The PracTEX Journal, 2007, No. 3
Article revision 2007/08/17

TpX — LaTeX-oriented graphical editor

Alexander Tsyplakov

Email tsy@academ.org
Website http://www.nsu.ru/ef/tsy/about_en.htm
Address Novosibirsk, Russia

Abstract This article describes TpX, a lightweight, easy-to-use graphical editor for Windows platform, presents guidelines for its use and discusses some features and limitations.

1 Introduction

TpX is a lightweight, easy-to-use graphical editor for Windows[1] [4]. The tool allows users to draw and include their drawings in LaTeX files. In addition, the tool can be used as a stand-alone editor for vector graphics.[2]

The tool supports both the (pdf)LaTeX→DVI→PS and pdfLaTeX→PDF ways of producing documents. The choice between two regimes is governed by the `ifpdf` package. Thus the same drawing can be used in a LaTeX document in both regimes.

TpX can import EMF/WMF pictures created by other Windows applications, including many applications producing scientific graphs. It also can import simple SVG pictures. So TpX can be used as an EMF-to-any and a SVG-to-any converter.

2 How it came to be

I am engaged in writing a large microeconomics textbook (more than 1000 pages). It is a long-term project of two co-authors and myself which has continued for

1. See below about the cross-platform version of TpX which is under development.
2. Vector graphics use geometric shapes based on analytical formulas to represent graphical information, which is different to bitmap or raster graphics that represent arrays of pixels.

several years and is not finished yet. At one stage a decision was made to switch from Word to LaTeX because of the high typesetting quality of LaTeX and the preferences of a co-author. Consequently, there were two main problems. The first one was to convert the documents from Word to LaTeX. The second one was to include a large number of graphical illustrations into the resulting LaTeX document in a form which allows further editing.

The graphic edition workflow using MS Word was relatively fast and easy. Double click the picture, edit with the mouse, close and here you are. In LaTeX, it is a little bit different because it has no built-in WYSIWYG graphical capabilities like MS Word. Thus, I was looking for some tool which would allow me to edit illustrations in a similarly rapid manner with the mouse.

It turned out that there are many different choices of vector graphics programs around. Some of them I knew when starting TpX (TeXCAD [7], jPicEdt [8] and many others). Some alternatives came to my sight only recently (Ipe [9] and LaTeXDraw [10]).

There exist several drawing programs based on the standard LaTeX `picture` environment (sometimes enhanced by additional packages). This environment produces what can be called pseudo-graphics. By default, the result is very primitive and is not usually suitable for a published book. (However, it is quite portable, as it uses only LaTeX fonts for rendering graphics and do not need special drivers.)

At the other extreme there is Inkscape [6]. It is a powerful vector graphics program based on the SVG format, but it is large and not at all LaTeX-friendly.

jPicEdt is somewhere between these two extremes. As well as the LaTeX `picture` environment enhanced by the `eepics` package it can also output PSTricks code, which is suitable for use in a published book. However, jPicEdt is based on Java, which is not very convenient (for example, it precludes use of the program on computers which do not have Java installed). And most importantly, it is not possible to use graphics produced by jPicEdt directly with pdfLaTeX to get PDF.

Thus, I decided to write my own tool. The following principles were behind the development of TpX:

- tight integration with LaTeX (for example, use of LaTeX for rendering text labels),
- support for multiple LaTeX-related formats (including formats compatible with the pdfLaTeX→PDF way of producing documents),

– small size and fast loading with a small memory footprint,

– usability,

– simple and open format for storing drawings,

– easy import and export from/to important formats of vector graphics.

The development of TpX was very helpful for writing of the above-mentioned textbook. TpX writes graphics in various formats and is very usable. After all, converting the illustrations from MS Word to TpX[3] has provided advantages for preparing the textbook for publication because MS Word encapsulates pictures in the document. This is a problem when you need to unify their style and appearance. I have used TpX for many other purposes. For example, I have included statistical diagrams into beamer presentation or printed pictures for my little daughter to colour.

3 The direct use of TpX

The TpX graphical user interface is similar to other Windows programs. Figure 1 shows a TpX screenshot. TpX allows you to do the most common vector drawing tasks (like creating, moving, copying, reshaping of graphical objects, changing colour and so on).

TpX is not fully WYSIWYG. The screen representation of a drawing is somewhat different from the resulting graphics included into LaTeX document. For example, mathematical formulas would be seen only in the LaTeX document. Having a system font with a rich set of symbols one can make text labels more similar to LaTeX output by including `&#xHHHH;` code into the label text where `HHHH` is hexadecimal unicode representation of a symbol.[4] Please, observe Greek letters and degree symbols in Figure 1.

