Graphics in LaTeX 2ε

osurs@bluewin.ch — Urs Oswald — http://www.ursoswald.ch

10th March 2003

Contents

1 Overview

There are two ways of providing LaTeX documents with pictures:

(1) by directly programming the pictures within LaTeX,

(2) by importing pictures.

We will keep in mind that ultimately, one of the following kinds of documents is to be produced:

- `PDF`,
- `PS`,
- `HTML+PNG`.

TeX itself comes along with a very powerful picture generator:

- `MetaPost`.

`MetaPost` is a variant of Donald Knuth's METAFONT. It produces `mps` files, which form a subset of the `eps` files (eps: "encapsulated PostScript"). "Encapsulated" means that these `PostScript` files have a frame preventing any disturbing interferences on import into `PostScript` or `PDF` documents.

Although programming pictures directly in LaTeX is severely restricted, and often rather tiresome, there are still reasons for doing so. The documents thus produced are "small" with respect to bytes. There are no additional graphics files to be dragged along. Therefore, Sections 2.3 and 2.4 of this script present some "real life" graphics done within the original `picture` environment of LaTeX.

1.1 Programming pictures in LaTeX

The `picture` environment allows programming pictures directly in LaTeX. However, there are rather severe constraints, especially for the mathematician, as the slopes of line segments as well as the radii of circles are restricted to a narrow choice of values. On the other hand, the `picture` environment brings with it the `qbezier` ("quadratic Bézier" curves) command. Many frequently used curves can be satisfactorily approximated by quadratic Bézier curves, although this may require some mathematical toil. If a programming language, such as Java, is used to generate `qbezier` blocks of LaTeX input files, the `picture` environment becomes quite powerful. Of course, the ideal language to produce `qbezier` blocks is `APL`. Detailed descriptions of the `picture` environment are presented, for example, in Lampert[6], or Kopka[7].

The restrictions on line segments and circles can be overcome by the use of packages like `epic` and `eepic` (described, for instance, in Goossens, Mittelbach, Samarin[1]), or `pstricks` (described in detail in Goossens, Rahtz, Mittelbach[2]).

While the former two packages just enhance the `picture` environment of LaTeX, the `pstricks` package has its own environment for pictures, `pspicture`. The power of `pstricks` stems from the fact that this package makes extensive use of `PostScript` possibilities. Fig. 1 illustrates the possible use of the three packages for producing `PDF`, `PS`, and `HTML+PNG`.

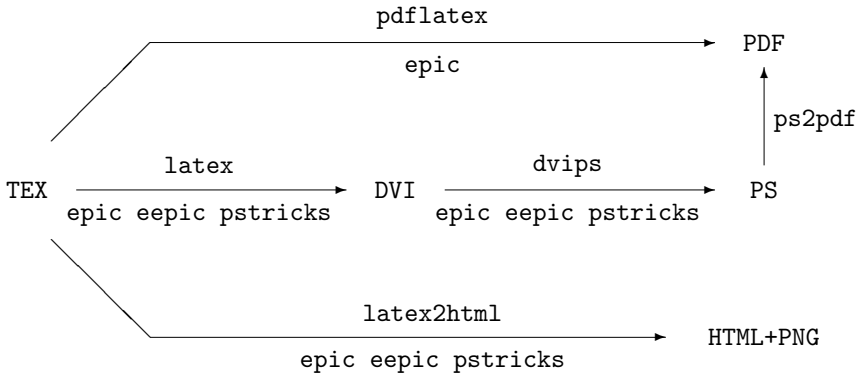


Fig. 1: Producing `PDF`, `PS`, and `HTML+PNG`

Remarks

- In this document, `latex2html` means version 2002-1 (1.68). Older versions of `latex2html` may generate `gif` files instead of `png` files.
- If a LaTeX input file uses the `pstricks` package, processing with `latex2html` requires the `\psset` commands to be written *within* the `pspicture` environments. Therefore, the arguments of the `\begin{pspicture}` command have to refer to the default unit of `pstricks`, which is 1 cm.

1.2 Import

There are several packages allowing the import of pictures in LaTeX. The package most broadly recommended today (2002) is `graphicx`. The use of other packages is not considered advisable any more, as they have some drawbacks, and are not developped any further.

