

Short introduction to PSTricks

Tobias Nahrung

February 22, 2005

1

This presentation was held in German at the T_EX-user-meeting Dresden (Germany) at June 12, 2004 (see also <http://www.carstenvogel.de>).

The slides are shown here in a scaled-down version with minor changes and some explanations are added.

PSTricks is well suited to draw simple vector graphics inside of L^AT_EX documents. The basic syntax of PSTricks is quite similar to that one of the `picture` environment native to L^AT_EX but the macros of PSTricks are in general much more comfortable and powerful.

Even if PSTricks is rather powerful it should be emphasised that every program should be used for the task it is designed for. For an example, if you want to construct complicated realistic three-dimensional scenes you should better use some ray tracer¹ (e. g. the freely available ray tracer `povray`).

¹... even if less complicated three-dimensional scenes can be visualized with the help of the PSTricksadd-on `vue3d`

1 Sources

- <http://www.tug.org/applications/PSTricks/>
Many, many examples. (Learning by doing.)
- <http://www.pstricks.de/>
Ditto.
- <http://www.pstricks.de/docs.phtml>
PSTricks user guide: as one PDF, PSTricks quick reference card
- Elke & Michael Niedermair, \LaTeX Praxisbuch, 2004, Franzis-Verlag,
(Studienausgabe für 20€)

2

The original documentation of PSTricks written by Timothy van Zandt (the author of the base packages of PSTricks) is 100 pages long. Documentations of add-ons for pstricks cover again hundreds of pages.

From that fact, it is clear that this short presentation can only cover very few aspects of the PSTricks package. So, the first slide gives some sources of informations about the PSTricks-package.

The intention of this presentation is to give an impression of what is possible with the basic PSTricks-packages and to encourage interested people to give PSTricks a try.

Contents

1 Sources	2
2 First example	7
3 Important tool: The grid	8
4 Setting options	9
5 Star versions of objects	10
6 Further basic geometric objects	11
7 Line ends 'Arrows'	12

8 File plots	13
9 Function plots (parametric)	14
10 Placing whatever, wherever	15
11 Clipping and scaling	16
12 Easy way to scale everything	17
13 Enrolling one's own path	18
14 Repetition (and rgbcolors)	19
15 Special coordinates (e.g. polar coordinates)	20

16 Special coordinates (postscript)	21
17 Example for the usage of (LA)T_EX-commands	22
18 Importing eps-files	23
19 Nodes and node connections	24
20 ‘Labeling’ node connections	25
21 More nodes and node connections	26
22 Node placement with psmatrix	27
23 Including postscript code in <code>\pscustom</code>	28

24 The corresponding postscript codes	29
25 ps4pdf: Preparing the L^AT_EX-file	30
26 ps4pdf: pstricks & pdflatex	31
27 Other nice stuff – fillstyle=gradient	32
28 Other nice stuff – Playing with Text	33

At first, some **geometric objects** of PSTricks are presented followed by some macros for the **modification** of these objects. This includes **function plots** and **embedded eps figures**.

Then, the power of the PSTricks-constructs called ‘**nodes**’ and ‘**node connections**’ are demonstrated with the help of some examples.

After that, the usage of **PDFL^AT_EX** for documents with PSTricks-pictures is discussed.

At last, two slides with **other nice stuff** of PSTricks are shown.

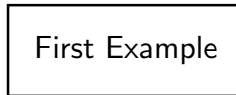
Pieces of **embedded Postscript fragments** throughout the presentation show the close relation of PSTricks with the Postscript language.

Some basic knowledge is advantageous when using PSTricks but many things are also possible without such knowledge.

2 First example

```
\documentclass{article}
\usepackage{pstricks}
\begin{document}
\begin{figure}
  \begin{pspicture}(4,5)
    \psframe(0.7,2)(3.3,3)
    \rput(2,2.5){First Example}
  \end{pspicture}
\end{figure}
\end{document}
```

pspictures can replace simple eps-figures.



7

To use PSTricks macros in L^AT_EX at least the style file `pstricks.sty` must be loaded. The basic PSTricks package has many more style files for various tasks. If you do not mind processing time then you can load all these files with the command `\usepackage{pst-all}` instead of `\usepackage{pstricks}`.

With the `\psframe` macro above the rectangle in the picture is drawn. PSTricks macros such as `\psframe` do not modify the current L^AT_EX position. If there is no extra space is reserved for the PSTricks objects then the following L^AT_EX Text overwrites the objects as it is demonstrated here.

The task of reserving space for the PSTricks pictures is accomplished by the `pspicture` environment. In the form above the coordinate pair (4,5) in `\begin{pspicture}(4,5)` stands for the width and the height of the PSTricks picture. By default lengths are measured in centimetres.

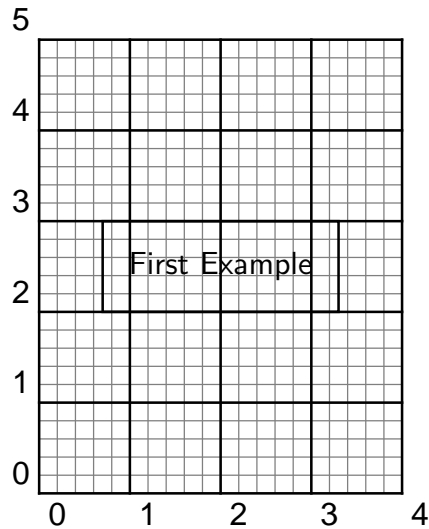
The pairs of numbers in the parentheses following the `\psframe` macro are the coordinates of the lower left and the upper right corners of the rectangle.

The most simple form of the `\rput` macro used above works very much like the `\put` macro of the L^AT_EX picture environment. Later we will see that the `\rput` macro is the much more powerful one of both.

