appear or not. And, just for symmetry, you can also use `\ixinv`, which equals `\index`:

	Text	Notation
Visible	`\ix`	`\ixn`
Invisible	`\ixinv`	`\ixninv`

An index entry is made automatically for any `\label` or `\ref` command. In the second case the entry goes into the index file preceded by a `%`.

Remember that, unless you have issued the command `\autolabelfalse`, there are lots of situations in which a label, and consequently an index entry, is generated automatically: in particular, this happens for each sectioning command (Section 5) and for each tagged theorem, lemma, and so on (Section 6). This provides a wealth of index entries, which later you can winnow out semi-automatically, if necessary, since entries of this sort appear in quotation marks, preceded by their type. You can see all the entries that a given page contributes to the index by looking at the small print on the top left of the page when proofing mode is on (Section 13).

Frank Mittelbach
†dagger
% No More Fragile Commands
Section "Proofing Aids"

* The `geom` style modifies the `\theindex` format of standard LaTeX so that the index is printed in smaller type. Moreover, it achieves a double-column format not by using standard LaTeX's `\twocolumn` command, which starts a new page, but rather Frank Mittelbach's `multicols` environment [Mit89], which is implemented in the file `multicol.sty`. Using this environment you can have single-column and double-column text on the same page. If you want to add an explanation before the index, for example, you might say something like this:

```
\begin{theindex}
\null             % this avoids what seems to be a bug in multicol.sty
\end{multicols}  % end double column format started by \begin{theindex}

Numbers in parentheses refer to pages where the entry should be
discussed but isn't, due to laziness on the author's part.

\begin{multicols}2 % restart double column format
\item aardvark...
\end{theindex}
```

Support for a glossary is somewhat more extensive than in standard LaTeX. You can mark a word as a glossary entry by saying `\glo{...}`; this causes a †dagger to be placed before the word, and an index entry for the word to be generated in the file *jobname*`.idx`, preceded by `\dag`.

* Note that the `geom` style does not use a separate *jobname*`.glo` file; although standard LaTeX's `\glossary` command is still available if you insist in writing entries to that file, this command has not been made robust in the way that `\index` and `\label` have. (See Section 4.)

The glossary itself is delimited by `\begin{theglossary}` and `\end{theglossary}`. Glossary entries start like this:

```
\entry{doohickey} ...
\entry[\cite{...}]{thingamajig} ...
```

The entry name is printed in boldface, followed by a period. The contents of the optional argument are printed immediately after the entry name, before the period. The comments made above about double-column formatting apply to the glossary as well.

The table of contents is treated in much the same way as in standard LaTeX, but there is one important difference: starred sectioning commands, like `\section*`, do show up in the table of contents, unless you've said `\starredcontentsfalse`. Since one generally doesn't want the table of contents to include an entry for the table of contents itself, the definition of `\tableofcontents` in `geomart.sty` starts like this:

```
\def\tableofcontents{{\starredcontentsfalse\section*{Contents}...}}
```

The extra pair of braces limits the action of `\starredcontentsfalse` to this entry only.

13. Proofing Aids

The `geom` style provides a number of proofing aids that you can turn on and off by saying `\proofingtrue` and `\proofingfalse`. The default is for proofing to be on. These facilities

% The Index, the Table of Contents and the Glossary
RCS
Section “Appendix: mathfig—A System for Doing Typesetting in Mathematica”

are independent of standard LaTeX's `draft` option to the `book` and `article` style, whose only effect is to make overfull box markers visible.

If you use LaTeX's `\includeonly` mechanism [Lam86, 76], you can print your final version by passing the macro `\noproofing` as the argument to `\includeonly`. This not only turns off all proofing aids, but also prevents LaTeX from starting a new page for each input file. (Naturally, the page breaks will likely be different because of that, so you'll need to run the job twice to get cross-references right.)

∗ Normally you have two lines like this in the main file for your document:

```
\typein[\sectionstoinclude]{Enter the sections that should be processed: }
\includeonly{\sectionstoinclude}
```

so you can type in the sections to be included from the terminal, without having to modify the source file. The macro `\noproofing` works in this mode, too.

By the way, if you are using the `book` style, the construction `\chap1{345}` expands to

```
chap1/1.3,chap1/1.4,chap1/1.5
```

This has nothing to do with proofing, but it provides a useful shorthand for use with `\includeonly`. Braces should be used around a section or chapter number having more than one digit.

In proofing mode you get, to the left of the main text, a column in tiny print with all the index entries contributed by that page. This includes automatic entries with their respective prefixes (see Section 12).

If you are in proof mode and the macros `\leftfoot` and `\rightfoot` are defined, they will be typeset flush left and flush right on the footline. These macros are often defined to contain information about draft version, date of last modification, etc., which can be updated automatically.

∗ Near the top of the source file for this document, I wrote the lines

```
\def\leftfoot{\RCSstring$Revision: 1.10 $}
\def\rightfoot{\RCSstring$Date: 92/08/25 19:55:41 $}
\def\RCSstring$#1 ${#1}
```

The first two lines are automatically updated by the UNIX utility RCS (Revision Control System) whenever I check the file in or out. RCS requires that the keywords `Revision`, `Date` and so on be found within dollar signs, and it tacks on a blank before the closing `$` for good measure. The `\RCSstring` macro eliminates this extraneous stuff.

