SynthSlant

A LaTeX Package for Artificial Text Slanting

Ch. L. Spiel[*]

v0.2   2025/10/27

Abstract

The synthslant package allows for the artificial slanting of text. It is designed to simulate slanted fonts for *short* text segments and create effects such as upright italic. It includes macros and environments to control the slanting process, enabling both forward and backward slanting of glyphs.

The manual also discusses how to determine and match the slant of italic fonts, facilitating integration with existing document typography.

fga
fga fga
fga

---

[*] cspiel@users.sourceforge.org

Contents

Table of Contents continued on next page.

The font samples ‘fga’ on the title page were generated with the help of METAPOST using “URW Palladio” in styles ‘roman’ and ‘italic’. The affine transformations were `slanted .2` for the slanted roman and `slanted -.2` for the upright italic.

List of Tables

List of Figures

Quick Reference

This is an alphabetically sorted list of all user macros and environments defined by package synthslant along with the page numbers of their descriptions. The list of all package options can be found on pages 7 and 8. The Index on pages 35 to 37 may provide some more detailed insights.

`\negslantcontext`
: Name of the microtype context used when typesetting backward slanted text. **14**

`negslantenvironment`
: Wrapper around `\synthslantbox` when slanting backward with `\textsynthuprightitalic`. **13**

`\slantcontext`
: Name of the microtype context used when typesetting forward slanted text. **14**

`slantenvironment`
: Wrapper around `\synthslantbox` when slanting forward with `\textsynthslant`. **13**

`\synthnegslant`
: Slant value used by `\textsynthuprightitalic`. **10**

`\synthslantbox{`⟨*slant*⟩`}{`⟨*text*⟩`}`
: Slant ⟨*text*⟩ (forward or backward) by ⟨*slant*⟩. **13**

`\synthslant`
: Slant value used by `\textsynthslant`. **9**

`\textsynthslant{`⟨*text*⟩`}`
: Forward slant roman, i. e., upright ⟨*text*⟩. **11**

`\textsynthuprightitalic{`⟨*text*⟩`}`
: Backward slant italic ⟨*text*⟩. **12**

Quick Start

For all users who want just the functionality, not the explanations, here is an example that shows how to use the most important macros.

```
\documentclass{article}

\usepackage{mlmodern}% load an example font

% Package Options, p. 7
% Slant values are suitable for ML Modern.
\usepackage[posslant=.23, negslant=-.21]{synthslant}

\begin{document}
% Variable-Like Macros, p. 9
% The redefinitions are unnecessary since posslant and
% negslant have already been specified as package options,
% but they can be used anywhere in the document.
\renewcommand*{\synthslant}{.23}
\renewcommand*{\synthnegslant}{-.21}

% Basic Interface, p. 10
% Slant upright shape to the right by \synthslant.
\textsynthslant{Generierter schräger Schriftschnitt,}%  (1)
\textsynthslant{Antiqua-Schrift,}%                      (2)
\textsynthslant{Sauer\-stoff\-fla\-sche,}%              (3)
\textsynthslant{Be\discretionary{tt-}{t}{tt}uch.}%      (4)

% Slant italic shape to the left by -\synthnegslant.
\textsynthuprightitalic{Aufrechte kursive Schriftlage.}

% Advanced Interface, p. 13
% Use separate slants for each box.
\synthslantbox{-.25}{H}
\synthslantbox{0}{H}
\synthslantbox{.25}{H}
\end{document}
```

The macros `\textsynthslant`, `\textsynthuprightitalic`, and `\synthslantbox` basically typeset their arguments in LR mode, i. e., as if wrapped in an `\mbox`. Synthslant reintroduces line breakability at spaces, hyphens, discretionary hyphens, and `\discretionary` primitives. The tagged lines in the above example thus present the following line breaking opportunities:

(1) Two breakpoints at the spaces.
(2) One breakpoint after the hyphen.
(3) Three breakpoints at the discretionary hyphens.
(4) One breakpoint at the `\discretionary`.

For an example in math mode see the tip on Page 12.

1 Introduction

The synthslant package provides a translator-independent way to shear glyphs. This means it works with LaTeX, pdfLaTeX, and LuaLaTeX. It implements a generic operation where a short piece of text gets slanted forward or backward. Moreover, specialized macros for the two most important use cases are provided, namely slanting an upright font forward and making an italic font upright. Unbeknownst to some users, pdfTeX performs a similar operation under the hood: of the 40,210 map lines in our *pdftex.map*, currently 1,236 instruct pdfTeX to artificially slant a font. This means approximately three percent of the shapes are generated this way.

Similar transformations can be achieved by other means; we elaborate on some of the alternatives in Sec. 6 on pages 19–21. Package synthslant however focuses on ease of use and strict locality of the glyph manipulation.

1.1 Technical Terms: Italic – Oblique – Slanted

We shall use several technical terms that describe the type or shape of a font. To avoid any confusion we define them right here.

Italic: An italic font is a complement to a given serif[1] font. It is drawn by a human font designer, who interprets the original serif font in a fundamentally artistic way.

The title page already shows an example: URW Palladio upright and italic in light gray, artificially slanted in black.

Oblique: An oblique font is a complement to a given sans-serif font. Some oblique fonts are designed in a tradition similar to that of italic fonts. These may feature for example single-story ‘a’s, whereas the original sans-serif fonts have double-story ‘a’s.[2] Oblique fonts that are artificially slanted versions of upright fonts also exist.

Slanted: A slanted[3] font is always created mechanically by applying a shear-transform (explained in Sec. 1.4 on p. 4) to the glyphs of a given font. The original font can be serif, sans serif, upright, italic, oblique, or mathematical.

To be clear, we sometimes add the adverbs ‘artificially’ or ‘synthetically’ to the term ‘slanted’.

In practice, these terms are used imprecisely; for example, an oblique sans-serif font may be called ‘italic’, or the notions of ‘oblique’ and ‘slanted’ may be conflated. We strive to be precise in this document, though.

---

[1] Called ‘roman’ in TeX’s technical terms.

[2] An oblique font’s characteristics even may vary for different releases, e. g., for Frutiger Italic (1976) compared to Frutiger Next Italic (2000) or Neue Frutiger Italic (2009) [14, p. 69].

[3] BRINGHURST uses the term ‘sloped’ [5, p. 49].

1.2 Appeal for Slanted Type

Artificially slanted typefaces have a poor reputation, much like artificial bold[4] or condensed ones.[5] This criticism can be traced for synthetic bold and condensed variants. They spoil the glyphs' outlines because they do not (and cannot) conserve the necessary proportions. For small caps the problems are somewhat minor; it is worth testing how far one can go using an OpenType font that supports a `size` axis as well as an `opsz` axis in the necessary ranges to construct convincing small caps out of the multiple master font.[6]

A common criticism of synthetically slanted type is that it lacks the contrast of true italic [22, p. 141]. However, in many settings, this reduced contrast is sufficient. Furthermore, a slight contrast with the main typeface is of secondary concern; it does not devalue the shape *per se* as is the case for artificial bold and condensed fonts.

What seems to have been lost in the discussion is the fact that true italic, designed alongside roman type, exhibit markedly different shapes. If we have an unbiased look at it—for example at the title page of this manual—the italic versions of the upright characters are so markedly different that it is worth asking whether they truly match the upright forms. For the double-storey ⌞a⌝ becomes single storey, the start of the loop of ⌞g⌝ moves from the far left to the middle. Besides, the aspect ratio of both counters change. These differences challenge the common guidelines [21, Ch. 6] for font pairing. The apparent contradiction makes sense when we recognize that italic are not just slanted versions of roman letters. They introduce tension through several additional design features. A famous quote of Zuzana Ličko applies once again:

> The most popular typefaces are the easiest to read; their popularity has made them disappear from conscious cognition. It becomes impossible to tell if they are easy to read because they are commonly used, or if they are commonly used because they are easy to read.

1.3 Some History

Italics accompanying a roman font date back to one of the earliest print shops, namely that of Aldo Manuzio (lat. Aldus Manutius) around 1500. Probably the first sans-serif font with an oblique version was Venus, released as an upright family of fonts in 1907 and accompanied by italic (ger. *kursiv*) since 1910 [10]. Artificially slanted versions of upright fonts, also known as 'oblique', appear in the twentieth century, when type designers and foundries start to save time and money by automatically constructing slanted versions of roman type [14, p. 68n]. Synthslant follows in their footsteps, offering similar functionality.