3 Important tool: The grid

```
\begin{pspicture}(4,5)  
  \psgrid  
  ...  
\end{pspicture}
```

Globally deactivated via
`\let\psgrid\relax`
in the final version.



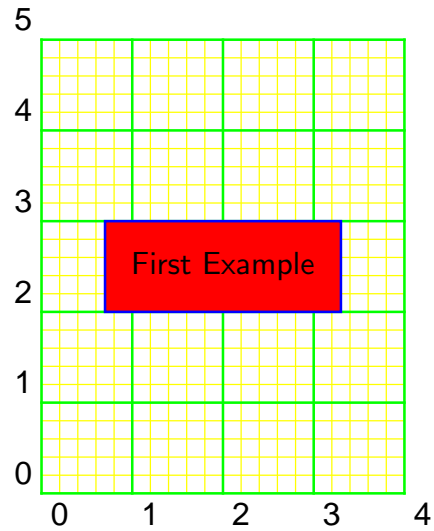
One important tool for drawing pictures with pstricks is the grid. It can be activated with the `\psgrid` macro. If no further arguments are given in the `\psgrid` command it produces a grid with width and height as determined by the size of the enclosing `pspicture`.

If one wants all other PSTricks elements in the `pspicture` drawn onto the grid one should specify `\psgrid` at first inside the `pspicture` environment as indicated above. On the other side, if one wants the grid drawn onto all the other stuff in the `pspicture` one should specify the grid as the last thing in the `pspicture` environment.

To remove all grids in the final document one can simply deactivate all the `\psgrid` commands in the document by the single command `\let\psgrid\relax` at the beginning of the document.

4 Setting options

```
\psset{gridcolor=green,  
       subgridcolor=yellow}  
\begin{pspicture}(4,5)  
...  
  \psframe[linecolor=blue,  
           fillcolor=red,  
           fillstyle=solid]  
           (0.7,2)(3.3,3)  
...  
\end{pspicture}
```



9

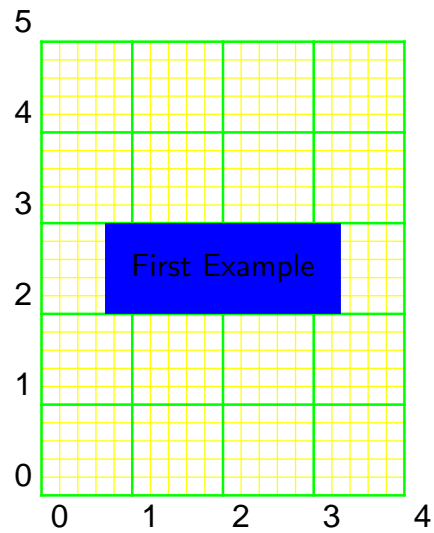
Many, many options can be set for PSTricks elements. There are two main ways to do so.

To set the options for all following PSTricks elements inside the scope of the current group one can use the `\psset` macro. The options are given as a comma separated list of key=value pairs. It is admissible to place `\psset` macros outside of `pspicture` environments.

For specifying options for a specific PSTricks element most PSTricks macros accept optional arguments in brackets as indicated on the slide.

5 Star versions of objects

```
\begin{pspicture}(4,5)
...
\psframe*[linecolor=blue,
          fillcolor=red]
          (0.7,2)(3.3,3)
...
\end{pspicture}
```

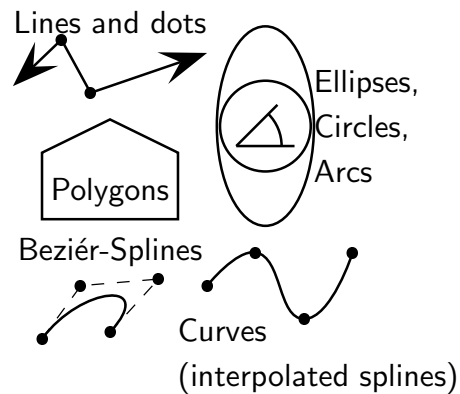


10

Often one needs just one evenly colorised area. For that purpose most PSTricks elements have a corresponding star version (Note the red star on the slide). The area of the star form of a PSTricks element is filled with the `linecolor` – regardless of the current setting of the option `fillcolor`.

6 Further basic geometric objects

`\psline`
`\psdots`
`\pspolygon`
`\pscircle`
`\psellipse`
`\psarc`
`\pscurve`
`\psbezier`



Exact syntax: pst-usr.pdf/pst-quik.ps

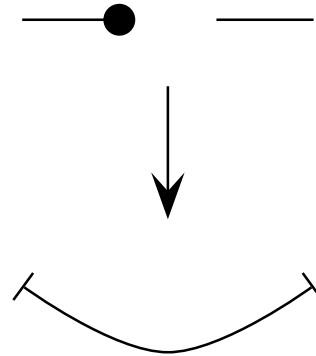
This slide just shows some of the more important geometrical PSTricks elements.

The interpolated points of the curve and the control points of the Bezier spline are visible because of the option `showpoints` is set to `true`.

For the exact syntax of the corresponding macros the reader is referred to the PSTricks user guide.

7 Line ends ‘Arrows’

```
\psline{-*}(1,6)(2,6)
\psline{-}(3,6)(4,6)
\psline{->}(2.5,5)(2.5,3)
\pscurve{|-|}(1,2)(2.5,1)(4,2)
```



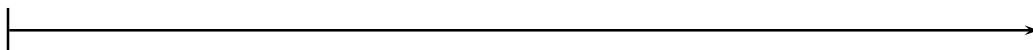
12

There is a special way to specify the look of the ends of lines, curves, splines and some other one-dimensional objects. Such ends are called as arrow heads in the user guide. You can set the shape of the arrow heads in curly braces just before the coordinate pairs of the points of the PSTricks element. There are many more arrow heads available in PSTricks than those on the slide.