The `graphicx` package is loaded with the command

- `\usepackage{graphicx}`

(which has to be located in the preamble). `graphicx` is an extension of `graphics`. Moreover, while `graphics` sticks to the original TeX conventions concerning arguments, `graphicx` allows optional arguments according to the more transparent *key=value* scheme. Pictures can be imported with the command

- `\includegraphics[`*key=value*,...,*key=value*`]{beispiel.eps}`.

The optional arguments give rise to a rich choice of options with respect, for example, to scaling and rotating. RECKDAHL[9] contains an excellent exposition of the capabilities of `graphicx`. Using `graphicx`, one can routinely import pictures according to the following scheme:

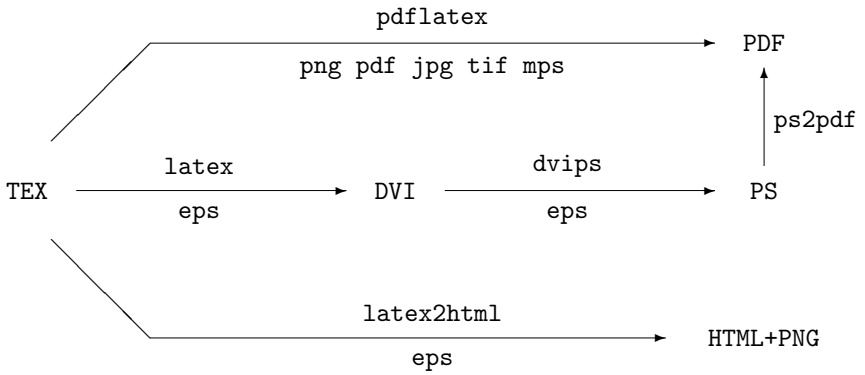


Fig. 2: Import of pictures with `graphicx`

Remarks

(1) `pdflatex`: The import of `mps` files requires the command `\DeclareGraphicsRule{*}{mps}{*}{}` in the preamble of the LaTeX file.

(2) `mps` files being special `eps` files, they can be imported, according to Fig. 2, with the other programs: `latex`, `dvips`, `latex2html`.

(3) On generation by `MetaPost`, `mps` files have names like `example.1`, `example.2`, `example.3`, ... These names may be used for the import in all described cases.

(4) In some cases, extensions such as `.eps` or `.png` can be omitted. This may be convenient if the same LaTeX file is processed to `PDF` as well as to `HTML+PNG`. For details, see RECKDAHL[9].

1.3 Technicalities

This document was produced with SuSE Linux, Version 8.0, and the TeX and LaTeX versions coming with it.

As for `latex2html`, the newer version 2002-1 (1.68) is preferrable, as the version included in SuSE 8.0 has some drawbacks. In order to get `latex2html`, go to `www.latex2html.org`.

2 The picture Environment of LaTeX

2.1 Basics

A picture environment is created with one of the two commands

- \begin{picture}(*xdim,ydim*)...\end{picture},
- \begin{picture}(*xdim,ydim*)(*x*$_0$,*y*$_0$)...\end{picture}.

The numbers *xdim*, *ydim*, x_0, and y_0 refer to \unitlength, which can be reset any time (but not within a picture environment) with a command such as \setlength{\unitlength}{1.2cm}. The default value of \unitlength is 1pt. The first pair, (*xdim,ydim*), effects the reservation, within the document, of rectangular space for the picture. The optional second pair, (x_0,y_0), assigns arbitrary coordinates to the bottom left corner of the reserved rectangle. If omitted, the second pair defaults to (0, 0), meaning that the coordinates in the drawing commands refer to the bottom left corner of the reserved rectangle.

Most drawing commands have one of the two forms

- \put(*x,y*){*object*},
- \multiput(*x,y*)(Δ*x*,Δ*y*){*n*}{*object*},

where *object* may be, i.e., \line(2, 3){1.75} or \circle*{0.25}. Bézier curves are an exception. They are drawn with commands such as

- \qbezier(1.2, 0.6)(3.1, 1.2)(2.5, 3.4).