---

[4] Package amsbsy defines a "Poor Man's Bold" macro `\pmb` that works by 'overprinting'. The authors of amsbsy recommend to prefer package bm for bold mathematical symbols, though.

[5] See for example reference 22, p. 97, but compare with p. 142 and also reference 14, p. 68n, for a more nuanced assessment.

[6] FontForge [25] provides a means to generate small caps of any given glyphs, which allows to control the x-scale and y-scale factors and the widths of the stems. GUI sequence: Element > Style > Add Small Capitals, script function: `SmallCaps`, and Python method: `addSmallCaps`.

Some fonts in current LaTeX distributions offer slanted series out of the box. In particular, the oldest (and once upon a time the only) font family shipping with TeX, CM Roman—nowadays member of the CM-Super family—is available in a variety of almost thirty shapes, which is quite remarkable. It covers not just slanted roman or slanted small caps, but also slanted typewriter and, somewhat surprisingly, upright italic. Only a few font families come with upright italic; see Table 1. Furthermore, the LaTeX 2$_\varepsilon$ font selection scheme provides ‘`sl`’ for slanted shapes and ‘`ui`’ for upright italic [11]. The former is associated with the macros `\slshape` and `\textsl`.

TABLE 1: Some font families with upright italic.

Font
Anona
Auto 3
Coline Extrême
FF Seria
Goudy Old Style
Joanna
Literata
Odile
Romanée
Seria

TABLE 2: A short list of some variable fonts with a slant axis (`slnt`).

Font
Adapter
Armoire
Cairo
Commissioner
Geologica
Gluten
Inter
Recursive
Ritualist
Roboto Flex

Refer to Tab. 2 for a brief list of variable fonts[7] that offer a slant-axis[8] that can be controlled with fontspec's `Slant`[9] key and Tab. 3 for a rather incomplete list of fonts that are shipped with slanted shapes. For these fonts synthslant is largely superfluous unless e. g. they also come with an italic shape that is to be typeset upright.

It seems that the original idea of automatically shearing text in LaTeX to simulate a slanted shape goes back to DAVID CARLISLE who suggested to use the pdfTeX primitive `\pdfliteral` for shearing [7]. Shortly thereafter BRUNO LE FLOCH pointed to another pdfTeX primitive, namely `\pdfsetmatrix`, available with (in 2013) more recent pdfTeX versions [24].[10] With the help of the latter affine transformations of arbitrary content can be coded directly by setting the transformation matrix. A slight variant of his code is used in this package for

---

[7] See also the LaTeX Font Catalogue for Fonts with OPENTYPE or TRUETYPE Support and search Google Fonts for families of variable fonts with a `slnt` axis or fonts with an unusual variation at Variable Fonts.

[8] The registered axis is called `slnt` and it is not to be confused with the `ital` axis.

[9] Since fontspec version 2.9a as of 2024/2/13.

[10] The user-level manipulation of the transformation matrix has been part of the PDF standard since its initial publication in 1993 [3, Secs. 3.8 and 3.9] in the form of operator `cm` (‘concat’—concatenate matrix to current transformation matrix). ¶ The primitive `\pdfliteral` was implemented already in the first release of pdfTeX in 1998 [23] and the primitive `\pdfsetmatrix` joined 2007 in pdfTeX version 1.40.0 [19].

TABLE 3: Selected font families that come with their own slanted series.

Serif Font	Sans-Serif Font
Arvo	Cabin
CM Roman	Clear Sans
Domitian	Cuprum
Droid Serif	Fira Sans
Erewhon	Gandhi Sans
Extended Charter	INRIA Sans
GFS Artemisia	Lato[†]
GFS Bodoni	Montserrat
GFS Didot	PT Sans
	Source Sans Pro

[†] The shape is activated with `\itshape`.

the PDF and l3draw slant engines. The implementations for PSTricks, Ti*k*Z, and fontspec are trivial as they build upon shear functions supplied by the packages.

1.4 Shear Transformation, Slant, and Angle

Mathematically, the slant operation is a shear transformation, which can be expressed with the equation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & \sin\alpha \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}, \tag{1}$$

where the vector $(x, y)^{\mathrm{T}}$ is mapped to $(x', y')^{\mathrm{T}}$ and both are elements of the two-dimensional drawing plane $\mathbb{E}^2$. Compare with Figure 1.

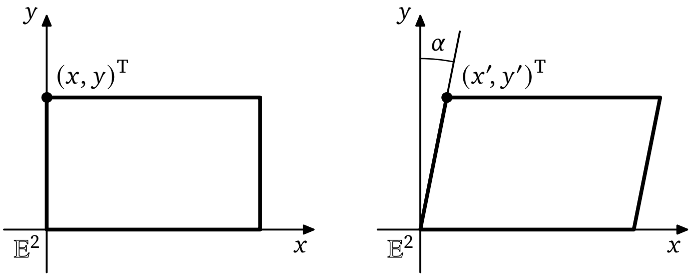


FIGURE 1: Shear transform of a rectangle by the angle α. The left-hand side shows the original figure the right-hand side the one sheared by α. The x-axis can be identified with the baseline of the text.

For $\alpha = 0$ the shear matrix becomes the identity matrix. Throughout synthslant, we use the ⟨*slant*⟩, which is $\sin\alpha$ in Equation 1, avoiding the need to convert back and forth to the shear angle α.[11] Some values for orientation: $\sin 5.74^\circ \approx .1$, $\sin 11.5^\circ \approx .2$, and $\sin 17.5^\circ \approx .3$.[12] For real-life serif fonts ⟨*slant*⟩ is in the range of .1 to .45 and a value of .2 seems to be quite common. Our *pdftex.map* lists negative ⟨*slant*⟩ values in the range −.4 to −.05 and positive ⟨*slant*⟩ values in the range .14 up to .45. See Table 4 on p. 9 for some actual values of serif fonts in LaTeX.

1.5 Usage Ideas

Automatic slanting, both forward and reverse, can be applied in a variety of typographic situations. Here are some usage ideas.

1a. Generate a slanted serif font when a serif font lacks italics, such as "URW Antiqua".

   Here, the user is relatively free to choose a ⟨*slant*⟩ value, as there are no italic whose angle must be matched. Synthslant's default of 0.2 should be a good starting point.

1b. If a secondary serif font—again assumed to have no italics or obliques—is paired with a primary serif font which has italics, the slant angle of the former can be matched to that of the primary font. An example of such a constellation is "Gentium" paired with "Eczar".

3. Augment a serif font that features an italic shape with upright italics.

   In nearly all cases, it is desirable to retain a slight forward slant of 1° to 2°.

4. An italic shape that has an excessive slant angle, such as "Libre Caslon", may be corrected—i. e., partially unslanted.

   Here, and generally if the entire font needs correction, alternative approaches such as those discussed in Sec. 6 (19–21) may be warranted.

5. Generate an oblique sans serif if a sans-serif font comes without an oblique shape as, e. g., "URW Grotesk".

6. Supply a slanted sans-serif shape for sans-serif fonts with designed, with *true* obliques such as e. g. "Open Sans".

7. Fixed width—also called 'typewriter' or 'teletype'—fonts without obliques (Yes, I am looking at you, Inconsolata!) finally get an oblique shape.

8. Some italic fonts only provide upright versions for selected glyphs, e. g. the square brackets. With synthslant these renegade characters can be slanted to match the italic font's natural angle.

9. Small caps without accompanying italic can be slanted, too.

---

[11] At least one slant engine currently requires such a conversion, namely PSTricks. The math is hidden from the user, though.

[12] For small angles $|\alpha|$ measured in radians the sine is approximately linear: $\sin\alpha \approx \alpha$.

10. As synthslant also works in TEX's math-mode, it is possible to give math-italic even more of a heeling.
11. If the slant of the math script font is at odds with the slant of the usual math italic, it may be possible to apply synthslant on the script symbols for matching angles.
12. Big mathematical operators like the sigma can be slanted and others, like the integral sign, can have their inclination adjusted.

It is possible to obtain slants that run against the reading direction, so called 'backslanted' glyphs, but we have rarely seen an example where the typography of a document could benefit from that.