A TpX drawing is stored in a file (with extension .TpX), which is intended for inclusion into a parent LaTeX document with the `\input` command. Figure 2 is the output from TpX drawing. It was included into this PracTeX article using

3. The illustrations were saved as RTF using a Visual Basic script. Then a Python script was used to convert RTF to some XML representation and eventually to TpX.

4. This is important when exporting TpX drawings to non-LaTeX formats like SVG or EMF.

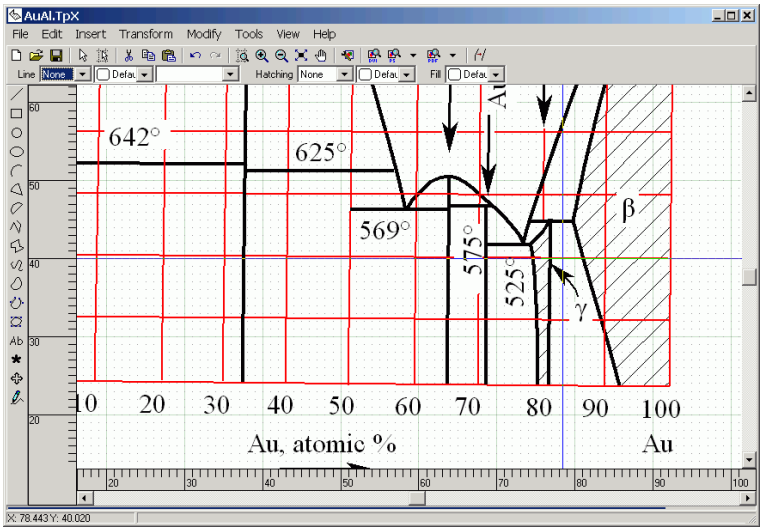


Figure 1: TpX screenshot

`\input{AuAl.TpX}`. Note that it uses the same URW Palladio font family as the article itself.

Let me describe the workflow to use TpX for creating a drawing from scratch. Suppose that one is going to create a drawing with the filename "Foo.TpX" in subdirectory `pics/`.

– In the text editor used to edit the parent LaTeX document add the following line to the document:

  ```
  \input{pics/Foo.TpX}.
  ```

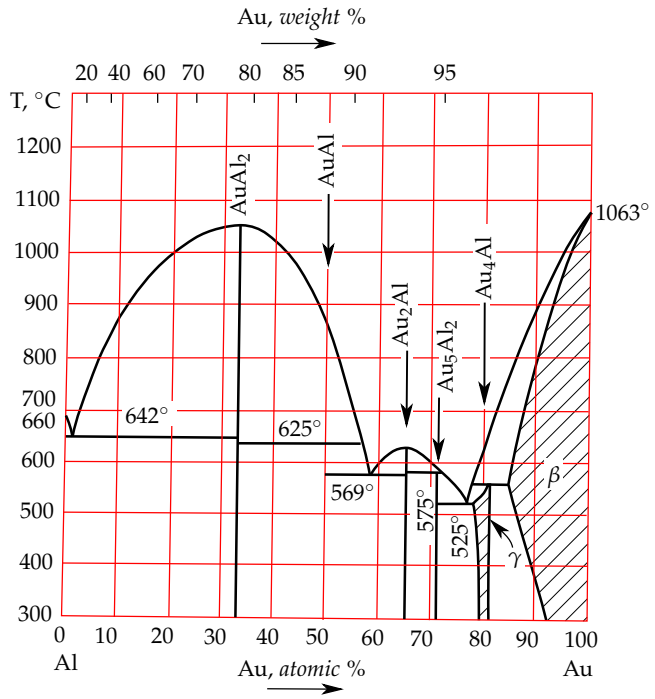
  Save the document.

– Move the cursor to the added line and press the hot key.[5] The TpX program will be started with the following command-line parameters:

  ```
  -i<filename> -l<line number>
  ```

  where <filename> is the name of the parent LaTeX document and <line number> is the current line number. TpX will scan the LaTeX document and find

5. I assume that there is already a hot key for running TpX. This requires an editor which can have a hot key for running an external program and which is able to pass the current line number as a command-line parameter to an external program. The TpX distribution includes support macros for the popular WinEdt editor.