2.1.1 Line segments

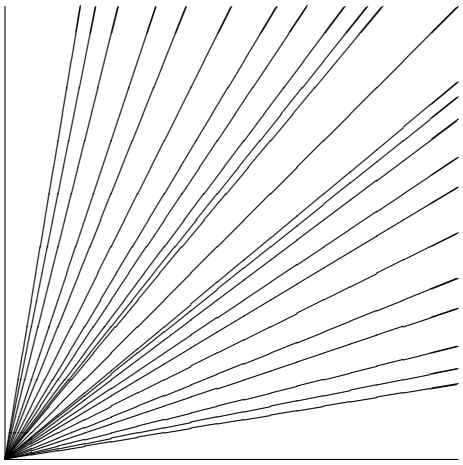


Fig. 3: Line segments

The \line command has two arguments:

- a direction vector,
- a length.

The components of the direction vector have to be coprime (no common divisor except 1), and they are restricted to the integers

$$-6,\ -5,\ \ldots,\ 5,\ 6.$$

```
\setlength{\unitlength}{6cm}
\begin{picture}(1, 1)
  \put(0, 0){\line(0, 1){1}}
  \put(0, 0){\line(1, 0){1}}
  \put(0, 0){\line(1, 1){1}}
  \put(0, 0){\line(1, 2){.5}}
  \put(0, 0){\line(1, 3){.33333}}
  \put(0, 0){\line(1, 4){.25}}
  \put(0, 0){\line(1, 5){.2}}
  \put(0, 0){\line(1, 6){.16667}}
  \put(0, 0){\line(2, 1){1}}
  \put(0, 0){\line(2, 3){.6667}}
  \put(0, 0){\line(2, 5){.4}}
  \put(0, 0){\line(3, 1){1}}
  \put(0, 0){\line(3, 2){1}}
  \put(0, 0){\line(3, 4){.75}}
  \put(0, 0){\line(3, 5){.6}}
  \put(0, 0){\line(4, 1){1}}
  \put(0, 0){\line(4, 3){1}}
  \put(0, 0){\line(4, 5){.8}}
  \put(0, 0){\line(5, 1){1}}
  \put(0, 0){\line(5, 2){1}}
  \put(0, 0){\line(5, 3){1}}
  \put(0, 0){\line(5, 4){1}}
  \put(0, 0){\line(5, 6){.8333}}
  \put(0, 0){\line(6, 1){1}}
  \put(0, 0){\line(6, 5){1}}
\end{picture}
```

Fig. 3 illustrates all 25 possible slope values in the first quadrant. The length is relative to \unitlength. The length argument is the vertical coordinate in the case of a vertical line segment, the horizontal coordinate in all other cases.

Because of the restrictions to direction vectors, drawing line segments usually necessitates an extensive search for suitable points. In many cases, drawing the desired line segments may not be possible without the use of additional packages, such as eepic, or pstricks.

2.1.2 Circles

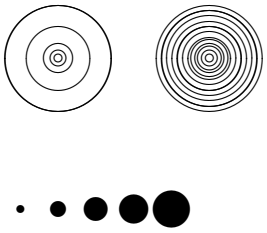


Fig. 4: Circles

The argument of the \circle command is relative to \unitlength and determines the *diameter* (not the *radius*). The original picture environment only admits diameters up to approximately 14 mm, and even below this limit, not all diameters are possible. The \circle∗ command produces disks (filled circles).

As for drawing larger circles, see Section 2.4.4 about circles and ellipses.

```
\setlength{\unitlength}{1mm}
\begin{picture}(60, 40)
  \put(20, 30){\circle{1}}
  \put(20, 30){\circle{2}}
  \put(20, 30){\circle{4}}
  \put(20, 30){\circle{8}}
  \put(20, 30){\circle{16}}
  \put(20, 30){\circle{32}}

  \put(40, 30){\circle{1}}
  \put(40, 30){\circle{2}}
  \put(40, 30){\circle{3}}
  \put(40, 30){\circle{4}}
  \put(40, 30){\circle{5}}
  \put(40, 30){\circle{6}}
  \put(40, 30){\circle{7}}
  \put(40, 30){\circle{8}}
  \put(40, 30){\circle{9}}
  \put(40, 30){\circle{10}}
  \put(40, 30){\circle{11}}
  \put(40, 30){\circle{12}}
  \put(40, 30){\circle{13}}
  \put(40, 30){\circle{14}}

  \put(15, 10){\circle*{1}}
  \put(20, 10){\circle*{2}}
  \put(25, 10){\circle*{3}}
  \put(30, 10){\circle*{4}}
  \put(35, 10){\circle*{5}}
\end{picture}
```