Using all possibilities to specify options of one-dimensional objects one ends up with the sequence: **1st** options in brackets, **2nd** arrow options in curly braces, **3rd** points in parenthesis¹.

Example:

```
\psline[tbarsize=10pt 5]{|->}(0.5,0.5ex)(\linewidth,0.5ex)
```



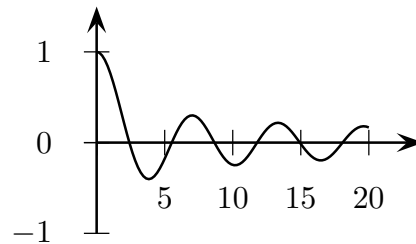
¹Thanks to Stefan Berthold who pointed out that I should mention that.

8 File plots

```
\usepackage{pst-plot}  
...  
\psset{xunit=0.3\psunit,yunit=2\psunit}  
\psaxes[Dx=5]{->}(0,0)(0,-1)(24,1.5)  
\fileplot{bessel.dat}
```

Contents of the file `bessel.dat`:

```
0 1  
0.20202 0.989823  
0.40404 0.959602  
:  
:
```



13

A further basic PSTricks element is the file plot which is defined in the style file `pst-plot.sty`.

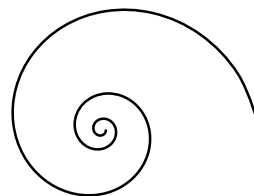
With the `\fileplot` macro data is read from a file (on the slide the file's name is `bessel.dat`). The points from the data file are plotted with lines connecting consecutive points (the latter behaviour can be changed via the option `plotstyle`).

The scale of the plot is determined by the options `xunit` and `yunit`. The origin of the coordinates is the current point $(0,0)$.

The `\psaxes` macro can be used to provide the plot with axes. The first of the three coordinate pairs behind `\psaxes` determines the origin of the coordinate system the second one determines the lower left corner of the coordinate system and the third the upper right corner (a similar syntax can be used for the `\psgrid` macro mentioned above).

9 Function plots (parametric)

```
\def\Euler{2.718 }  
\parametricplot[plotstyle=curve]{0}{360}{  
  3 t mul cos \Euler -0.01 t mul exp mul  
  3 t mul sin \Euler -0.01 t mul exp mul }
```



$$(x(t), y(t)) = \exp(-0.01t) \cdot (\cos(3t), \sin(3t))$$

with $t \in [0, 360^\circ]$

Postscript: Chapter 'Operators' in RedBook.pdf by Adobe Inc.

14

Also function plots can be made by PSTricks. For x - y -plots the `\psplot` macro may be used. On this slide, however, the more flexible `\parametricplot` macro is demonstrated.

The parametric description of the curve to be plotted is shown as the black formula on the slide (it represents three windings of a logarithmic spiral).

The parameter identifier in parametric plots is always `t`. The start and the end of the parameter domain are given inside the first and the second curly braces, resp., of the `\parametricplot` macro.

The defining function of the curve is described as a postscript program fragment inside the third curly braces of the `\parametricplot` macro.

With the option `plotstyle` set to `curve` splines are used to interpolate the computed samples along the function graph. Without this option points are connected with line segments. You should try this for yourself to see the difference.

For an exact description of the Postscript programming language the reader is referred to the Postscript Language Reference called the Red Book which can be downloaded from

<http://partners.adobe.com/asn/tech/ps/specifications.jsp>.

For the moment think of the program segment just as a representation of the formula in reverse polish notation¹.

In postscript fragments expandable \LaTeX macros such as `\Euler` on the slide may be used. Note the space in the definition of the macro `\Euler`. This space is necessary since the space after `\Euler` in the postscript fragment is eaten during the \LaTeX expansion process.

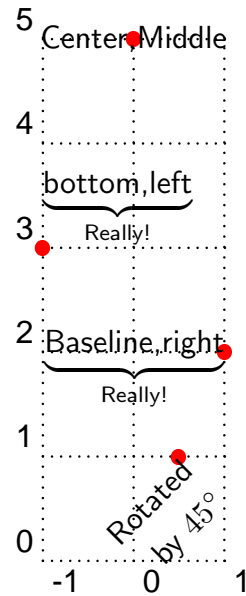
¹Todo: This should be explained in some more detail.

10 Placing whatever, wherever

```

\psdots[linecolor=red,dotsize=10pt]
(0,5)(-1,3)(1,2)(0.5,1)
\rput(0,5){Center,Middle}
\rput[b1](-1,3){$\underbrace{
  \text{bottom,left}
}_\{\text{Really!}\}$}
\rput[Br](1,2){$\underbrace{
  \text{Baseline,right}
}_\{\text{Really!}\}$}
\rput[tr]{45}(0.5,1)
{\parbox{5cm}{\flushright Rotated\
  by $45^\circ$}}

```



15

Now, that we know some PSTricks elements we can focus on the modification of such elements or groups of such elements.

As we have already seen, the `\rput` macro can be used to place objects. The second mandatory argument (in curly braces) is the stuff to place the first mandatory argument (in parenthesis) is the coordinate pair of the point where the stuff is placed.

Now we turn to the optional arguments of the `\rput` macro.

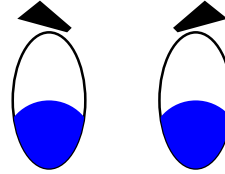
The first one is given in brackets. It determines the justification of the bounding box of the object to place with respect to the point given in parenthesis. The admissible values are the same as the values for the option `origin` of the `\includegraphics` macro. For an instance `[br]` for bottom – right. The default is `mc` meaning middle – center.