2 Package Options

`\usepackage[⟨option⟩...]{synthslant}`

This section lists the ⟨*option*⟩s that the SynthSlant package understands. The package options allow to predefine the forward and backward slant angles as well as the selection of a particular slanting engine.

If no ⟨*option*⟩s are specified, synthslant defaults to `auto` with a slant value of `slant=.2`.

`auto`

Let the package choose a slant engine. This is the default.

For pdfLaTeX package synthslant selects the PDF engine, for LuaLaTeX the fontspec engine, and in all other cases the l3draw layer handles the shear transformation.

`disable`

Disable slanting completely.

`fontspec`

Use fontspec [16] as slanting back end.[13]

`l3draw`

Select the 'draw' layer of LaTeX3 as base for the slanting engine.[14]

Caution

This engine is experimental and the 'draw' layer of LaTeX3 itself is still experimental, too. See Sec. 5 on p. 18 for details. ■

`negslant=`⟨*slant-expr*⟩

Set the default value for `\synthnegslant` *only*. The argument ⟨*slant-expr*⟩ is a floating-point expression. Note that for this option ⟨*slant-expr*⟩ must evaluate to a nonpositive value.

`PDF`, `pdf`

Select the PDF-slant engine. This option requires that the document is translated with pdfLaTeX or a compatible program.

`posslant=`⟨*slant-expr*⟩

Set the default value for `\synthslant` *only*. The argument ⟨*slant-expr*⟩ is a floating-point expression. Note that for this option ⟨*slant-expr*⟩ must evaluate to a nonnegative value.

`PS`, `ps`

Use PSTricks to delegate slanting to the PostScript interpreter. Obviously requires PSTricks[15] and DVI-to-PostScript translation.

---

[13] Requires LuaTeX and *fontspec.sty*.

[14] This option requires *l3draw.sty*.

[15] The package actually required is *pst-3d.sty*.

Caution

This engine is still experimental and produces low-quality output! See Sec. 5 on p. 18 for details. ■

`slant`=⟨*slant-expr*⟩

Set the default values for both `\synthslant` and `\synthnegslant`, this is, act as if the two package options `posslant` = ⟨*slant-expr*⟩ and `negslant` = −(⟨*slant-expr*⟩) have been given. The argument ⟨*slant-expr*⟩ is a floating-point expression.

`TikZ`, `tikz`

Use TikZ for slanting.[16]

Caution

This engine is still experimental and produces low-quality output! See Sec. 5 on p. 18 for details. ■

The package options `slant`, `posslant`, and `negslant` all accept floating-point *expressions* as their arguments not just plain floating-point literals. See reference 12, Ch. 29, “The l3fp module – Floating points” for a description of the floating-point expression syntax and the available functions.

---

[16] Requires *tikz.sty*.

3 Macros and Environments

This section describes how to actually apply the functionality of synthslant to some text. If the ⟨*slant*⟩ value matching a given font is known this is about it. To figure out an unknown ⟨*slant*⟩ value check out Sec. 4.

3.1 Variable-Like Macros

The amount of slanting forward (positive slant angles) and backward (negative slant angles) is controlled by two macros. They are set during package initialization. However, they can be changed at any time to accommodate for different fonts or special needs.

`\synthslant` Control the slant applied by `\textsynthslant`. This value is nonnegative.

```
\synthslant
```

To change the slant value to .24 say

```
\renewcommand*{\synthslant}{.24}
```

Table 4 summarizes some suggested slant values for selected fonts.

TABLE 4: Suggested slant values for selected *serif* fonts. The ⟨*slant*⟩ shown in the tables is not necessarily the one closest to the font's italic. Also compare with the left-hand table of Tab. 3.

Font	Slant
ADF Accanthis	.26
ADF Baskervald	.32
ADF Berenis	.2
ADF Venturis	.2
Alegreya	.2
Aleo	.13
Arvo	.2
BaskervilleF	.2
Bera Serif	.2
Bitter	.16
Brill	.22
Caladea	.14
Castoro	.18
Charis SIL	.17
Clara	.24
Cochineal	.2
Coelacanth	.2
Crimson Pro	.2
Crimson Text	.2
Day Roman	.2
EB Garamond	.3
etbb	.2
Faustina	.15
fbb	.2
Fraunces9pt	.27
Garamond Expert	.2
Gandhi Serif	.2
Gentium	.2
Ibarra Real Nova	.2
IBM Plex Serif	.24
INRIA Serif	.2
Libertinus Serif	.2
Libre Baskerville	.3
Libre Caslon	.38
Lora	.07
Merriweather	.14
ML Modern	.23
Neuton	.16
Noto Serif	.22
PT Serif	.2
Roboto Slab	.2
Quattrocento	.2
Source Serif Pro	.18
Spectral	.18
STIX	.2
TeX Gyre Pagella	.16
TX Fonts Serif	.2
URW Antiqua	.2
URW Nimbus Roman	.2
Utopia	.2
Vollkorn	.17

`\synthnegslant` Control the slant applied by `\textsynthuprightitalic`. This value is nonpositive.

```
\synthnegslant
```

3.2 Basic Interface

The synthslant package provides two easy-to-use macros for slanting glyphs. For a more flexible and powerful interface, see Sec. 3.3.

Note

The following restrictions and workarounds to get line breaking and TEX's automatic hyphenation working again do *not* apply to the fontspec back end. ■

Both macros, `\textsynthslant` and `\textsynthuprightitalic` offer basic support for slanting explicitly hyphenatable words and space-separated phrases within a given ⟨*text*⟩. The fundamental shear transformation produces a single unbreakable horizontal box. We have added some provisions to re-enable at least some breakability.

SUPPORT FOR `\space` SINCE V0.2

1. Spaces (literal '␣' or `\space`) introduce breakpoints, e. g.

   `\textsynthslant{topological dual space}`

   slants the first word (producing a horizontal box) inserts a space and then slants the second word (producing another horizontal box) and so on. TEX sees three (unbreakable) boxes and a discardable space when it comes to linebreaking.

2. Discretionary hyphens in the form of '`\-`' get propagated. So, we could improve on our above example by saying

   `\textsynthslant{topo\-log\-i\-cal dual space}`

   to 'recover' hyphenation of the first word.

SUPPORT FOR `\discretionary` SINCE V0.2

3. Explicit `\discretionary` primitives are honored. The full glory of `\discretionary` is required for example to typeset the German compound nouns 'Ballettänzer', 'Bettuch', 'Brennessel', and 'Schiffahrt' while sticking to the "old orthography". Also see the example below.

This is neither a complete nor an elegant solution but it will take us quite far.

Note

Synthslant splits ⟨*text*⟩ at spaces. This leads to some inconvenient interactions with TEX or LATEX constructions that themselves need white space as terminators, e. g. `\kern` or `\skip` or starred macros that normally consume white space on their right-hand sides with the help of `\ignorespaces`.

A workaround in these cases is to terminate the constructions with `\relax` as white space after `\relax` is handled the usual way. Also see the following example. ■

***Example* — Manually breaking ligatures**

If a ligature is broken inside of `\textsynthslant` with an explicit kern, the `\kern` must immediately be followed with a `\relax` otherwise the space that normally is used as macro-argument terminator gets picked up by the gearing inside of `\textsynthslant`.

```
\textsynthslant{%
    Schi\discretionary{ff-}% ligature ok here
                      {f}
                      {f\kern.0333pt\relax f}ahrt%
}
```

This problem can be circumnavigated by wrapping the `\kern` in a parameterless macro; a simpler way would be using macro `\nolig` of package typog [20] right from the beginning. ■

`\textsynthslant` Forward slant some upright glyphs.

```
\textsynthslant{⟨text⟩}
```

In horizontal mode switch to an upright shape, slant ⟨*text*⟩ with the slant value stored in `\synthslant` and apply "slant correction"—the equivalent of italic correction—at the right-hand side of ⟨*text*⟩.

In math mode, just slant ⟨*text*⟩ with the slant value stored in `\synthslant`.

Tip

Discriminating typesetters will want to include trailing punctuation in ⟨*text*⟩. Compare for example:

```
\textsynthslant{FONT},
\textsynthslant{bar},
\textsynthslant{bay}.
```

FONT, *bar*, *bay*.

with

```
\textsynthslant{FONT,}
\textsynthslant{bar,}
\textsynthslant{bay.}
```

FONT, bar, bay.