2.1.3 Arrows

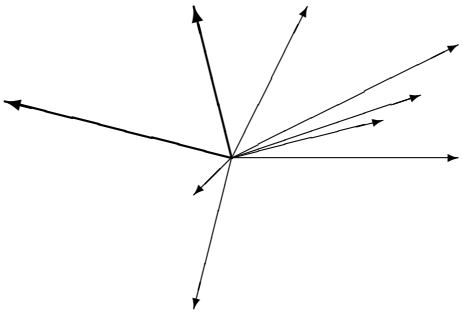


Fig. 5: Arrows

```
\setlength{\unitlength}{1mm}
\begin{picture}(60, 40)
  \put(30, 20){\vector(1, 0){30}}
  \put(30, 20){\vector(4, 1){20}}
  \put(30, 20){\vector(3, 1){25}}
  \put(30, 20){\vector(2, 1){30}}
  \put(30, 20){\vector(1, 2){10}}
  \thicklines
  \put(30, 20){\vector(-4, 1){30}}
  \put(30, 20){\vector(-1, 4){5}}
  \thinlines
  \put(30, 20){\vector(-1, -1){5}}
  \put(30, 20){\vector(-1, -4){5}}
\end{picture}
```

For arrows, the components of the direction vector are even more restricted than for line segments, namely to the integers

$$-4,\ -3,\ \ldots,\ 3,\ 4.$$

Components also have to be coprime (no common divisor except 1). Notice the effect of the `\thicklines` command on the two arrows pointing to the upper left.

2.1.4 The `multiput` and the `linethickness` command

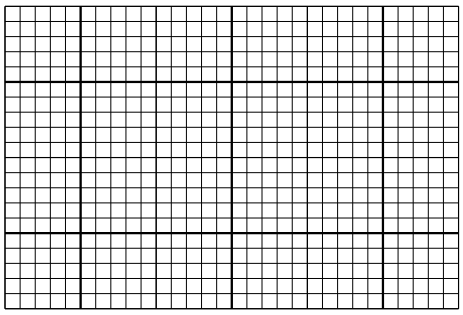


Fig. 6: The `multiput` and the `linethickness` command

```
\setlength{\unitlength}{2mm}
\begin{picture}(30, 20)
  \linethickness{0.075mm}
  \multiput(0, 0)(1, 0){31}{\line(0, 1){20}}
  \multiput(0, 0)(0, 1){21}{\line(1, 0){30}}
  \linethickness{0.15mm}
  \multiput(0, 0)(5, 0){7}{\line(0, 1){20}}
  \multiput(0, 0)(0, 5){5}{\line(1, 0){30}}
  \linethickness{0.3mm}
  \multiput(5, 0)(10, 0){3}{\line(0, 1){20}}
  \multiput(0, 5)(0, 10){2}{\line(1, 0){30}}
\end{picture}
```

The `\multiput` command has 4 arguments: the starting point, the translation vector, the number of objects, and the object to be drawn. The `\linethickness{`*length*`}` command applies to horizontal and vertical line segments, but neither to oblique line segments, nor to circles. It does, however, apply to quadratic Bézier curves!