The second optional argument is given in curly braces just before the left parenthesis. It is a number that stands for the rotation angle as illustrated in the last instance of the `\rput` macro on the slide.

The two optional arguments make `\rput` more flexible than the `\put` macro of the `picture` environment.

11 Clipping and scaling

```
\def\myEye{
  \begin{psclip}{\psellipse(0,0)(0.8,1.5)}
    \pscircle*[linecolor=blue](0,-1){1}
  \end{psclip}
  \pspolygon*(-0.4,1.5)(0.7,1.8)
    (0.2,2.2)(-0.5,1.6)
}
\rput(8,2){\myEye}
\rput(6,2){\scalebox{-1 1}{\myEye}}
```



16

A very important feature provided by Postscript is clipping. Clipping is also supported by PSTricks. It makes PSTricks even superior to some wysiwyg¹ vector drawing programs.

Clipping requires two arguments firstly a closed path the so called clip path and secondly the clipped graphics. Only that portion of the clipped graphics is shown that lies inside the region bounded by the clip path.

PSTricks provides the `psclip` environment for clipping. The mandatory argument of the `psclip` environment in the curly braces is the clip path. The clipped graphics is written into the `psclip` environment. On the slide the blue iris of the eye is drawn as a blue full circle. The visible part of the iris is cut out via a clip path drawn by a `\psellipse` macro. Note, that the clip path is drawn as an ordinary PSTricks element with the current line style and fill style. If you want to make the clip path invisible you have to set the `linestyle` option to `none`.

A standard feature of drawing programs is scaling. This is provided by PSTricks through the `\scalebox` macro. The two numbers in the first argument of `\scalebox` are the scaling factors in x - and y -direction. The stuff to be scaled is put into the second argument.

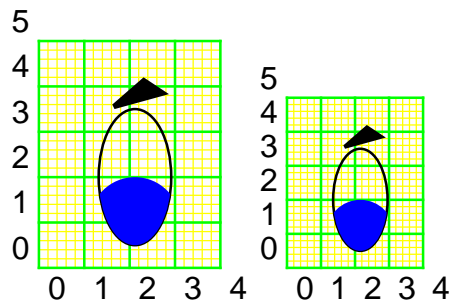
On the slide we use scaling for reflecting the eye in x -direction².

¹What You See Is What You Get.

²One can also use `\reflectbox` for this purpose (thanks to Björn).

12 Easy way to scale everything

```
\begin{pspicture}(4,5)
  \rput(2,2)\myeye
\end{pspicture}
\hspace{1cm}
\psset{unit=0.75cm}
\begin{pspicture}(4,5)
  \rput(2,2)\myeye
\end{pspicture}
```



17

One could put units (like `pt`, `cm`, `in`) on each measure inside a `pspicture` environment. But it is very wise **not** to do so, since if one uses unit-free measures the picture keeps scalable.

One can easily change the size of the picture by setting the option `unit` to the desired value. None-uniform scaling is also possible with the options `xunit` and `yunit`.

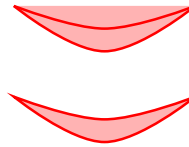
On the slide the default value of `1cm` for `unit` is used for the left picture. Exactly the same picture is drawn on the right side with `unit` set to `0.75cm`.

13 Enrolling one's own path

```

\psset{linecolor=red,fillcolor=pink,fillstyle=solid}
\rput(0,2){
  \pscurve(1,0)(0,-1.0)(-1,0)
  \pscurve(-1,0)(0,-0.5)(1,0)
}
\pscustom{
  \pscurve(1,0)(0,-1.0)(-1,0)
  \pscurve[liftpen=1](-1,0)(0,-0.5)(1,0)
}

```



18

With the Postscript language a visible object is created in two steps. In the first step a path is defined and in the second step some visualisation like stroking or filling is done with the path. Different geometrical objects can be used to describe one path. An example is the postscript fragment

```

0 -50 rlineto
50 0 100 0 100 50 rcurveto
stroke

```

which defines a path consisting of a line and a bezier curve and afterwards strikes the path. The result of this Postscript fragment is shown here: - with `stroke` replaced by `fill` one obtains: -

Normally, PSTricks concludes the postscript fragment corresponding to a macro like `\pscurve` immediately with a `stroke` and/or `fill` command (depending on the `linestyle`/`fillstyle` options). Thus, each of the first two `\pscurve` commands on the slide fills its own path and one obtains the upper figure on the slide.

It might be that one just wanted to fill the region between the two curves. This means that the two curves should belong to the same filled path.

PSTricks defines the `\pscustom` macro. Inside the argument of this macro the Postscript fragments belonging to macros like `\pscurve`/`\psline`/`\psbezier` are not concluded with `fill` or `stroke`. Thus all the objects inside the `\pscustom` argument belong to the same path. At the end of the execution of the `\pscustom` macro the path is stroked/filled according to the `fillstyle`/`linestyle` options.

The option `liftpen=1` prevents PSTricks to try to connect the two curves in some strange manner. Without this option one would obtain something like:

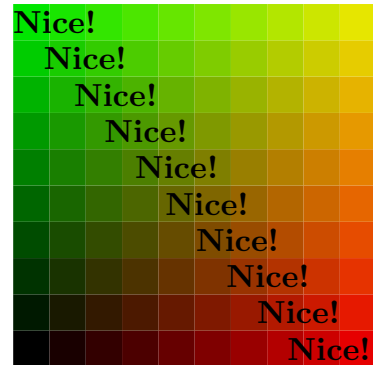


14 Repetition (and rgbcolors)

```

\usepackage{pstcol,multido}
...
\psset{fillstyle=solid,linestyle=none}
\multido{\nx=0.0+0.1}{10}{%
  \multido{\ny=0.0+0.1}{10}{%
    \newrgbcolor{c}{\nx}{\ny}{0}%
    \rput(\nx,\ny){%
      \psframe[fillcolor=c](0,0)(0.1,0.1)%
    }}
\multirput[B1](0,0.92)(0.084,-0.1){10}{Nice!}

```



19

The `multido` style file from the PSTricks package provides the two macros `\multido` and `\multirput`.