Aluminium-gold compounds diagram

Figure 2: TpX output

the case of `\input{<filename>.TpX}` which is closest to the given line (that is, `\input{pics/Foo.TpX}`). This will open Foo.TpX in TpX.

– Fill the newly created drawing with graphical content and save it in the `pics/` subdirectory.

– Close TpX and return to the editor.

The most time-consuming part of the procedure is preparation of the drawing. The other operations are very fast. TpX in combination with a smart text editor saves time which is typically spent on finding a graphic on the disk and opening it in a graphical editor. Time savings are even more pronounced if only some minor corrections are needed for an existing drawing.

The program's own data are put into TEX comments (lines starting with "%") so that the drawing could be loaded into TpX and edited again. (The internal TpX format is based on XML and can be understood and edited easily). These

data are not seen by the LaTeX program. After the comments TpX writes normal commands that are seen by LaTeX. Depending on the required output format this code would include direct drawing commands (like commands from the `picture` environment, TikZ commands) or an `\includegraphics` link to an external file created by the program.

Code inside a typical TpX file looks like this:

```
%<TpX v="4" ArrowsSize="0.7" ... ...
%  <polygon fill="whitesmoke" ... ...
%  ... ... drawing data ... ...
%</TpX>
\begin{figure}
\centering \ifpdf
  ... ... code for PDF ... ...
\else
  ... ... code for DVI->PS ... ...
\fi
\end{figure}
```

The output format is chosen separately for PDF and DVI→PS. TpX provides several output formats:

- Formats for which graphics are drawn using direct LaTeX code: LaTeX `picture` environment, PSTricks [13], PGF and TikZ [14].
- Formats for which graphics from an external file generated by TpX is included into the document with the help of the `graphics` package [5]: EPS, PDF, MetaPost EPS [15] and PNG.[6]

EPS and PDF are TpX defaults for the DVI→PS and PDF regimes respectively. With these output formats LaTeX text labels are overlaid above included EPS (PDF) graphics using the LaTeX `picture` environment.[7] This allows using the same font for text labels as is used in the parent document. See TpX help for more information about output formats.

6. Please note that PNG is, as opposed to all other formats mentioned, a bitmap (or raster) format rather than a vector format.
7. This is similar to how the `overpic` package [11] works.

4 Porting graphics into LaTeX using TpX

It is common for a Windows program which produces some kind of vector graphics to allow copying a metafile image (EMF) to the clipboard and/or exporting it as an EMF file. TpX can import this (though not 100% correctly) for subsequent inclusion into LaTeX document. Old-style Windows Metafiles (WMF) are imported by converting them to EMF. In most cases the result of importing is nice, though often the imported picture needs some manual editing. TpX can also import SVG images. Note that the SVG format is extremely rich (see [1]), so TpX can understand only a basic subset of it.

For example, in order to include an Excel diagram into a LaTeX document one can do the following:

- Copy the Excel diagram to the Windows clipboard. This copies also an EMF image representing the diagram.
- Use TpX to capture the EMF image from the clipboard ("Tools" > "Capture EMP") and save it to an EMF file.
- Import this EMF file into TpX.
- Edit the picture. Sometimes scaling of the picture's physical size is needed (for example, in order to satisfy page size constraints).

An example of such a use is given in Figure 3.

Of course, there are other ways to include external graphics into LaTeX document. The most popular procedure is printing graphics to an EPS file with a PostScript printer driver. However, the TpX way gives important advantages:

- TpX can add TeX formulas to graphs.[8]
- It is possible to edit imported images and correct their deficiencies. For example, TpX can help to shift misplaced text labels.
- The graph imported through TpX uses the same fonts as LaTeX document itself, which adds to the typographic quality of the document. (For example, Figure 3 uses the sans serif font which is implied by the PracTeX article style[9]).

8. Well-known alternatives are `overpic` [11] and `psfrag` [12] packages which allow adding text labels to included EPS files by means of LaTeX commands.
9. The graph was included with `{\sffamily\input{images/ExcelGraph.TpX}}`.