2.1.5 Ovals, `thinlines`, `thicklines`

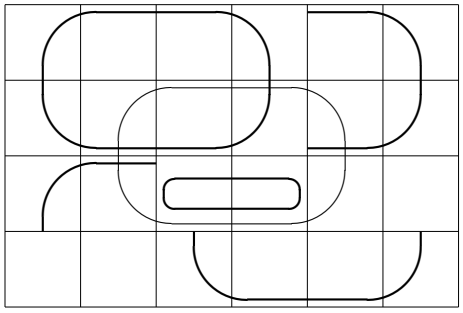


Fig. 7: Ovals, `thinlines`, `thicklines`

```
\setlength{\unitlength}{1cm}
\begin{picture}(6, 4)
  \linethickness{0.075mm}
  \multiput(0, 0)(1, 0){7}{\line(0, 1){4}}
  \multiput(0, 0)(0, 1){5}{\line(1, 0){6}}
  \thicklines
  \put(2, 3){\oval(3, 1.8)}
  \thinlines
  \put(3, 2){\oval(3, 1.8)}
  \thicklines
  \put(2, 1){\oval(3, 1.8)[tl]}
  \put(4, 1){\oval(3, 1.8)[b]}
  \put(4, 3){\oval(3, 1.8)[r]}
  \put(3, 1.5){\oval(1.8, 0.4)}
\end{picture}
```

Line thickness can be controlled by two kinds of commands: `\linethickness{`*length*`}` on the one hand, `\thinlines` and `\thicklines` on the other. While `\linethickness{0.075mm}` applies only to horizontal and vertical lines (and quadratic Bézier curves), `\thinlines` and `\thicklines` apply to oblique line segments as well as to curves such as circles or ovals.

2.1.6 Text, formulas, colors

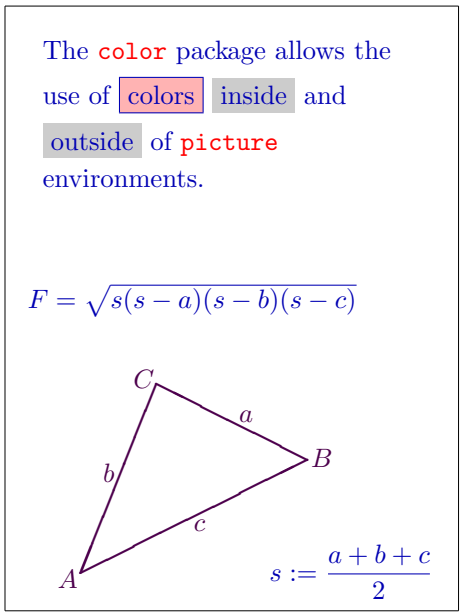


Fig. 8: Text, formulas, colors

The `color` package allows the use of colors inside and outside of `picture` environments.

Predefined colors of the `color` package are: `black`, `white`, `red`, `green`, `blue`, `yellow`, `cyan`, and `magenta`. As the current example shows, the `\definecolor` command allows for the definition of arbitrary further colors in terms either of the `gray`, or the `rgb` *color model*. (LaTeX also supports a third color model, `cymk`).

```
\definecolor{hellgrau}{gray}{.8}
\definecolor{dunkelblau}{rgb}{0, 0, .7}
\definecolor{roetlich}{rgb}{1, .7, .7}
\definecolor{dunkelmagenta}{rgb}{.3, 0, .3}

\setlength{\unitlength}{1cm}
\begin{picture}(6, 8)
  \linethickness{0.075mm}
  \multiput(0, 0)(6, 0){2}{\line(0, 1){8}}
  \multiput(0, 0)(0, 8){2}{\line(1, 0){6}}
  \thicklines
  \color{dunkelmagenta}
  \put(1, .5){\line(2, 1){3}}
  \put(4, 2){\line(-2, 1){2}}
  \put(2, 3){\line(-2, -5){1}}
  \put(.7, .3){$A$}
  \put(4.05, 1.9){$B$}
  \put(1.7, 2.95){$C$}
  \put(3.1, 2.5){$a$}
  \put(1.3, 1.7){$b$}
  \put(2.5, 1.05){$c$}
  \color{dunkelblau}
  \put(0.3, 4){$F=\sqrt{s(s-a)(s-b)(s-c)}$}
  \put(3.5, 0.4){$\displaystyle
    s:=\frac{a+b+c}{2}$}
  \put(0.5, 7.3){The {\textcolor{red}{\texttt{%
                color}}} package allows the}
  \put(0.5, 6.7){use of \fcolorbox{dunkelblau}%
                {roetlich}{colors} \colorbox%
 {hellgrau}{inside} and}
  \put(0.5, 6.1){\colorbox{hellgrau}{outside}
                of \textcolor{red}{\texttt{%
                picture}}}
  \put(0.5, 5.6){environments.}
\end{picture}
```

