

dvidx: a T_EX DVI driver for OpenDX

J. P. Hagon
Physics Centre
School of Natural Sciences
The University
Newcastle upon Tyne
NE1 7RU
United Kingdom*

In this manual, we will describe how to use dvidx and its associated scripts and macros to produce high-quality annotation in OpenDX. It is assumed the reader has a basic familiarity with both OpenDX and T_EX.

I. THE dvidx SCRIPT

dvidx is a T_EX dvi driver program similar to dvips etc. It takes a dvi file as input and generates an OpenDX object. This object contains the correctly scaled and positioned characters from the OpenDX fonts converted from outline originals. It understands dvips colour specials and can output in two different OpenDX formats: a *compact* form which consists of external references to OpenDX fonts; an *inclusive* format in which all the relevant data from the external font files is included in the output. dvidx can be used standalone to produce OpenDX output if desired. Multiple pages are handled by collecting individual pages in an OpenDX *Group* object.

dvidx is a perl script, written using two clever perl packages written by Jan Pazdziora — `Font::TFM` and `TeX::DVI::Parse`⁴. dvidx works roughly as follows:

1. First, run `dvicopy`⁶ on the original dvi input to translate all the virtual font references to base fonts.
2. Parse the `dvicopy` output with `TeX::DVI::Parse`.
3. For each font encountered, obtain the appropriate metrics from the T_EX font metric (`tfm`) file using `Font::TFM`.
4. Map the base font to a raw OpenDX font and extract the appropriate characters.
5. Position and scale the character via an OpenDX rotation/translation operation (this is an *XForm* transformation object in OpenDX jargon).

A. Features and Limitations of dvidx

In this section we summarize the important features and limitations of dvidx:

- dvidx cannot read the packed font (PK) files traditionally used by dvi drivers and usually created (indirectly) from METAFONT source files. There is no reason in principle why it could not be made to use such files but the approach described here produces higher quality and is much easier to implement.
- Instead, dvidx makes use of *native* OpenDX fonts. You *must* have OpenDX versions of all fonts used in your T_EX document. Most outline fonts (PostScript, TrueType, OpenType) can be converted to OpenDX format

using the author's `font2dx` translator so this should not be a problem in most cases. However, it does mean that METAFONT sources which have not been converted to outline form cannot be rendered using dvidx.

- This, and other limitations, mean that dvidx is unlikely ever to comply with the stringent requirements of the *TUG DVI Driver Standards Committee Level 0* standard³.
- dvidx uses the *scaled point* (sp) as its internal unit and unlike most DVI drivers, does not rescale its output — the output coordinates are also in scaled points. Since a scaled point is defined such that $65536\text{sp}=1\text{pt}$ and there are 72.27pt in 1 in, the numbers in the output are invariably *very* large. Scaling is achieved within OpenDX by wrapping suitable scripts and macros around dvidx as we shall illustrate later.
- dvidx currently works *only* with OpenDX fonts in *text* format rather than the more compact binary format. This is a limitation that may be addressed in future. Also, dvidx requires that characters with a zero width have a well-defined height so that scaling ratios can be calculated. This is a problem because OpenDX fonts generally do not define character heights (just widths). The author's font conversion utility, `font2dx`, produces OpenDX fonts that have an additional *height attribute*. dvidx will look for this attribute whenever it encounters zero width characters.
- dvidx understands dvips colour specials. Hence, characters can be individually coloured within L^AT_EX using, say, the `color` package. It is hoped that future releases of dvidx will also enable OpenDX objects to be included in the output via a `\special` directive.
- dvidx is fairly slow. Formatting large amounts of data (e.g. a whole page of text) may take several minutes.

B. Running dvidx

The general form of a dvidx command is:

```
dvidx [OPTION]... FILENAME(.dvi)
```

where FILENAME is a T_EX DVI file. At present, the following options are available:

`--filter` Run as a filter — i.e. send output to standard output rather than to a file. By default `dvidx` sends its output to a file whose name has the same prefix as the input DVI file and has an extension `'.dx'`.

`--nocopy` Do not preprocess the file with `dvicopy`.