The `\multido` macro allows to repeat some sequence of L^AT_EX macros a given number of times. The first argument of the `\multido` macro contains an 'identifier = initial value + increment' sequence, the second argument contains the number of repetitions and the last one contains the L^AT_EX commands to be repeatedly executed.

The first letter in the identifier of the control variable must be a 'n' for a real number and an 'i' for an integer.

As demonstrated on the slide the `\multido` macros may be nested.

More than one control variable might be defined in the first argument of the `\multido` macro by a coma separated list.

The `\multirput` macro on the slide produces once a box from the L^AT_EX code 'Nice!' then it places it ten times with initial position (0,0.92) and increment (0.084,-0.1).

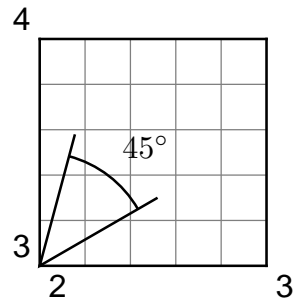
The package `pstcol` allows to define new RGB-colors with the help of a macro `\newrgbcolor`.

The first argument of `\newrgbcolor` is the name of the new color the second argument is a space separated triple of numbers which stand for the red, green, and blue fraction of the newly defined color.

That color can then be used as value for color options like `linecolor` and `fillcolor`.

15 Special coordinates (e.g. polar coordinates)

```
\SpecialCoor  
\rput(2,3){  
  \psline(0.6;30)(0,0)(0.6;75)  
  \psarc(0,0){0.5}{30}{75}  
  \rput[b1](0.6;52.5){$45^\circ$}  
}
```



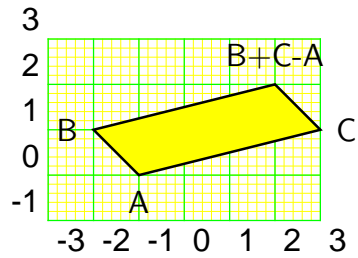
20

Normally, points are specified in Cartesian coordinates written as a **coma** separated pair inside of parenthesis. With the `\SpecialCoor` macro one can tell PSTricks to recognise other types of coordinates.

For an example polar coordinates are written as a pair of radius and angle (in degrees) separated by a **semicolon**.

Polar coordinates are always given with respect to the current origin. If needed one can move that origin first with the help of the `\rput` macro (that is done by the `\rput(2,3)` on the slide).

16 Special coordinates (postscript)



```

\Pt A(-1,0)\Pt B(-2,1)\Pt C(3,1)
% \Pt A(-1,0) -> \A=-1,0 \AX=-1 \AY=0
\pspolygon[fillstyle=solid,fillcolor=yellow](\B)(\A)(\C)
(!
  \BX\space \CX\space add \AX\space sub
  \BY\space \CY\space add \AY\space sub
)

```

21

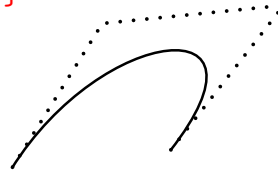
One can also use Postscript fragments to compute the Cartesian coordinates of a point. This is determined by the exclamation mark next to the left parenthesis. On the slide we use a Postscript fragment to compute the fourth point of a parallelogram.

`\Pt` is a self-made auxiliary macro to extract the coordinates from a given pair (it assigns to an argument like `A(-1,0)` the macros `\A`, `\AX` and `\AY` as indicated in the comment on the slide).

Note the `\space` macros in the Postscript fragment. These expand to spaces which are necessary because of `LATEX` eats all spaces behind a macro name. With `\BX=-1` and `\CX=3` the sequence `'\BX \CX add'` would expand to `'-13add'` instead of the desired `'-1 3 add'`.

17 Example for the usage of (L)A_TE_X-commands

```
\newcommand\myPairs{(0.3,0.2)(0.7,1.0)%  
                    (1.5,1.1)(1,0.3)}  
{\psset{linestyle=dotted,  
        linewidth=1.5\pslinewidth}  
  \expandafter\psline\myPairs  
  \expandafter\psbezier\myPairs  
}
```



22

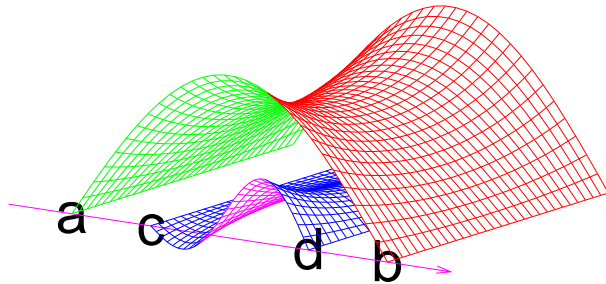
(This slide should probably be left out.)

In general L^AT_EX macros can be used together with PStricks macros. Thereby, one has to be careful with the argument expansion of PStricks macros.

In the example on the slide a sequence of points is used twice and, therefore, abbreviated by a macro `\myPoints`. The `\psline` macro expects the coordinates of the polygon section points in parenthesis. Therefore, we have to expand `\myPairs` to get the parenthesis before we may expand `\psline`. An `\expandafter` before `\psline` serves that purpose.