This is similar advice as for italic and also holds for `\textsynthupright-italic` as well as any other slanting macro. ■

Use Cases

If italics seem to be too intrusive in the body, we can substitute slanted text, for example, for foreign phrases like 'et al.' and 'etc.':

```
\newcommand*{\foreignphrase}[2][USenglish]
            {\foreignlanguage{#1}{\textsynthslant{#2}}}
```

where we show the font modification in conjunction with the babel macro `\foreignlanguage` [4]. ¶

In math mode, you cannot have enough fonts, symbols, and most of the gizmos over there! We like to mark up automorphism groups associated with a given group with a slanted roman typeface, though our macro has a more general name.

```
\newcommand*{\functionspace}[1]
            {\mbox{\textsynthslant{#1}}}
```
■

`\textsynthuprightitalic` Backward slant some italic or oblique glyphs.

```
\textsynthuprightitalic{⟨text⟩}
```

In horizontal mode switch to an italic shape, slant ⟨*text*⟩ with the slant value stored in `\synthnegslant`.

In math mode, just unslant ⟨*text*⟩ with the slant value stored in `\synthnegslant`.

Example

To set apart operators in an algebra, like, e. g., the radical, we could use upright italic

```
\newcommand*{\algebraoperator}[1]
            {\mbox{\textsynthuprightitalic{#1}}}
```

and follow up with

```
\DeclareMathOperator{\rad}{\algebraoperator{rad}}
```

where we have assumed that amsmath [1] has been loaded to bring `\DeclareMathOperator` into scope. ■

Tip — Display Style Math Operators

Slanted or unslanted display style mathematical operators are best predefined with `\DeclareMathOperator` [1].

```
\DeclareMathOperator*{\slantedsum}
  {\synthslantbox{.15}{$\displaystyle\sum$}}
\DeclareMathOperator*{\uprightint}
  {\synthslantbox{-.15}
                 {$\displaystyle\int\negthickspace$}}
```

Now compare the pairs `\sum` `\slantedsum` and `\int` `\uprightint`:

$$\sum \quad \sum \qquad \text{and} \qquad \int \quad \int \tag{2}$$

In the following equation, we used `\uprightint` and `\slantedsum` like

```
\uprightint_{\negthickspace 0}^{\;\;1}
```

for the integral,

```
\slantedsum_{j=0}^{\;\;m}
```

for the first summation sign, and

```
\slantedsum_{s=1}^{\;\;l}{\vphantom{\slantedsum}}''
```

for the second summation sign:

$$\int_0^1 f(x)\,dx = \frac{1}{m}\sum_{j=0}^{m} f(j/m) - \sum_{s=1}^{l}{}'' \frac{B_{2s}}{2s!}\,\frac{f^{(2s-1)}(1) - f^{(2s-1)}(0)}{m^{2s}} + O\bigl(m^{-2l-2}\bigr).$$

The aesthetic appeal of this solution is doubtful; however, it demonstrates the capabilities of synthslant in math mode. ■

3.3 Advanced Interface

`\synthslantbox` Slant ⟨*text*⟩ with an amount of ⟨*slant*⟩ that can be positive, negative, or zero.

```
\synthslantbox{⟨slant⟩}{⟨text⟩}
```

This is the unadorned call to the chosen slanting engine. In particular, neither the values of `\synthslant` nor of `\synthnegslant` enter its expansion! No corrections or TEX-mode adjustments are made.

Example

Generate a substitute for a missing solidus character:

```
\renewcommand*{\textfractionsolidus}[1]
  {\kern-.125em
   \raisebox{.125em}
            {\smaller
             \synthslantbox{.3}{\char‘/}}%
   \kern.1em}
```

where the `\smaller` macro is from the relsize package [2]. ■

The following two environments are responsible for setting up everything before the actual slant or unslant code runs and what happens after the slant engine finishes. They can be redefined or patched to meet different needs.

`slantenvironment` (*env.*) This environment is a wrapper around `\synthslantbox` that is called for every forward slanting operation with `\textsynthslant`.

```
\begin{slantenvironment}
  ...
\end{slantenvironment}
```

Switch to an upright font shape and—if package microtype [18] has been loaded—enter the Microtype context defined by macro `\slantcontext`. At the end, add some slant correction, which is the equivalent of italic correction.

***Use Cases* — “Patch Cases”**

Left-italic correction. ¶ Simultaneous left-italic and right-italic correction for a shift-left effect. ■

`negslantenvironment` (*env.*) This environment is a wrapper around `\synthslantbox` that is called for every backward slanting operation with `\textsynthuprightitalic`.

```
\begin{negslantenvironment}
  ...
\end{negslantenvironment}
```

Switch to an italic font shape and—if package microtype [18] has been loaded—enter the Microtype context defined by macro `\negslantcontext`.

`\slantcontext` Name of the microtype context used when typesetting slanted text.

```
\slantcontext
```

The expansion of this macro may be empty. The package's default is

`tracking = synthslant`

Note

The tracking context `synthslant` is *not* defined by synthslant itself. And Microtype ignores undefined contexts. ■

`\negslantcontext` Name of the microtype context used when typesetting backward slanted text.

```
\negslantcontext
```

The expansion of this macro may be empty. The package's default is

`tracking = synthnegslant`

Note

The tracking context `synthnegslant` is *not* defined by synthslant itself. And Microtype ignores undefined contexts. ■

Example

Upright italic often looks somewhat tight. We like to add some extra tracking to them. So, we define a microtype context named `synthnegslant`:

```
\SetTracking[context = synthnegslant]
            {encoding = *, shape = it}
            {10}
```
■

Tip

When the tracking of upright italic is changed, it may be advisable

- to break ligatures, e. g., `no ligatures = {f}`,
- to adjust the outer kerning, e. g., `outer kerning = {0, 0}`, and
- to adapt the inter-word spacing, e. g., `spacing = {100„}`.

The document *synthslant-gauge.tex*, which comes with package synthslant, has sample texts and tracking variations already set up for experimentation.

■

4 How to Determine and Match Slant

When matching a synthetically slanted piece of text to an existing italic or oblique font, it is important to determine the slant angle α or $\langle slant \rangle$.

Note

The slant angles of different italic or oblique glyphs in the same font may slightly differ from each other. Usually, longer shapes have less slant than shorter shapes.

Seek a representative slant value, an average that achieves a visual match with the italics or obliques. ■

In the following, we suggest three techniques to determine or match the slant of a glyph with sufficient accuracy. Direct measurements of the slant angle (Sec. 4.1) claim to be the most precise. However, as the previous Note indicated, its accuracy is limited by the differing slant angles of the letterforms. The visual comparison of shapes (Sec. 4.2) appeals to the user's judgment of matching angles for several letters, thus inherently incorporating some kind of averaging. Two variants of this method are conceivable. One is to overlay italic letters with appropriately slanted versions of the regular font's letters in a graphics program (Sec. 4.2.1); another is to find upright italic (Sec. 4.2.2) and, in that way, determine the negative slant value, which is equally useful. All $\langle slant \rangle$ values of Table 4 on p. 9 were determined with the former method. The latter method is faster, does not require an extra application—just a previewer—and it yields surprisingly accurate results.

4.1 Direct Measurement

Measure the angle of some reference glyphs with a graphics program.

1. Prepare a page with some sample glyphs of the font shape to be matched.
2. Render it as PostScript or in PDF.
3. Load the file at a resolution of 1200 dpi or higher into your favorite graphics editor that supports measuring angles.
4. In the graphics editor, center the interesting letters and set the zoom to 100 percent or more.
5. Measure some letters and write down the angles.
6. Convert the angle α to a $\langle slant \rangle$ by calculating $\sin \alpha$.

   If no computer is available, the following approximation might help:

$$\langle slant \rangle = \sin \alpha \approx \frac{11}{630^\circ}\,\alpha,$$

   where α is given in degrees.