`--mode=<internal|external>` The default value, `internal`, creates an OpenDX data file with *all* data contained in the file. Selecting `external` results in an OpenDX data file with *external references* to the OpenDX font files. This results in very compact output files at a cost of greater subsequent processing time when reading the files. Also, files created this way are inherently less portable, since they rely on the font files being in a specific location.

`--colour=<white|black>` Colour used for text. The default is `white` since by default OpenDX images have a black background.

`--debug=<level>` Produce debugging information. `level` is an integer greater than zero.

`--help` Print command line help and then exit.

Note that `dvidx` outputs its coordinates in the $z = 0$ plane, with the baseline along the x axis (unless some transformation has been applied at a low level to change the coordinates in the DVI file).

C. The `dvidx` Map File

The mapping of raw \TeX font names to OpenDX font files is done via a map file similar to (but much less versatile than) the map file used by `dvips`. The map file contains two columns, the first column giving the name of the raw \TeX font and the second column giving the name of the corresponding OpenDX font file. It is possible that a single OpenDX font file may map to more than one raw \TeX font but not vice-versa. If a raw \TeX font maps to more than one OpenDX font file, then the *last* entry in the map file is the one that is used.

One of the features of the `dvips` map file is that one can reencode a PostScript font on the fly via a reencoding directive within the map file itself. For example, a re-encoding to `TeXBase1` is achieved within the map file with the following directive:

```
" TeXBase1Encoding ReEncodeFont " <8r.enc
```

where `8r.enc` is a file containing the appropriate PostScript encoding commands. This type of functionality could be added to the `dvidx` map file but in many cases would be redundant. This is because the OpenDX font file contains *exactly* 256 characters whereas Type 1 PostScript fonts generally contain ‘hidden’ glyphs that are not contained within the visible 256 character slots. It is often the case that the purpose of reencoding is actually to place many of these hidden glyphs in one of the visible slots.

Our solution to this problem is somewhat brute-force but effective. A font editor such as the freely-available `fontforge`⁵ can be used to re-encode the original outline font using the

required encoding file (such as `8r.enc`, for example). The re-encoded PostScript font is then converted to an OpenDX font using `font2dx` but given a different name to the original. The convention we use, is to append the string `-<enc>` to the OpenDX file name. This produces map file entries like:

```
ptmri8r      Times-Italic-8r.dx
tii          Times-Italic-8y.dx
```

whereas in the `dvips` map file we would have:

```
ptmri8r Times-Italic
" TeXBase1Encoding ReEncodeFont " <8r.enc
tii Times-Italic
" TeXnANSIEncoding ReEncodeFont " <texansi.enc
```

A similar brute-force approach can be used to deal with ‘slanted’ fonts created on the fly via a `dvips` map file entry such as:

```
ptmro8r Times-Roman " .167 SlantFont ...
```

It is hoped to incorporate this functionality into `dvidx` in a future release.

II. THE `latex2dx` SCRIPT

`latex2dx` is another perl script, designed as a wrapper around `dvidx`. Its purpose is to convert raw \LaTeX code directly to OpenDX data. It is specifically designed to be used in OpenDX macros via the `"!command"`² interface to the OpenDX `Import` module. `latex2dx` works roughly as follows:

1. Create a temporary \LaTeX file.
2. Run \LaTeX on this file to produce a temporary DVI file.
3. Run `dvidx` on the DVI file and send the result to standard output.
4. Delete all temporary files created.

`latex2dx` can be used as a template for similar scripts utilizing other \TeX formats such as `ConTeXt` and plain \TeX .

A. Running `latex2dx`

The general form of a `latex2dx` command is:

```
latex2dx [OPTION]
```

At present, the following options are available:

`--e=<LaTeX command string>` A string (in quotes) containing \LaTeX commands. These are the commands which appear *after* the preamble and *between* the `\begin{document}` and `\end{document}` directives which will be automatically wrapped around.

`--f=<LaTeX command file>` Similar, except that the argument is now an external *file* containing the *body* \LaTeX commands. This option takes priority in the case that both `--e` and `--f` are given.

`--preamble=<LaTeX preamble directives>` This option can be used to supply directives which would normally be placed in the *preamble*, e.g. `\usepackage` commands. It takes the form of a single string in quotes. Note that one preamble instruction is hardcoded in `latex2dx`, namely `\pagestyle{empty}`, which removes headers, footers and page numbers which are usually not needed in this context.