Appendix: `mathfig`—A System for Doing Typesetting in Mathematica

Warning. The software described in this Appendix depends on Mathematica [Wol91]. I make no representation whatsoever concerning its performance. Although I have used `mathfig` under Mathematica versions 1.2, 2.0 and 2.1, I emphasize that `mathfig` may fail to work as described here and that I take no responsibility for such failure.

% PostScript Figures
% Installation
% PStorus

The `mathfig` program, for use on UNIX systems, takes a Mathematica source file `foo.m` and calls Mathematica to create a PostScript file `foo.ps`. Then `mathfig` postprocesses `foo.ps` to take out the PostScript labels putting the same information in a file `foo.lab` that can be read later by LaTeX, as explained in Section 10. The usage of this program is the following:

`mathfig` [`-d` *directory*] [`-i` *initfile*] *files*

The `-d` option makes all file names relative to the specified directory. The `-i` option tells Mathematica to read an initialization file. Any number of files can be passed as subsequent argument; `\mathfig` processes one at a time, starting a new Mathematica session each time.

The steps performed by `mathfig` when processing the file `foo.m` are the following. It calls `math` and gives it the commands `Get["`*initfile*`"]` (if an initialization file was specified) and `Get["foo.m"]`. The result of `Get["foo.m"]` should be an expression with head `Graphics` or `Graphics3D`. This expression is passed to `Display`, with output sent to the filter `geomfix`; this filter is analogous to the `psfix` program supplied with Mathematica. The output of `geomfix` is further processed to create the files `foo.ps` and `foo.lab`. This last step is performed by `mathlabels`, a `perl` program. (`perl` is a widely used UNIX utility, and it should be available at your site. If `mathfig` claims that `mathlabels` cannot be found, it may be because `perl` is not where `mathfig` expects to find it; see the end of Section 3.)

By default, `\mathfig` creates a figure two inches high and five inches wide. A different height can be specified by including the assignment `$height=`*height* in the Mathematica source file, where *height* is a number specifying the height in inches. You can also say `$width=`*width*. The reason to put these numbers in the source file, rather than choosing them at run time, is archival: in this way the output is entirely determined by the contents of the source file.

* Furthermore, if you assign values to the variables `$xlo`, `$xhi`, `$ylo` and `$yhi`, the figure produced by Mathematica will be clipped to a window with the given boundaries, and rescaled so as to fit in a box of the given height and width. For example, setting `$xlo=$ylo=.1` and `$xhi=$yhi=.9` will trim 10% off the size of the box around each edge, and the remaining graphics will be rescaled by a factor of $1/(.9-.1)=1.25$. Under Mathematica 1.2, this is the only way I know to make a 3D-graphics object occupy the whole bounding box. Unfortunately the values of `$xlo` and so on have to be determined by trial and error.

The symbols `$height`, `$width`, `$xlo`, `$ylo`, `$xhi`, `$yhi` and `$Def` should not be used in your Mathematica source file except with the meanings described above. (`$Def` is used internally.)

When Mathematica generates a picture with labels, the scaling computations are implicit in the resulting PostScript code, and they take into account the dimensions of the labels present in the code. Thus the graph generated by

```
Plot[x,{x,0,1}]
```

looks a bit smaller on the screen than the one generated by

```
Plot[x,{x,0,1},Ticks->False]
```

because the first has to accomodate labels along the axes, in the same size window. However, when you run `mathfig`, the labels are taken out of the PostScript file, and therefore don't have any effect on the scaling of the picture. The `mathfig` output for the two commands above would be scaled by exactly the same amount, and the TeX labels generated for the first command would stick out of the bounding box a little bit, just as they do in Figure 2.

Section "References"

References

[Ame] American Mathematical Society. $\mathcal{AMS}$-LaTeX version 1.1 user's guide (1991). Distributed with the software (use anonymous ftp from `e-math.ams.com`).

[Dar] Trevor Darrell. Psfig/TeX 1.8 users guide (1991). Distributed with the software (use anonymous ftp from `whitechapel.media.mit.edu`).

[ECL+92] David B. A. Epstein, Jim Cannon, Silvio Levy, Derek Holt, Mike Paterson, and William Thurston. *Word Processing in Groups.* Jones and Bartlett, Boston, 1992.

[Lam86] Leslie Lamport. LaTeX*: A Document Preparation System.* Addison-Wesley, Reading, MA, 1986.

[Mit89] Frank Mittelbach. An environment for multi-column output. *TUGboat*, 10:407–415, 1989.

[Rok] Tom Rokicki. DVIPS: A TeX driver (version 5.490, 1992). Distributed with the software (use anonymous ftp from `labrea.stanford.edu`).

[S+88] Mike Schuster et al. *Adobe Illustrator.* Adobe Systems, Palo Alto, CA, 1988.

[Wol91] Stephen Wolfram. *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley, Reading, MA, 2nd edition, 1991.