2.1.7 Quadratic Bézier curves

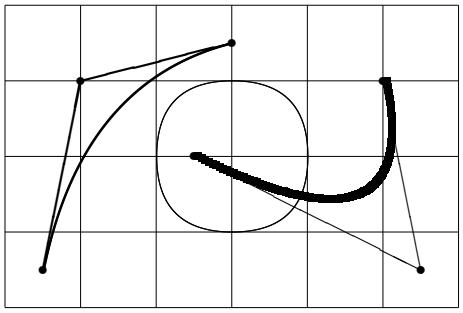


Fig. 9: Quadratic Bézier curves

```
\setlength{\unitlength}{1cm}
\begin{picture}(6, 4)
  \linethickness{0.075mm}
  \multiput(0, 0)(1, 0){7}{\line(0, 1){4}}
  \multiput(0, 0)(0, 1){5}{\line(1, 0){6}}
  \put(.5, .5){\circle*{.1}}
  \put(1, 3){\circle*{.1}}
  \put(3, 3.5){\circle*{.1}}
  \thicklines
  \put(.5, .5){\line(1, 5){.5}}
  \put(1, 3){\line(4, 1){2}}
  \qbezier(.5, .5)(1, 3)(3, 3.5)
  \thinlines
  \put(2.5, 2){\circle*{.1}}
  \put(5.5, .5){\circle*{.1}}
  \put(5, 3){\circle*{.1}}
  \put(2.5, 2){\line(2, -1){3}}
  \put(5.5, .5){\line(-1, 5){.5}}
  \linethickness{1mm}
  \qbezier(2.5, 2)(5.5, .5)(5, 3)
  \thinlines
  \qbezier(4, 2)(4, 3)(3, 3)
  \qbezier(3, 3)(2, 3)(2, 2)
  \qbezier(2, 2)(2, 1)(3, 1)
  \qbezier(3, 1)(4, 1)(4, 2)
\end{picture}
```

As Fig. 9 illustrates, splitting up a circle into four quadratic Bézier curves is not satisfactory. For a better solution, see (2) in Section 2.4.4.

Fig. 9 again shows the effect of the `\linethickness` command on horizontal or vertical lines, and of the `\thinlines` and `\thicklines` commands on oblique line segments. It also shows that both kinds of commands affect quadratic Bézier curves, each command overriding all previous ones.

Let $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ denote the end points, and m_1, m_2 the respective slopes, of a quadratic Bézier curve. The intermediate control point $S = (x/y)$ is then given by the equations

$$\left\{ \begin{array}{rcl} x & = & \dfrac{m_2 x_2 - m_1 x_1 - (y_2 - y_1)}{m_2 - m_1}, \\ y & = & y_i + m_i\,(x - x_i) \quad (i = 1, 2). \end{array} \right. \tag{1}$$

The Java program `qbezier` (see appendix for the source file `qbezier.java`) started with the line

- `java qbezier <`x_1`> <`y_1`> <`m_1`> <`x_2`> <`y_2`> <`m_2`>`

writes a file `qbezier.tex` which can be pasted into a `picture` environment, whence it draws the desired curve.

2.1.8 Marking angles

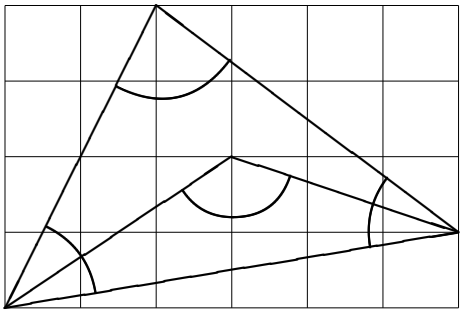


Fig. 10: Marking angles

In the current picture, arcs are approximated by quadratic Bézier curves. In order to get the control points of an arc of radius r marking the angle $\angle P_1P_2P_3$, one has to

(1) translate and rotate the angle in order to move the vertex to the origin and have the bisector point in the direction of the x axis (transformation T),

(2) reduce the vectors P_2P_1 and P_2P_3 to length r,

(3) let the intermediate control point P_2 be equal to $((x_1^2 + y_1^2)/x_1, 0)$, x_1 and y_1 being the coordinates of point P_1 after transformation T and reduction to length r,