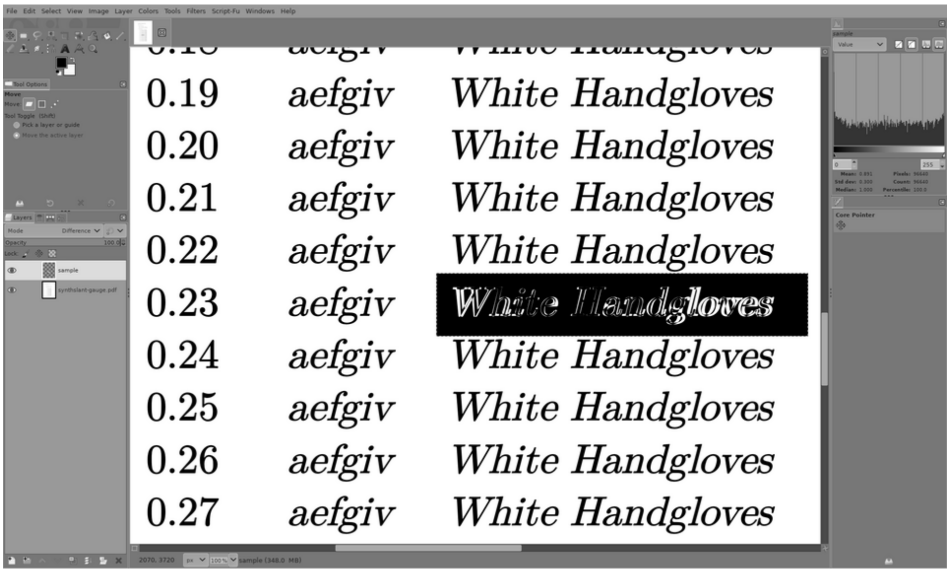
4.2 Comparison of Shapes

Compare some reference glyphs with a differently slanted or unslanted versions.

4.2.1 Slanted Upright & Graphics Program

Compare the italic or oblique shapes of a font with the synthetically slanted upright shapes in a graphics program.

1. In file *synthslant-gauge.tex*—which comes with the synthslant package—insert the code to load your font of interest.
2. Render the document as PostScript or as PDF.
3. Load the first page ("Slanted Samples") at a resolution of 600 dpi to 900 dpi into your favourite graphics editor.
4. Cut the italic sample at the top allowing for generous white space around it as a rectangle
5. Paste the rectangle in a new layer called e. g. 'sample'.
6. On layer 'sample' move the rectangle down the list of different slant values until it match best.
7. Switch the layer mode of 'sample' to 'difference' and fine position the rectangle over the slanted sample. Compare different letters in that way. Change line until the best match is found.
8. Read the slant value at the left-hand side of the line. See Fig. 2 on p. 16.


File Edit Select View Image Layer Colors Tools Filters Script-Fu Windows Help
Layers
Difference
sample
synthslant-gauge.pdf

0.19	*aefgiv*	*White Handgloves*
0.20	*aefgiv*	*White Handgloves*
0.21	*aefgiv*	*White Handgloves*
0.22	*aefgiv*	*White Handgloves*
0.23	*aefgiv*	*White Handgloves*
0.24	*aefgiv*	*White Handgloves*
0.25	*aefgiv*	*White Handgloves*
0.26	*aefgiv*	*White Handgloves*
0.27	*aefgiv*	*White Handgloves*

FIGURE 2: Compare italic and slanted samples with The Gimp. ¶ For this screenshot we loaded the samples on the first page of *synthslant-gauge.pdf* at a resolution of 600 dpi into The Gimp. The 'sample' layer is aligned to the letter 't' in the word 'White'. Note that accidentally the letter 'H' of the next word 'Handgloves' confirms the good match.

4.2.2 Upright Italic & Document Viewer

Assess synthetically upright italic with any document viewer.

1. In file *synthslant-gauge.tex*—which comes with the synthslant package—insert the code to load your font of interest.
2. Render the document as PostScript or as PDF.
3. Go to the second page ("Upright Italic Samples").
4. Magnify the page as necessary and look for a line where the italic look upright or leaning to the right ever so slightly.
5. Read the slant value at the left-hand side of the line. The sought after ⟨*slant*⟩ is the negative of this value.

4.3 Exploring Further

Once a usable slant value has been found, it can be fed into *synthslant-gauge.tex* and—after recompiling with the appropriate LaTeX engine—be used to examine the details of the slant operations.

Page 3, Sec. 2.1, 'Copy', shows wild mixes of different font shapes, native and synthesized ones. Here, the slanted glyphs as well as the upright italic should blend well with the native italic/obliques and with the normal font.

Page 4, Sec. 2.2 and following subsections, examine the coupling of synthslant with the TeX system and some of its extensions. If a slant engine malfunctions, it will become evident on this page.

5 Limitations and Known Problems

This section lists some known limitations and issues with synthslant. There may be additional issues not listed here.

All except fontspec: Synthslant manipulations may not survive preprocessing by METAPOST.

l3draw engine:

- Depending of the shear, direction the l3draw engine may generate some extra positive or negative space at the ends of the text.
- Any box sheared loses its depth; technically, `\dp` becomes 0pt.
- Markedly slower than the PDF implementation!

PSTricks engine: The PSTricks engine produces some extra space at the ends of the text.

Ti*k*Z engine: The Ti*k*Z engine produces some extra space at the ends of the text.

6 Alternative Solutions

Here are some alternative solutions to synthslant that we are aware of. All of them change the slant of a font as a whole.

6.1 Use pdfTeX

In pdfTeX, fonts can be remapped in the document preamble with the primitive `\pdfmapline`; see the pdfTeX Reference Manual [24, Sec. 6.1] for a description of the syntax. This possibility makes it possible to splice in a slanting operation on the fly.

Here is a simplified syntax of a font map line, which does not indicate any of the optional parts for better readability:

```
⟨tfm-name⟩ ⟨ps-name⟩ ⟨font-flags⟩
          "⟨special⟩" <⟨encoding-file⟩ <⟨font-file⟩
```

where

- ⟨*tfm-name*⟩ is the basename of the TeX font-metric file (**.tfm*),
- ⟨*ps-name*⟩ is the name the font will acquire inside of TeX,
- ⟨*font-flags*⟩ optionally specify some characteristics of the font,
- ⟨*special*⟩ prescribes font manipulations in the same way as **dvips** [17, Sec. 6.3] does,
- ⟨*encoding-file*⟩ is the filename (**.enc*) where the encoding to be used with ⟨*font-file*⟩ is stored, and
- ⟨*font-file*⟩ sets the filename of the font's definition. It is given without path but includes an extension, which typically is *otf*, *pfb*, or *ttf*.

We are particularly interested in the ⟨*special*⟩ part that allows us to slant the whole font with a single instruction.

Let us to elaborate the example given in Sec. 1.5, item 4 and generate a less-angled italic for Libre Caslon. Here is a suitable map line taken from *pdftex.map* on our system:

```
LibreCsln-Italic-osf-t1-base LibreCsln-Italic
    "␣AutoEnc...␣ReEncodeFont␣"
               <[lcsln....enc <LibreCsln-Italic.pfb
```

which we had to break into three lines to make it fit this page. The '`...`' indicate parts of the identifiers that we left out beyond that. There are in fact four map lines for `T1`-encoded Libre Caslon italic: those for lining figures '`lf`', oldstyle figures '`osf`', tabular lining figures '`tlf`', and tabular oldstyle figures '`tosf`'.

The slant operation we want to add to the ⟨*special*⟩ part has the format:

`⟨slant⟩ SlantFont`

so, for a shear to the left, for example, ⟨*slant*⟩ = −.12, which means the font gets slanted by −8° the ⟨*special*⟩ part becomes

```
"␣-0.12␣SlantFont␣AutoEnc...␣ReEncodeFont␣"
```

Finally, we select the modified font, e. g., with macro

```
\usefont{⟨encoding⟩}{⟨family⟩}{⟨series⟩}{⟨shape⟩}
```

See reference 13 for details. In our case, the call to `\usefont` is

```
\usefont{T1}{LibreCsln-OsF}{regular}{it}
```

Example

Here is all the talk of above put into action as this very document contains exactly the `\pdfmapline` just described.

Uncorrected, original italic	*White Handgloves*
Less angled version	*White Handgloves*

The only trick we have to reveal is that for the "original italic" we used the lining figures '`lf`' version of the font, whereas the "less angled" version shows the oldstyle figures '`osf`' version.