`--help` Print usage information and help.

III. L^AT_EX ANNOTATION MACROS

Two OpenDX macros — *LaTeXText* and *LaTeXCaption* — were developed as alternatives to the native *Text* and *Caption* modules. They each utilize the `latex2dx` script and take L^AT_EX commands as their main argument. The user may enter, optionally, a set of L^AT_EX *preamble* instructions if required. For example, `\usepackage{cmbright}` might be entered if the CM Bright fonts were required. The double backslash is not an error — it is an unfortunate consequence of the fact that backslash itself is treated as an escape character in OpenDX macro and module string arguments.



FIG. 1: *LaTeXText* and *LaTeXCaption* macros as they appear in the OpenDX Visual Programming Editor.

Within the OpenDX *Visual Programming Editor* (VPE), the macros appear as shown in figure 1. Both *LaTeXText* and *LaTeXCaption* convert the *XForm* objects output by `dvidx` into normal objects via the “apply transform” directive of the *Inquire* module.

A. The *LaTeXText* Macro

LaTeXText positions text anywhere in 3D with any orientation. Optionally, it can also add an *extrusion* perpendicular to the plane of the text, giving the text a *thickness*. It takes the following arguments:

`latex_string` A string of L^AT_EX commands. This is passed internally to the `--e` option of `latex2dx`.

`height` Height of text in user (world) coordinates. Note that this refers to the *total* height of the whole body of text, not the height of individual characters.

`position` The position vector of the *reference point* (see below) in user coordinates.

`baseline` Direction of the baseline expressed as a vector.

`angle` Euler-type angle specifying a rotation about the baseline axis.

`preamble` A string of L^AT_EX preamble directives.

`extrusion` A scalar defining the extrusion length in user coordinates. A number ≤ 0 produces no extrusion — this is the default. The extrusion is applied symmetrically, so for an extrusion length, ℓ , the text will be extruded by an amount $\pm\ell/2$ above and below the text plane.

`reference` An integer (1–9) specifying the reference point on the formatted text to be used for positioning. (1) refers to the bottom left (the default); (2) is bottom centre; (3) is bottom right etc. up to (9) which is top right.

There are four outputs:

`text` Complete object including extrusions and all (top and bottom) surfaces.

`extrusion` No upper or lower surfaces — just the extruded sides.

`top_surface` The top surface only.

`bottom_surface` The bottom surface only.

The four outputs allow the upper/lower surfaces and extruded sides to be handled differently within OpenDX. For example, the upper and lower surfaces can be given different colours.

There are several coordinate transformations applied by *LaTeXText*. The order in which the transformations are applied is as follows:

1. The raw output from `dvidx` (remember this is in scaled points) has its origin translated according to the value of the `reference` parameter.
2. A linear scaling of the coordinates is then applied according to the `height` parameter.
3. The text is then rotated according to the value of the `baseline` vector.
4. The text is then rotated about the new baseline direction according to the value of the `angle` parameter. The actual rotation line used passes through the reference point.
5. The scaled, translated and rotated object then has its transformed origin (reference point) placed at the point specified by the `position` parameter.

Some of these transformations are illustrated in figure 2.

B. The *LaTeXCaption* Macro

LaTeXCaption produces a so-called *screen* object. This is an object which is essentially 2D in nature and usually remains in a fixed position on screen.

LaTeXCaption relies on a *ScreenObject* module which is not part of the core OpenDX system. This module can be obtained from *Visualization and Imagery Solutions, Inc.*¹. You must have this module (or an equivalent) enabled in order to use *LaTeXCaption*.