(4) apply the transformation T^{-1} to points P_1, P_2, and P_3.

```
\setlength{\unitlength}{1cm}
\begin{picture}(6, 4)
  \linethickness{0.075mm}
  \multiput(0, 0)(1, 0){7}{\line(0, 1){4}}
  \multiput(0, 0)(0, 1){5}{\line(1, 0){6}}
  \thicklines
  \put(0, 0){\line(6, 1){6}}
  \put(6, 1){\line(-4, 3){4}}
  \put(2, 4){\line(-1, -2){2}}
  % arc
  % P1=(2.0/4.0) P2=(0.0/0.0) P3=(6.0/1.0)
  % r=1.2
  \qbezier(0.5367, 1.0733)(1.0832, 0.8)
          (1.1837, 0.1973)
  % arc
  % P1=(0.0/0.0) P2=(6.0/1.0) P3=(2.0/4.0)
  % r=1.2
  \qbezier(4.8163, 0.8027)(4.7319, 1.3092)
          (5.04, 1.72)
  % arc
  % P1=(6.0/1.0) P2=(2.0/4.0) P3=(0.0/0.0)
  % r=1.2
  \qbezier(2.96, 3.28)(2.3591, 2.4788)
          (1.4633, 2.9267)
  \put(0, 0){\line(3, 2){3}}
  \put(3, 2){\line(3, -1){3}}
  % arc
  % P1=(0.0/0.0) P2=(3.0/2.0) P3=(3.0/0.0)
  % r=0.8
  \qbezier(2.3344, 1.5562)(2.5719, 1.2)
          (3.0, 1.2)
  % arc
  % P1=(3.0/0.0) P2=(3.0/2.0) P3=(6.0/1.0)
  % r=0.8
  \qbezier(3.0, 1.2)(3.5766, 1.2)
          (3.7589, 1.747)
\end{picture}
```

If the Java program `arc` (see appendix for the source file `arc.java`) is started with the command line

- `java arc` <x_1> <y_1> <x_2> <y_2> <x_3> <y_3> <r>,

it produces a file `arc.tex` which can be pasted into a `picture` environment, whence it draws the required arc. As can be seen from the above input file, the obtuse angle is marked in parts: by consequtively marking the angles $\angle(0, 0)(3, 2)(3, 0)$ and $\angle(3, 0)(3, 2)(6, 1)$. The commands are:

- `java arc 2 4 0 0 6 1 1.2`
- `java arc 0 0 6 1 2 4 1.2`
- `java arc 6 1 2 4 0 0 1.2`
- `java arc 0 0 3 2 3 0 0.8`
- `java arc 3 0 3 2 6 1 0.8`

2.1.9 Line thickness again: the `linethickness` command

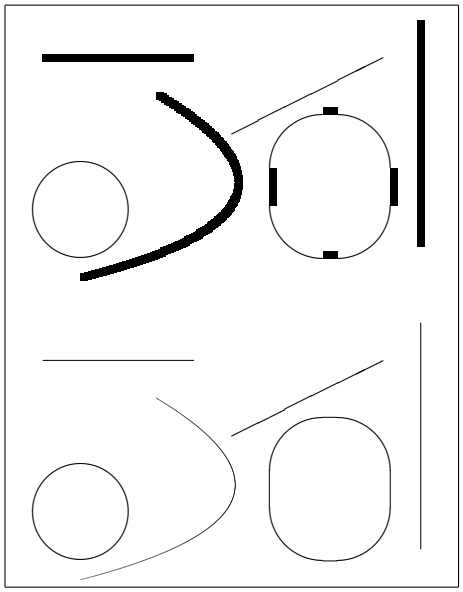


Fig. 11: Line thickness again: the `linethickness` command

```
\setlength{\unitlength}{1cm}
\begin{picture}(6, 7.7)
  \multiput(0, 0)(6, 0){2}{\line(0, 1){7.7}}
  \multiput(0, 0)(0, 7.7){2}{\line(1, 0){6}}
  \linethickness{.075mm}
  \put(0.5, 3){\line(1, 0){2}}
  \put(3, 2){\line(2, 1){2}}
  \put(5.5, 0.5){\line(0, 1){3}}
  \put(1, 1){\circle{1.3}}
  \qbezier(1, 0.1)(4.5, 1)(2, 2.5)
  \put(4.3, 1.3){\oval(1.6, 1.9)}
  \linethickness{1mm}
  \put(0.5, 7){\line(1, 0){2}}
  \put(3, 6){\line(2, 1){2}}
  \put(5.5, 4.5){\line(0, 1){3}}
  \put(1, 5){\circle{1.3}}
  \qbezier(1, 4.1)(4.5, 5)(2, 6.5)
  \put(4.3, 5.3){\oval(1.6, 1.9)}
\end{picture}
```