The TeX Font Metrics file (TFM) for this particular variant of Libre Caslon was not touched. ■

6.2 Combine LaTeX and dvipdfmx

The alternative when using LaTeX is similar the one elaborated in the previous section. The font mapline gets modified by `\special` primitive

```
\special{pdf:mapline ⟨font-mapline⟩}
```

that forwards the task of remapping the font, e. g. to **dvipdfmx**. Our running example becomes

```
\special{pdf:mapline
    LibreCsln-Italic-osf-t1-base LibreCsln-Italic
    "␣-0.12␣SlantFont␣AutoEnc...␣ReEncodeFont␣"
                <[lcsln....enc <LibreCsln-Italic.pfb}
```

The mapline contains **dvips** options for special font effects; see reference 17, Sec. 6.3. Note that there is no '='-sign at the beginning of the `pdf:mapline` in contrast to `\pdfmapline`.

The font is activated in the same way as in the PDF path (Sec. 6.1). The further translation of the resulting DVI file *must* be performed with an application that is aware of the `\special` primitive as for example **dvipdfmx** [8] is.

Note

Despite the option syntax originates with **dvips** it is not able to interpret any `\special{pdf:mapline ...}`. ■

6.3 Directly Use LuaLATEX and Package fontspec

The LuaTEX engine can be coaxed to transform glyphs similar to pdfTEX. With the support of package fontspec [16] slanting is available via the `\fontspec`-macro:

`\fontspec{⟨FONT-FILENAME⟩}[FakeSlant=⟨slant⟩]`

Actually, package synthslant uses a similar call to implement its own slanting macros if the package is loaded with option `fontspec`.

6.4 Harness a Font Editor

An alternative outside of the typical LATEX toolchain is to harness a font editor, as, for example, FontForge [25], to create a slanted version of an upright font or upright italic. Here, the first step is to check the font's license to see whether the generation of a variant is permitted and what restrictions apply to its use.

FontForge can generate slanted versions of a given set of glyphs with the GUI sequence: Element > Style > Oblique and the desired slant angle in degrees (though with the opposite sign that synthslant uses). In FontForge's own extension language, the corresponding function is called `Skew`; in the Python extension, use module function `psMat.skew` to generate the transformation matrix and apply method `transform` to a glyph with this matrix as the method's argument.

Note that some of the glyphs' attributes as, e. g., tracking, kerning, or hints, may be gone or wrong for the slanted versions of the selected glyphs.

When exporting the changed font from a font editor, users of pdfTEX and LuaTEX may choose different formats—for example, PFB *versus* TTF. If you plan to use the new font with METAPOST, you must export it in PFB format.

Finally, install the font, e. g., with autoinst which is part of fonttools [15], or using package fontinst [9]. If the slanted font is to integrate seamlessly with the existing fonts, usually some of the relevant font description files (*.fd*) have to be modified.

A Package Code

This is the “Reference Manual” section of the documentation where we describe the package’s code and explain its implementation details.

```
⟨*package⟩
\NeedsTeXFormat{LaTeX2e}[2005/12/01]
\ProvidesPackage{synthslant}
               [2025/10/27  v0.2  Synthetically Slant glyphs]

\RequirePackage{etoolbox}
\RequirePackage{iftex}
\RequirePackage{xkeyval}

```

A.1 Declaration of Default Slants

`\synthslant` Introduce a reasonable default for the slant. Let the users override it if they know better.

Remember that the slant is not an angle (with respect to the y-axis), but the sine of it. The value .2 approximately corresponds to a slant angle of 12°.

```
\providecommand*{\synthslant}{.2}
```

`\synthnegslant` Also introduce a reasonable default for the negative slant, which is used for upright italic.

```
\providecommand*{\synthnegslant}{-.2}

```

A.2 Selection of Slant Engine

We provide several methods to slant glyphs. The actual slanting is delegated to a ‘slant engine’ which shears the glyphs.

`\synthslant@engine` Default to automatic selection of the slant engine.

```
\def\synthslant@engine{-1}

```

Expose default forward and backward slant values as package options.

```
\DeclareOptionX{slant}{%
  \xdef\synthslant{\fpeval{#1}}%
  \xdef\synthnegslant{\fpeval{-(#1)}}}
\DeclareOptionX{negslant}{\xdef\synthnegslant{\fpeval{#1}}}
\DeclareOptionX{posslant}{\xdef\synthslant{\fpeval{#1}}}

```

Make slant-engine selection configurable.

```
\DeclareOptionX{auto}{\def\synthslant@engine{-1}}
\DeclareOptionX{PDF}{\def\synthslant@engine{0}}
\DeclareOptionX{pdf}{\def\synthslant@engine{0}}
\DeclareOptionX{l3draw}{\def\synthslant@engine{1}}
```

```
\DeclareOptionX{ps}{\def\synthslant@engine{2}}
\DeclareOptionX{PS}{\def\synthslant@engine{2}}
\DeclareOptionX{tikz}{\def\synthslant@engine{3}}
\DeclareOptionX{TikZ}{\def\synthslant@engine{3}}
\DeclareOptionX{fontspec}{\def\synthslant@engine{4}}
\DeclareOptionX{disable}{\def\synthslant@engine{10000}}

\ProcessOptionsX\relax

```

Require sane parameter values.

```
\ExplSyntaxOn
\fp_compare:nNnTF {\synthslant} < {.0}
  {\PackageError{synthslant}
                {\string\synthslant\space <\space 0}
                {Pass\space a\space value\space that\space
                 is\space nonnegative.}}
  {}
\fp_compare:nNnTF {\synthnegslant} > {.0}
  {\PackageError{synthslant}
                {\string\synthnegslant\space >\space 0}
                {Pass\space a\space value\space that\space
                 is\space nonpositive.}}
  {}
\ExplSyntaxOff

```

Announce the positive and negative slant values now that we are sure they are ok. This may be useful information if the user passed a (complicated) floating-point expression and wants to know how LaTeX did evaluate it.

```
\PackageInfo{synthslant}{\string\synthslant=\synthslant}
\PackageInfo{synthslant}{\string\synthnegslant=\synthnegslant}


```

A.3 Slant Engines

The auto selection code is pretty trivial. If we identify pdfTeX running we select the PDF engine, for LuaLaTeX we select the fontspec engine, and in all other cases we let the l3draw layer handle the shearing.

```
\ifnum\synthslant@engine<0
  \PackageInfo{synthslant}{auto-selecting slant engine}

  \ifpdftex
    \ifnum\pdfoutput>0
      \def\synthslant@engine{0}
    \else
      \def\synthslant@engine{1}
    \fi
  \else
    \ifluatex
```

```
      \def\synthslant@engine{4}
    \else
      \def\synthslant@engine{1}
    \fi
  \fi
\fi


```

`\synthslant@shear@box`

The various slant engine macros are all subsumed under `\synthslant@shear@box`. So, the higher-level code becomes (almost) engine independent.

`\synthslant@engine@name`

Sometimes we would like to recover the (printable) name of the selected slant engine.

```
\newcommand*{\synthslant@engine@name}{%
  \ifcase\synthslant@engine
    PDF%
  \or% 1
    l3draw%
  \or% 2
    PSTricks%
  \or% 3
    TikZ%
  \or% 4
    fontspec%
  \else
    null-implementation%
  \fi
}

```

A.3.1 PDF Slant Engine

The PDF engine works well and it is the best tested alternative.

```
\ifcase\synthslant@engine%  0: PDF
  \PackageInfo{synthslant}{shearing done by PDF}

  \newbox{\synthslant@box}

```

`\synthslant@pdf@shear@box`

```
  \newcommand*{\synthslant@pdf@shear@box}[2]{%
    \mbox{\sbox{\synthslant@box}{#2}%
          \hskip\wd\synthslant@box
          \pdfsave
          \pdfsetmatrix{1 0 #1 1}%
          \llap{\usebox{\synthslant@box}}%
          \pdfrestore}%
  }

  \let\synthslant@shear@box=\synthslant@pdf@shear@box
```

A.3.2 l3draw Slant Engine

Using LaTeX3 may be like cheating on a very high level as the draw subsystem may delegate to the PDF engine itself. LOL!

```
\or%  1: LaTeX3 draw subsystem
  \PackageInfo{synthslant}{shearing delegated to l3draw}

  \RequirePackage{l3draw}

  \ExplSyntaxOn
```

synthslant@latex@shear@box Slanting implemented with the experimental l3draw subsystem.