LaTeXCaption has just a single output and the following inputs:

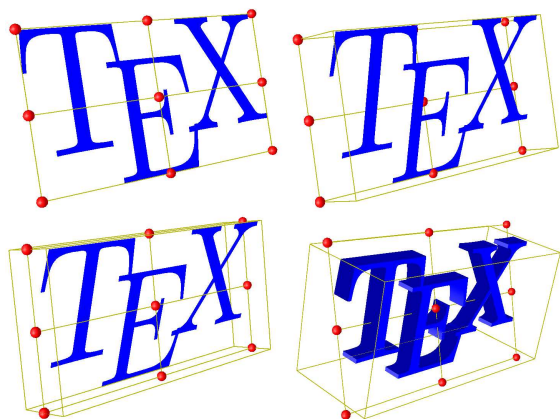


FIG. 2: Geometry of a \LaTeX object generated by *LaTeXText*. (a) Euler angle, $\vartheta = 0^\circ$. (b) $\vartheta = 20^\circ$, ref. point = 1. (c) $\vartheta = 20^\circ$, ref. point = 5. (d) Extrusion of (c). The dots show the 9 reference points.

`latex_string` A string of \LaTeX commands.

`coords` A string specifying the type of coordinates used to position the text on the screen display — “viewport”, “pixel” or “world”. Using “world” coordinates means that the text string will be attached to a particular point in user coordinate space but retain the same orientation with respect to the viewing camera. “viewport” and “pixel” both refer to screen coordinates. Both define the lower left corner of the screen as the origin. “viewport” defines the upper right of the screen as the point $[1, 1, 0]$ (the z coordinate is ignored). “pixel” defines the upper right position as $[w, h, 0]$, where (w, h) refers to the pixel dimensions of the display window in OpenDX.

`direction` Direction of the baseline expressed as a vector.

`position` The screen position of the text reference point. The interpretation of this parameter depends on the value of the `coords` parameter. This should always be entered as a 3D vector even if “viewport” or “pixel” were used (in those cases, the z value is ignored).

`height` Total height of the body of text in pixels.

`preamble` A string of \LaTeX commands.

`reference` An integer (1–9) specifying the reference point on the formatted text to be used for positioning. (1) refers to the bottom left (the default); (2) is bottom centre; (3) is bottom right etc. up to (9) which is top right.

C. *LaTeXFileText* and *LaTeXFileCaption*

If there is a lot of text to be typeset, it is more convenient to supply a file containing that text rather than type in the text as a macro argument. Two additional macros have been created for this purpose — *LaTeXFileText* and *LaTeXFileCaption*.

The only difference between these macros and *LaTeXText/LaTeXCaption* is that the `latex_string` parameter has been replaced by `latex_file`. One advantage of using these macros (in addition to their primary purpose) is that the backslash characters do not need to be doubled up.

D. *TextAlias*

TextAlias is designed to smooth the text output from the above macros. It takes the following inputs:

`object` The input object — this will often come directly from the output of one of the above macros.

`alias_factor` A scalar between 0 and 1. This parameter determines the degree of anti-aliasing (smoothing). The default is 0.5, which is usually the optimum value.

`background` An RGB vector specifying the background colour, with $[1, 1, 1]$ representing pure white.

TextAlias works by adding a shaded line, one pixel wide, around the boundary of each character. If the characters are drawn at low resolution (as is often the case for screen display), then this tends to thicken and smooth the characters enough to make them readable, even at small sizes. If the characters are drawn at high resolution, the line is barely noticeable as it is just one pixel wide. This is appropriate because at high resolutions, anti-aliasing is normally not required.

Note: *TextAlias* should be used only if the background colour behind the text is *uniform*.

IV. ACKNOWLEDGEMENTS

I am grateful to *Visualization and Imagery Solutions Inc.* for providing a copy of their *ScreenObject* module during the development of this software.

* Electronic address: Jerry.Hagon@ncl.ac.uk

¹ VIS Inc. <http://www.vizsolutions.com>.

² IBM Visualization Data Explorer User's Reference. <http://opendx.npaci.edu/docs/html/refguide.htm>, 1999–2004.

³ TUG DVI Driver Standards Committee. *The DVI Driver Standard, Level 0*. <http://www.ctan.org/tex-archive/dviware/driv-standard>, 1995.

⁴ J. Pazdziora. `TEX::DVI::PARSE` and `Font::TFM`. <http://www.cpan.org>, 2004.

⁵ G. Williams. *Fontforge*. <http://fontforge.sourceforge.net>, 2004.

⁶ `dvicopy` is a program which is routinely available as part of all modern \TeX implementations. Its sole purpose is to expand virtual font definitions. This is useful in cases where a dvi driver doesn't understand virtual fonts.