18 Importing eps-files

```
\usepackage{graphicx}  
...  
\rput(7,-5){%  
  \includegraphics%  
    [width=1\linewidth]%  
    {graph1.eps}%  
}
```



23

No special macro is provided by PSTricks to embed pictures into the document. This is no problem at all since the `graphicx` package already provides the `\includegraphics` macro that is very well suited for that purpose.

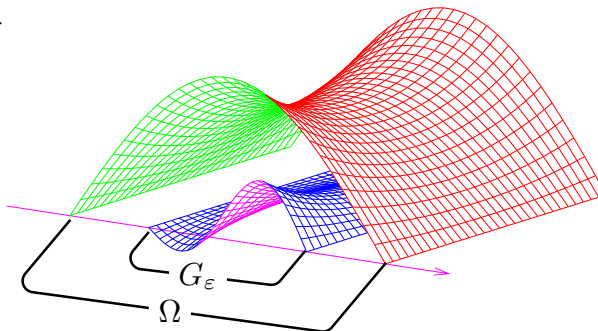
As shown on the slide the `\rput` macro of PSTricks can help to place figures. That picture had been created by the function plotting program `gnuplot`. The labels `a,b,c,d` have been placed by `gnuplot` with their center exactly at the vertices of the domain of the plotted functions. That is relevant for the next slide where we will see that PSTricks is a good tool to make certain modifications in embedded eps pictures.

19 Nodes and node connections

```

\usepackage{psfrag,pst-node}
...
\psfrag{a}[mc][mc]{\pnode{NodeA}}
\psfrag{b}[mc][mc]{\pnode{NodeB}}
\includegraphics ...
\ncdiag[angle=-130,
        arm=2cm,
        linearc=0.25cm]
        {NodeA}{NodeB}
\mput*{\Omega}

```



24

Nodes and Node connections are extremely useful tools of PSTricks.

With the help of nodes one can refer to the position of L^AT_EX objects (or better their Postscript representations) without knowing their exact coordinates.

On the slide the `\psfrag` macro of the package `psfrag.sty` has been used for replacing the labels `a,b,c,d` (see the figure on the previous slide) with `pnodes`. These nodes do not have a geometrical extent (`pnode` stands for ‘point node’). They just make the label positions available to PSTricks. The nodes created in that way can then be referred to by the assigned node names `NodeA` and `NodeB`.

In the example on the slide the nodes `NodeA` and `NodeB` are connected by a line (`NodeC` and `NodeD` are treated analogously). This connection has been created by the `\ncdiag` macro (`\ncdiag` standing for ‘node connect diagonal’). The `\ncdiag` macro draws arms of `arm=2cm` length in an angle of `angle=-130` degrees starting from the nodes. The end points of these arms are then connected by a straight line. The option `linearc=0.25cm` smoothes out the resulting vertices at the arm ends by small arcs.

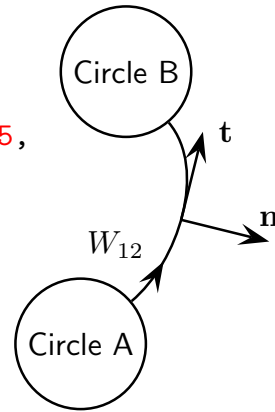
The label Ω is placed in the middle of the node connection by the `\mput*` macro. The star form of `\mput` is used to get the white underlay of the label Ω .

20 ‘Labeling’ node connections

```

\usepackage{pstricks-add}
...
\cnodeput(2,1){cnA}{Circle A}
\cnodeput(2.5,4){cnB}{Circle B}
\ncurve[ArrowInside=->,ArrowInsidePos=0.25,
  angleA=40,angleB=-50]{cnA}{cnB}
\aput(0.25){$W_{12}$}
\lput{:0}{
  \psline{->}(0,0)(1,0)
  \uput[0]{*0}(1,0){$\mathbf{t}$}
  \psline{->}(0,0)(0,-1)
  \uput[90]{*0}(0,-1){$\mathbf{n}$}}

```



25

There are many different types of nodes. On this slide `\cnodeput` has been used to create two circle nodes with inserted text. The curve connecting the circles is created by the `\ncurve` macro. The options `angleA=40`, `angleB=-50` determine the angles at which the curve leaves the circle nodes.

`ArrowInside=->`, `ArrowInsidePos=0.25` are options that were added to PSTricks after the development of PSTricks had been froze. Such add-ons to PSTricks are contained in `pstricks-add.sty` which must be loaded separately.

With help of the `ArrowInside=->` option one can set an arrow head anywhere on a curve. The place where to set the arrow head is determined by the `ArrowInsidePos=0.25` option. The curve is parameterised from 0 to 1 and the 0.25 tells PSTricks to set the arrow head at one quarter of the curve.

The `\aput` macro is used to put the label ' W_{12} ' above the curve at the curve parameter 0.25 given in parenthesis.

The `\lput` macro is also intended for labelling. In the example however, it serves to draw the local coordinate system with the tangent vector \mathbf{t} and the normal vector \mathbf{n} . The stuff to be placed is given in the second argument of `\lput`. The stuff is rotated by the angle in the first argument of `\lput`. The colon says that the angle is given in local polar coordinates of the curve. That means the tangent of the curve has an angle of 0 degrees. That makes the task to draw the tangent and the normal vector of the curve very easy.

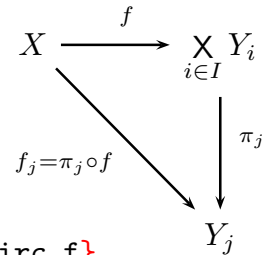
The `\uput` macro is used to label the vectors \mathbf{t} and \mathbf{n} . The angle in the brackets determines in which direction the label is translated from the point given in parenthesis. The argument in the first curly braces determines the angle by which the label is rotated. The star indicates that the angle is measured with respect to the global coordinate system. The value 0 leaves the labels unrotated upright.