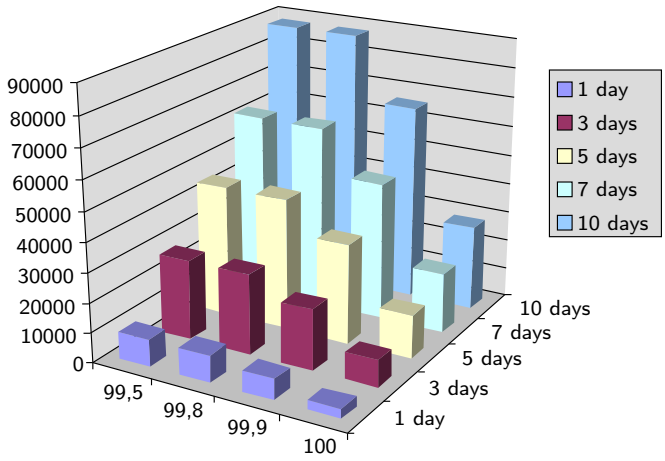


Figure 3: Excel graph imported into TpX

To see what the TpX capabilities and limitations are in this respect, have a look at 'Demonstration of TpX import capabilities' [2]. This document provides examples of import from many well-known as well as lesser known Windows programs.

5 Drawing with the mouse vs. drawing by code

Some people prefer to draw visually and interactively using the mouse while others prefer to draw logically (programmatically) by providing coordinates of graphic objects directly (or using geometric concepts like line crossing). TpX is intended for people of the first type. TpX is most useful when one has some visual scheme in one's head and wants to implement this scheme quickly as a sketch.

Figure 4 shows the typical task which is performed using the mouse and which requires the human eye to control the accurateness. This is unlike a clever graphical programming language which could make it possible to say: "Draw a line of this length through these two points".

Note that TpX's viewport can be zoomed easily using the mouse wheel or hot keys. This allows accurate visual placement of lines for most common purposes. The human eye cannot distinguish small nuances in the positions of graphical

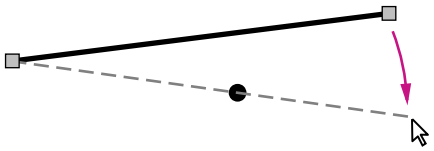


Figure 4: Passing a line through a point

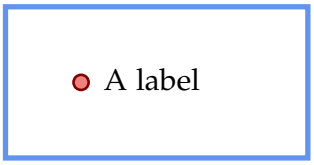


Figure 5: Simple drawing for TikZ example

objects anyway so it is not usually necessary to place them 100%-exactly.

For more accurate placement with the mouse additional features can be used: snap to grid and angular snap. Snap to grid mode allows to restrict oneself to "round" coordinates. Angular snap mode helps to restrict the angle when moving something with mouse (to a multiple of 45° in TpX).

The fact that TpX is mouse-driven does not mean that it is impossible to specify exact coordinates in TpX. Open the properties window of a graphical object, then press the "Points" button. This will open a table editor for editing coordinates. Consider, for example, generating coordinates in a spreadsheet application and then pasting them into this table editor.

An interesting point is that there is actually no antagonism between using TpX and drawing by code. TpX can be used to generate PSTricks, METAPOST or TikZ code. Generated code can then be included into LaTeX documents and edited with the keyboard in a text editor together with the document.

Figure 5 shows a simple example drawing. When saved in a TikZ format it gives the following code:

```
\begin{tikzpicture}[x=1.00mm, y=1.00mm, inner xsep=0pt, inner ysep=-1.2pt]
\path[line width=0mm] (-2.00,8.00) rectangle +(44.00,24.00);
\definecolor{L}{rgb}{0.392,0.584,0.929}
\path[line width=0.70mm, draw=L] (0.00,10.00) rectangle
+(40.00,20.00);
\definecolor{L}{rgb}{0.502,0,0}
```

```
\definecolor{F}{rgb}{0.941,0.502,0.502}
\path[line width=0.35mm, draw=L, fill=F] (10.00,20.00) circle
(1.00mm);
\draw(13.00,20.00) node[anchor=west]{\fontsize{11.38}{13.66}\selectfont
A~label\strut};
\end{tikzpicture}%
```

Of course, this is not very suitable for logical editing as it is. One would prefer to do some further editing. First, it is better to replace `\fontsize...\selectfont` with something like `\large`. Second, one should prefer the use of logical commands for line width (like "very thin" or "ultra thick") to the direct declaration of physical size (like `line width=2mm`). See [3] for more on TikZ.

TpX's own data are in XML format and are written inside TEX comments as plain text. This makes it straightforward to program a script to generate a TpX drawing. I wrote a simple Python module, TpXpy, which can help generate TpX drawings. Here is a sample TpXpy code:

```
pic = TpXpic()
pic.addPolyline(((10,0), (30,-20), (15,-30)))
pic.fill('mintcream')
pic.lw(1.5)
for i in range(5):
  pic.addLine(i*10 + 50,-0.7,i*10 + 50,0.7)
  pic.li('dash')
pic.SaveToFile('Foo.TpX')
```

Any other scripting language would be as good for the task as Python. Of course, any of the formats produced by TpX can be generated using a script. However, generating TpX code has some advantages. First, it is possible to obtain output in many different formats from the same source. Second, the result can be further edited by mouse if needed.