> ***Anticipated Change***
>
> As soon as the l3kernel offers an x-shear operation (`\box_xshear:Nn`?) we shall ditch this implementation and switch to the one that is tailored to *text* instead of the current one for graphics.

```
  \NewDocumentCommand{\synthslant@latex@shear@box}{mm}{
    \hbox_set:Nn \l_tmpa_box {#2}
    \dim_set:Nn \l_tmpa_dim {\box_wd:N \l_tmpa_box}
    \dim_set:Nn \l_tmpb_dim {\box_ht:N \l_tmpa_box}
    \draw_begin:
      \draw_transform_xslant:n {#1}
```

Force the baseline of the payload (#2) to coincide with the baseline of the surrounding text. This—of course—screws up our bounding box at least vertically.

```
      \box_set_dp:Nn \l_tmpa_box {\z@}
```

Here comes a fudge because the l3draw bounding boxes are way too lose. For positive slants: shrink the box width by the box height times ⟨*slant*⟩. For negative slants: shrink the box width as for positive slants and in addition shift the payload to the left by the box height times ⟨*slant*⟩.

```
      \fp_compare:nNnTF {#1} >= {.0}
        {
          \box_set_wd:Nn \l_tmpa_box
              {\l_tmpa_dim - #1\l_tmpb_dim}
        }
        {
          \draw_suspend_begin:
            \kern#1\l_tmpb_dim
          \draw_suspend_end:
          \box_set_wd:Nn \l_tmpa_box
              {\l_tmpa_dim + #1\l_tmpb_dim}
        }
```

Now typeset the box.

```
      \draw_box_use:N \l_tmpa_box
    \draw_end:
  }
```

```
  \ExplSyntaxOff

  \let\synthslant@shear@box=\synthslant@latex@shear@box
```

A.3.3 PSTricks Slant Engine

Shearing via PSTricks works, but exhibits a weird interface.

```
\or%  2: PSTricks
  \PackageInfo{synthslant}
              {shearing deferred to PostScript via PSTricks}

  \RequirePackage{pst-3d}% \pstilt

```

Package pstricks offers `\pstilt` and `\psTilt` both with typographically suboptimal outcomes.

`thslant@pstricks@shear@box`

```
  \newcommand*{\synthslant@pstricks@shear@box}[2]{%
    \pstilt{\fpeval{57.2958 * acos(#1)}}{#2}%
  }

  \let\synthslant@shear@box=\synthslant@pstricks@shear@box
```

A.3.4 Ti*k*Z Slant Engine

The Ti*k*Z code has not been tested thoroughly yet, but it looks like it could work after some tweaking.

```
\or%  3: TikZ
  \PackageInfo{synthslant}{shearing by TikZ}

  \RequirePackage{tikz}

```

`\synthslant@tikz@shear@box`

```
  \newcommand*{\synthslant@tikz@shear@box}[2]{%
    \tikz[baseline = (ANCHOR.base), xslant = #1]
    \node[inner sep = 0pt, xslant = #1] (ANCHOR) {#2};
  }

  \let\synthslant@shear@box=\synthslant@tikz@shear@box
```

A.3.5 fontspec

The fontspec works particularly well, but it does not jibe with pdfTEX.

```
\or%  4: fontspec
  \PackageInfo{synthslant}
              {use fontspec's artificial font transformations}

  \RequirePackage{fontspec}

  \ExplSyntaxOn
```

`antbox@fontspect@shear@box`

```
  \newcommand*{\synthslantbox@fontspect@shear@box}[2]{
    \begingroup
    \expandafter
    \fontspec[FakeSlant=#1]{\l_fontspec_family_tl}
    #2
    \endgroup
  }
```

```
  \ExplSyntaxOff

  \let\synthslant@shear@box=\synthslantbox@fontspect@shear@box
```

A.3.6 Null Implementation

The null implementation—which does exactly what its name implies—can be useful for debugging or to get rid of the effect temporarily.

```
\else%  >=5: Null implementation
  \PackageWarning{synthslant}{shearing disabled}

```

`thslant@identity@shear@box`

```
  \newcommand*{\synthslant@identity@shear@box}[2]{#2}

```

```
  \let\synthslant@shear@box=\synthslant@identity@shear@box
\fi


```

A.4 Generic Slant Code

Here comes the engine-independent code.

`\synthslant@nolinebreak` The LaTeX3 and Ti*k*Z engines break lines at 'unexpected' points. Here is a duct-tape solution for them that concretes together the adajacent parts.

```
\def\synthslant@nolinebreak{%
  \ifnum\synthslant@engine=1% l3draw
    \nolinebreak
  \else
    \ifnum\synthslant@engine=3% TikZ
      \nolinebreak
    \fi
  \fi
}

```

`ynthslantbox@discretionary` Honor explicit `\discretionary` primitives.

```
\def\synthslantbox@discretionary#1\discretionary#2#3#4#5\@nil{%
  \synthslant@shear@box{\synthslant@slant@value}{#1}%
  \ifx\relax#5%
```

```
    \relax
  \else
    \synthslant@nolinebreak
    \setbox0=\hbox{\synthslant@shear@box{\synthslant@slant@value}{#2}}%
    \setbox1=\hbox{\synthslant@shear@box{\synthslant@slant@value}{#3}}%
    \setbox2=\hbox{\synthslant@shear@box{\synthslant@slant@value}{#4}}%
    \discretionary{\box0}{\box1}{\box2}%
    \synthslantbox@discretionary#5\@nil
  \fi
}

```

`\synthslantbox@soft@hyphen` Allow for line breaks at hyphenation opportunities (‘`\-`’).

```
\def\synthslantbox@soft@hyphen#1\-#2\@nil{%
  \synthslantbox@discretionary#1\discretionary{}{}{}\@nil
  \ifx\relax#2%
    \relax
  \else
    \synthslant@nolinebreak
    \setbox0=\hbox{\synthslant@shear@box{\synthslant@slant@value}{-
  }}%
    \discretionary{\box0}{}{}%
    \synthslantbox@soft@hyphen#2\@nil
  \fi
}

```

`\synthslantbox@hard@hyphen` Allow for line breaks at embedded, explicit hyphens (‘-’).

```
\def\synthslantbox@hard@hyphen#1-#2\@nil{%
  \synthslantbox@soft@hyphen#1\-\@nil
  \ifx\relax#2%
    \relax
  \else
    \synthslant@nolinebreak
    \synthslant@shear@box{\synthslant@slant@value}{-}%
    \synthslant@nolinebreak
    \discretionary{}{}{}%
    \synthslantbox@hard@hyphen#2\@nil
  \fi
}

```

`lantbox@space@codesequence` This stage lets us cope with the macro `\space` in contrast to `\synthslantbox@literalspace`, which handles literal spaces ‘␣’.

```
\def\synthslantbox@space@codesequence#1\space#2\@nil{%
  \synthslantbox@hard@hyphen#1-\@nil
  \ifx\relax#2%
    \relax
  \else
    \space
    \synthslantbox@space@codesequence#2\@nil
  \fi
}

```

`ynthslantbox@literal@space`

Allow for line breaks at embedded spaces (‘␣’).

```
\def\synthslantbox@literal@space#1 #2\@nil{%
  \synthslantbox@space@codesequence#1\space\@nil
  \ifx\relax#2%
    \relax
  \else
    \space
    \synthslantbox@literal@space#2\@nil
  \fi
}

```

`\synthslantbox`

We define two completely different implementations depending on the request for fontspec doing the slanting or any other package.

Macro 1: Immediately call the fontspec-specific macro. Bypass the hierarchy needed for the other slant engines.

```
\ifnum\synthslant@engine=4% fontspec
  \newrobustcmd*{\synthslantbox}[2]{%
```

The following (expanding) definition is only here for the compatibility of both branches.

```
    \edef\synthslant@slant@value{#1}
    \synthslantbox@fontspect@shear@box{\synthslant@slant@value}
                                      {#2}%
  }
```

Macro 2: This is the firestarter for the processing of all different kinds breakpoints until we reach unbreakable chunks to be passed on to the selected slant engine.