21 More nodes and node connections

```

\begin{equation*}
\begin{array}{c@{\hspace{3cm}}c}
\color{red}\Rnode{N1}{X} & \color{red}\Rnode{N2}{\color{red}
\times\limits_{i\in I} Y_{i}}\color{red} \\
& \color{red}\Rnode{N3}{Y_{j}}
\end{array}
\end{equation*}
\psset{nodesep=0.3cm}
\everypsbox{\scriptstyle}
\color{red}\ncLine{->}{N1}{N2}\Aput{f}
\ncLine{->}{N1}{N3}\Bput{f_{j}=\pi_{j}\circ f}
\ncLine{->}{N2}{N3}\Aput{\pi_{j}}
\end{equation*}

```



26

Another important application of nodes and node connections is the illustration of relationships as, for an example, the commutative diagram on the slide.

The diagram is embedded into an `equation*` environment of AMS- \LaTeX . For node placing the \LaTeX native `array` environment is used. That shows that nodes and node connections can be used everywhere in a \LaTeX document. They are not restricted to `pspicture` environments.

The `\Rnode` macro puts a invisible rectangular node with the name its first argument around the stuff in its second argument. The arrows are drawn by the `\ncLine` commands. Labels are attached to the arrows by the `\Aput` and `\Bput` commands. `\Aput` and `\Bput` are variants of `\aput` and `\bput` which use default values for the positioning of the labels.

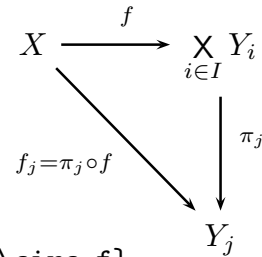
`\ncLine` is a variant of `\ncline` which takes care that lines connecting nodes on the same line are straight horizontal independent of the size of the stuff inside the nodes.

22 Node placement with psmatrix

```

\begin{psmatrix}[mnode=R,colsep=3cm,rowsep=3cm]
  X & \bigtimes\limits_{i\in I} Y_i \\
  & Y_j
\end{psmatrix}
\psset{nodesep=0.3cm}
\everypsbox{\scriptstyle}
\ncLine{->}{1,1}{1,2}\Aput{f}
\ncLine{->}{1,1}{2,2}\Bput{f_j}=\pi_j\circ f}
\ncLine{->}{1,2}{2,2}\Aput{\pi_j}

```



27

The same commutative diagram as on the previous slide is created here via the `psmatrix` environment of PSTricks. This is an environment especially designed for node placing.

About each entry of the `psmatrix` a node is set whose type is determined by the `mnode` option of the `psmatrix` environment (`mnode=R` stands for `\Rnode`).

The `colsep` and `rowsep` options specify the distances of the nodes.

The node labels of the `psmatrix` entries are automatically generated as pairs of row and column numbers.

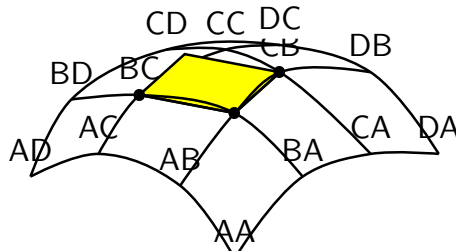
These labels can then be used with the `\ncLine` macros.

23 Including postscript code in `\pscustom`

```

\pnode(0.0,-0.6){AA}
...
\pnode(1,4.1){DC}
\pscurve(AD)(AC)(AB)(AA)
...

```



```

\pscustom[fillcolor=yellow,fillstyle=solid]{
  \psline(BC)(BB)(CB)
  \coord(BC)(CB)
  \code{\AddPairs} % x1 y1 x2 y2 -> (x1+x2) (y1+y2)
  \coord(BB)
  \code{\SubPairs lineto}
\closepath}

```

28

We have already seen an example where the fourth point of a parallelogram has been computed by a Postscript fragment. There the coordinates of the points had been defined as \LaTeX macros `\AX`, `\AY`, `\BX`, `\BY`, `\CX`, `\CY`. The situation is a bit different when the points are defined as nodes.

The coordinates of nodes are not already known during the \LaTeX run but only when the Postscript file is interpreted.

In the example on the slide nodes are used to draw the curved surface and the parallelogram has to be computed depending on these nodes.

Inside the `\pscustom` environment (which we have already used to construct Postscript paths) there are some additional macros available which are in a certain sense ‘closer’ to the postscript interpreter.

The `\coord` macro allows to insert¹ the coordinate pairs of the points given as its arguments into the Postscript file. Thereby, the points can be specified in any admissible way of PSTricks, especially as nodes, too.

The `\code` macro allows to insert Postscript code fragments. Such fragments do the calculations in the example. They are abbreviated as macros `\AddPairs` and `\SubPairs`. The definitions of these macros are given on the next slide.

The last Postscript fragment is finished with a `lineto` command which draws the line from node `CB` to the just computed point. The special command `\closepath` is used to close the parallelogram (The line from the computed point to the node `BC` is automatically added).

The `\pscustom` macro takes care of stroking and filling the path corresponding to the chosen options.

¹More exactly, postscript code is inserted which computes the coordinates.

24 The corresponding postscript codes

```
%% x1 y1 x2 y2 -> (x1+x2) (y1+y2)
\def\AddPairs{ exch 4 1 roll add 3 1 roll add exch }

%% x1 y1 x2 y2 -> (x1-x2) (y1-y2)
\def\SubPairs{ exch 4 1 roll sub 3 1 roll exch sub exch }
```

These are just the Postscript fragments of the previous example.