6 Ideas for automating editing tasks

If a document contains just a couple of illustrations then modifying them is not a terrible problem. However, a document with dozens of illustrations requires special attention. For example, one can come upon a need for uniform scaling of the illustrations to fit a different paper size or changing colours throughout.

This section gives some ideas, from my own experience, to help you maintain illustrations in such documents.

- One can write a script gathering all `\input{<filename>.TpX}` commands in a temporary document. Then one can quickly revise all illustrations one by one switching between LaTeX editor, DVI (PDF) previewer and TpX.
- In order to make some uniform correction in a series of drawings all one needs to do is to use a simple search and replace utility. For example, one can change output format for DVI from PGF to PSTricks by replacing all cases of ␣TeXFormat="pgf" by ␣TeXFormat="pstricks".[10] The drawing has to be refreshed to reflect changes. This can be done by running the following command:

  ```
  TpX.exe -f<filename>.TpX -o
  ```

  A renew_all.py script which is distributed with TpX shows how one can refresh all drawings in a specified directory.
- A more advanced way of automatically correcting a series of drawings is to use a scripting language with an XML DOM library. As was already mentioned, the TpX format for storing drawings is based on XML. To read the corresponding XML document one has to read all lines staring with "%" from the TpX file and strip the "%" symbol from each line. This will give an XML document. The XML document is then converted to a DOM tree. After editing the DOM representation one has to go this way backwards by first getting XML document and then prepending each line by the "%" symbol. Finally, the resulting TpX file has to be refreshed.

7 Further Improvements

Currently TpX is a one-man project. However it is placed at SourceForge[11] under the GNU Public License and anybody with sufficient knowledge of Object Pascal is invited to participate.

10. Note the space char before TeXFormat; it is needed to distinguish TeXFormat and PdfTeXFormat.
11. http://sourceforge.net/.

Although TpX was originally created for the Windows platform using Delphi I recently reshaped it for Lazarus, a cross-platform clone of Delphi. The Lazarus variant of TpX is at alpha stage and lacks some functionality of the Delphi variant (like export of PNG and EMF images). Also it was not yet fully adapted for other platforms (Linux or OS X). In the future, this will hopefully result in a usable product.

I conclude this section listing some of the planned enhancements from my todo file: · group/ungroup; · grouping objects into compound paths (to get disjoint shapes and shapes with holes); · bitmap object; · diagram object for creating simple plots; · adding custom graphical objects from a library; · additional properties for graphical objects (miter limit, line caps, etc.); · simplifying of Bezier paths; · layers; · codepages.

References

[1] W3C. Scalable Vector Graphics (SVG). XML Graphics for the Web.
http://www.w3.org/Graphics/SVG/

[2] Tsyplakov, Alexander. Demonstration of TpX import capabilities, January 31, 2006.
http://tpx.sourceforge.net/TpX-Demo.pdf

[3] Tantau, Till. The TikZ and pgf Packages. Manual for Version 1.01.
http://sourceforge.net/projects/pgf

[4] TpX.
http://tpx.sourceforge.net/, CTAN:graphics/tpx/

[5] `graphics` and `graphicx` packages.
CTAN:macros/latex/required/graphics/

[6] Inkscape.
http://www.inkscape.org/

[7] TeXCAD.
http://homepage.sunrise.ch/mysunrise/gdm/texcad.htm

[8] jPicEdt.
http://jpicedt.sourceforge.net/site/index.php,
CTAN:graphics/jpicedt/

[9] Ipe.
http://tclab.kaist.ac.kr/ipe/

[10] LaTeXDraw.
http://latexdraw.sourceforge.net/

[11] `overpic` package.
CTAN:macros/latex/contrib/overpic/

[12] `psfrag` package.
CTAN:macros/latex/contrib/psfrag/

[13] PSTricks.
http://tug.org/PSTricks/, CTAN:graphics/pstricks/

[14] PGF/TikZ.
http://sourceforge.net/projects/pgf/, CTAN:graphics/pgf/

[15] METAPOST.
http://sarovar.org/projects/metapost/, CTAN:graphics/metapost/