Normally, a user wants to call `\textsynthslant` or `\textsynthupright-italic`, however LaTeX wizards may have other ideas.

```
\else
  \newrobustcmd*{\synthslantbox}[2]{%
    \edef\synthslant@slant@value{#1}%
```

The literal space in front of `\@nil` is crucial to initiate the chain of macro calls.

```
    \expandafter\synthslantbox@literal@space#2 \@nil
  }
\fi

```

`box@right@slant@correction`

This is a simple yet surprisingly effective heuristic for slant correction on the right-hand side of the slanted text. The value `\synthslant` is sin α, where α is the slant angle; see Equ. 1 on p. 4. Multiplied with the ex height of the current font, `\fontdimen5`, this is a good approximation of the necessary slant correction.

```
\newcommand*{\synthslantbox@right@slant@correction}{%
  \dimen0=\fontdimen5\font
  \kern\synthslant\dimen0\relax
}

```

`\slantcontext` If we have microtype support we enter the context defined by this macro in `slantenvironment`.

```
\newcommand*{\slantcontext}{tracking=synthslant}

```

`slantenvironment` (*env.*) We use this environment as a pair of hooks that are called right before and right after the actual slanting code runs. The default sets up an upright type shape before and adds some italic correction after slanting.

```
\NewDocumentEnvironment{slantenvironment}{}
  {\upshape
   \ifcsdef{microtypecontext}
           {\expandafter\microtypecontext
            \expandafter{\slantcontext}}
           {}}
  {\ifcsdef{endmicrotypecontext}
           {\endmicrotypecontext}
           {}%
   \synthslantbox@right@slant@correction}

```

`\textsynthslant` User-level macro to slant some text.

```
\NewDocumentCommand{\textsynthslant}{m}
  {\ifmmode
     \synthslantbox{\synthslant}{#1}%
   \else
     {\slantenvironment
      \synthslantbox{\synthslant}{#1}%
      \endslantenvironment}%
   \fi}

```

`@right@negslant@correction` We could play the same trick here as in `\synthslantbox@right@slant@correction` and use `\synthnegslant` instead of `\synthslant`. But our experiments show no need for a correction. Anyhow, this macro may be convenient to override someday.

```
\newcommand*{\synthslantbox@right@negslant@correction}{}

```

`\negslantcontext` If we have microtype support we enter the context defined by this macro in `negslantenvironment`.

```
\newcommand*{\negslantcontext}{tracking=synthnegslant}

```

`negslantenvironment` (*env.*) We use this environment as a pair of hooks that are called right before and right after the actual unslanting code runs.

The default sets up an italic shape before unslanting and adds some negative italic correction after unslanting.

```
\NewDocumentEnvironment{negslantenvironment}{}
  {\itshape
   \ifcsdef{microtypecontext}
           {\expandafter\microtypecontext
            \expandafter{\negslantcontext}}
           {}}
  {\ifcsdef{endmicrotypecontext}
           {\endmicrotypecontext}
           {}%
   \synthslantbox@right@negslant@correction}

```

`\textsynthuprightitalic` User-level macro to unslant some italic or oblique text.

```
\NewDocumentCommand{\textsynthuprightitalic}{m}
  {\ifmmode
     \synthslantbox{\synthnegslant}{#1}%
   \else
     {\negslantenvironment
      \synthslantbox{\synthnegslant}{#1}%
      \endnegslantenvironment}%
   \fi}

```

Change History

v0.1
- General: Initial version. i

v0.1a
- General: Add missing dependency on etoolbox. Fix suggested by `mbertucci47`. 22

v0.1b
- `\synthslantbox`: Replace `\relax` with `\@nil` as end marker. 29
- `\synthslantbox@hard@hyphen`: Replace `\relax` with `\@nil` as end marker. 28
- `\synthslantbox@literal@space`: Replace `\relax` with `\@nil` as end marker. 29
- `\synthslantbox@soft@hyphen`: Replace `\relax` with `\@nil` as end marker. 28

v0.2
- `\synthslantbox@discretionary`: New macro that allows parsing `\discretionary` primitives. 27
- `\synthslantbox@literal@space`: Rename from `\synthslantbox@space` to distinguish from new macro `\synthslantbox@space@codesequence`. 29
- `\synthslantbox@soft@hyphen`: Also slant discretionary hyphens at the end of a line. 28
- `\synthslantbox@space@codesequence`: New macro to handle explicit `\space`—in addition to literal spaces—in ⟨*text*⟩. 28

References

[1] American Mathematical Society and the LaTeX3 Project Team. *Package amsmath*. 2020, `https://ctan.org/pkg/amsmath`.

[2] Arseneau, Donald. *The relsize package*. 2013, `https://ctan.org/pkg/relsize`.

[3] Bienz, Tim and Richard Cohn. *Portable Document Format Reference Manual*. Addison-Wesley Publishing Company, Reading/MA, 1993, `https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/pdfreference1.0.pdf`.

[4] Bezos, Javier. *Package babel*. 2021, `https://ctan.org/pkg/babel`. The original author of babel was J. L. Braams.

[5] Bringhurst, Robert. *The Elements of Typographic Style*. 3rd ed. Hartley & Marks, Vancouver/BC, Canada, 2004.

[6] Burke, Christopher. *Paul Renner: The Art of Typography*. Princeton Architectural Press, 1998.

[7] Carlisle, David. *Shear Transform a Box*. 2013-12-7, `https://tex.stackexchange.com/questions/63179/shear-transform-a-box/63188`.

[8] Dvipdfmx Project Team, ed. ***dvipdfmx***. 2020, `https://ctan.org/pkg/dvipdfmx`.

[9] Jeffrey, Alan, Rowland McDonnell, and Lars Hellström. *Package fontinst*. 2009, `https://ctan.org/pkg/fontinst`.

[10] Klingspor Museum. *Venus*. 2022, `https://www.klingspor-museum.de/KlingsporKuenstler/Schriftfamilien/Venus.pdf`.

[11] LaTeX3 Project Team, ed. *LaTeX 2$_\varepsilon$ font selection*. 2023, `https://www.latex-project.org/help/documentation/fntguide.pdf`.

[12] LaTeX3 Project, *The LaTeX3 Interfaces*. 2024, `https://texdoc.org/serve/interface3/0`.

[13] latexref.xyz. *LaTeX 2$_\varepsilon$: An unofficial reference manual*. 2023, `https://latexref.xyz/dev/latex2e.pdf`.

[14] Middendorp, Jan. *Shaping Text*. bis publishers, Amsterdam, 2014.

[15] Penninga, Marc. *fonttools*. 2025, `https://ctan.org/tex-archive/fonts/utilities/fontools`.

[16] Robertson, Will *Package fontspec*. 2024, `https://ctan.org/tex-archive/macros/unicodetex/latex/fontspec`.

[17] Rokicki, Tomas. ***dvips***. 2022, `https://tug.org/texlive/Contents/live/texmf-dist/doc/dvips/dvips.pdf`.

[18] SCHLICHT, ROBERT. *Package microtype*. 2020, `https://ctan.org/pkg/microtype`.

[19] SCHRÖDER, MARTIN. *pdftex 1.40*. 2007, `https://tug.org/mail-archives/pdftex/2007-January/006910.html`.

[20] SPIEL, CHRIS. *Package typog*. 2024, `https://ctan.org/pkg/typog`.

[21] STAMM, PHILIPP. *Schrifttypen – Verstehen Kombinieren: Schriftmischung als Reiz in der Typografie*. Birkhäuser, Basel, 2020.

[22] STRIZVER, ILENE. *Type rules!: the designer's guide to professional typography*, 4th ed. John Wiley & Sons, Hoboken/NJ, 2014.

[23] THÀNH, HAN THE. *The pdfTEX user manual*. Baskerville, 8(1), 9–14 (1998), `http://uk-tug-archive.tug.org/wp-installed-content/uploads/2008/12/81.pdf`.

[24] THÀNH, HAN THE et al. *pdfTEX*. 2023, `http://mirrors.ctan.org/systems/doc/pdftex/manual/pdftex-a.pdf`.

[25] WILLIAMS, GEORGE et al. *FontForge*. `https://fontforge.org/en-US/`.

Index

Numbers written in italic style refer to definitions; numbers in regular style refer to pages where an entry is used. References to code lines are prefixed with ‘ℓ’. Again, italic style refers to definitions and regular style to lines where the code is used.