25 ps4pdf: Preparing the L^AT_EX-file

```
\documentclass{article}
\usepackage{hyperref,graphicx,ps4pdf}
\PSforPDF{\usepackage{pstricks,pst-plot}}
\begin{document}
\title{Example for the usage of ps4pdf}\maketitle\centering
\PSforPDF{
  \begin{pspicture}(-5,-5)(5,5)
    \rput(0,0){\psovalbox{That would be some complicated graphic.}}
  \end{pspicture}
}%% End of PSforPDF.
\par\hypertarget{Target}{That's the target.}
\newpage
\hyperlink{Target}{That's the link.}
\end{document}
```

30

PSTricks produces Postscript fragments which are inserted into the final Postscript file. That means, that L^AT_EX files with PSTricks macros cannot directly be translated by `pdflatex`. On the other side some L^AT_EX features are only available by `pdflatex`.

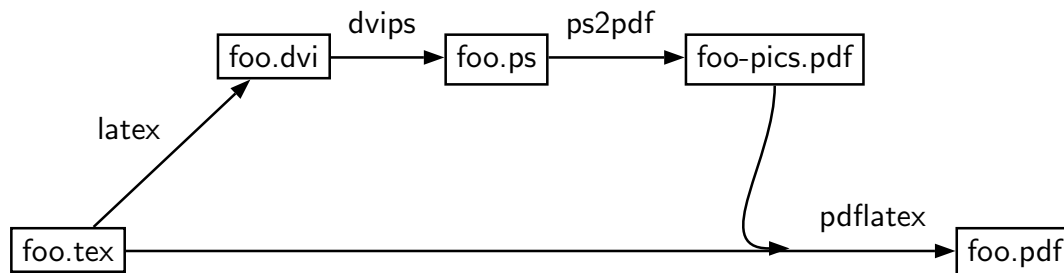
The package `ps4pdf` makes it possible to translate L^AT_EX files with PSTricks macros via `pdflatex`.

For the usage of `ps4pdf` the packages `graphicx` and `ps4pdf` must be loaded and all PSTricks relevant parts of the document must be encapsulated as arguments of the `\PSforPDF` macro as shown in the example on the slide.

The next slide explains how to use the L^AT_EX file with `pdflatex`.

26 ps4pdf: pstricks & pdflatex

CTAN: /tex-archive/macros/latex/contrib/ps4pdf/ps4pdf.sty
(needs graphicx, preview, ifpdf, and ifvtex)



31

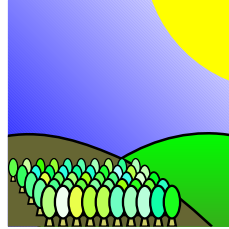
Translating a \LaTeX file `foo.tex` with PSTricks macros via `pdflatex` is a two path process which is shown on the slide.

In the first path (the upper path on the slide) exclusively the pictures are created in a Postscript file and translated into a PDF file via `ps2pdf`. The resulting PDF file should be named `foo-pics.pdf`.

In the second path `foo.tex` is translated with `pdflatex` into `foo.pdf`. On this path the pictures from `foo-pics.pdf` are included into the PDF file.

27 Other nice stuff – fillstyle=gradient

```
\usepackage{pst-grad}
...
\begin{psclip}{
  \psframe[linestyle=none,
    fillstyle=gradient,
    gradbegin=white,gradend=blue,
    gradmidpoint=1,
    gradangle=-45](0,0)(5,5)
}
... other stuff ...
\end{psclip}
```



32



At the end of the presentation there are two slides left with some nice funny features of PSTricks.

The package `pst-grad` provides a new fill style named `gradient`. This fillstyle allows continuous transitions from one color to another one. The start color and end the color can be set via the options `gradbegin` and `gradend`. These options can also be set like a color definition, e. g.,

```
\newrgbcolor{gradbegin}{0.1 0.2 0.3}
```

where 0.1, 0.2, and 0.3 are the red, the green and the blue color value, resp.

The option `gradangle` determines the direction in which the color keeps constant.

The region to be filled is parameterised by values from 0 to 1. The parameter value set by the `gradmidpoint` option determines where the color `gradend` is taken on. For an example with `gradmidpoint=1` the color `gradend` is taken on at the end of the filled region, e. g. , and for the setting `gradmidpoint=0.5` the color is taken on in the middle of the filled region, e. g. .

28 Other nice stuff – Playing with Text

```
\usepackage{pst-text,pst-char,ae}
...
\pstextpath(0,-3ex){\psellipse(0,0)(3,2)}{
  \multido{}{6}{PS\LaTeX{}}}
\psset{fillstyle=gradient,gradbegin=red,gradend=blue}
\rput(0,0){\pscharpath{\fontsize{1.3cm}{1.3cm}\selectfont\LaTeX}}
```



33

The `pst-text` style provides the macro `\pstextpath`. This macro serves to set text along a curve. The first argument (given in parenthesis) is the position of the first text from the start of the curve the second argument defines the curve and the third one is the text to be set along the curve.

The package `pst-char` provides the macro `\pscharpath`. With the help of `\pscharpath` one can transform the outline of a letter into an ordinary Postscript path. This Postscript path can then be used in the same way as a postscript path created by commands like `\psframe` or `\pspolygon`. In the example it has been used to dye the string `LATEX` with `fillstyle=gradient`.

The path of characters can only be extracted for outline fonts. Therefore, you have to load some package which provides an outline font. In the example on the slide the package `ae` is used for that reason.

That is it for today. Hope you had some fun,
Tobias Nähring

PS: My english not so good. If you find some errors please let me know. My email-address is: `i@tn-home.de`.

Thanks to Dominic Hughes (Stanford University) for correcting some spellings.