As can be seen from this picture, the command \linethickness{*length*} applies to horizontal and vertical lines as well as to Bézier curves, but not to circles, or oblique line segments. Ovals are a combination of horizontal and vertical lines, and arcs.

2.1.10 Line thickness again: `thinlines` and `thicklines`

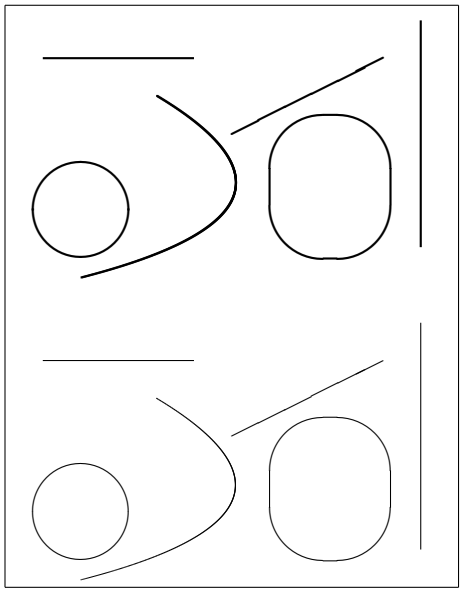


Fig. 12: Line thickness again: `thinlines` and `thicklines`

```
\setlength{\unitlength}{1cm}
\begin{picture}(6, 7.7)
  \multiput(0, 0)(6, 0){2}{\line(0, 1){7.7}}
  \multiput(0, 0)(0, 7.7){2}{\line(1, 0){6}}
  \thinlines
  \put(0.5, 3){\line(1, 0){2}}
  \put(3, 2){\line(2, 1){2}}
  \put(5.5, 0.5){\line(0, 1){3}}
  \put(1, 1){\circle{1.3}}
  \qbezier(1, 0.1)(4.5, 1)(2, 2.5)
  \put(4.3, 1.3){\oval(1.6, 1.9)}
  \thicklines
  \put(0.5, 7){\line(1, 0){2}}
  \put(3, 6){\line(2, 1){2}}
  \put(5.5, 4.5){\line(0, 1){3}}
  \put(1, 5){\circle{1.3}}
  \qbezier(1, 4.1)(4.5, 5)(2, 6.5)
  \put(4.3, 5.3){\oval(1.6, 1.9)}
\end{picture}
```

As the comparison of the lower half of the picture with the upper half shows, the \thinlines and the \thicklines commands apply to all line segments and curves, but the effect cannot said to be very striking.

2.1.11 Using the qbezier command to draw line segments

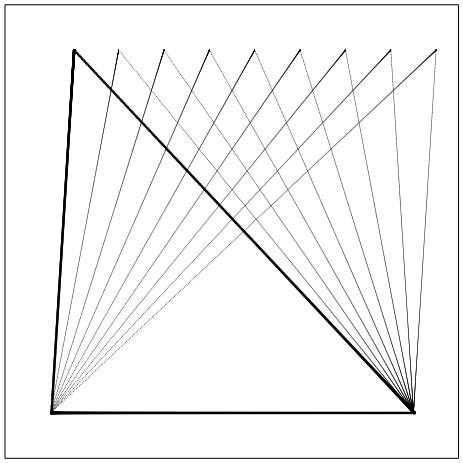


Fig. 13: Using the **qbezier** command to draw line segments

```
\newcommand{\strecke}[2]{\qbezier(#1)(#1)(#2)}

\setlength{\unitlength}{3cm}
\begin{picture}(2,2)(-1,-1)
  \linethickness{.075mm}
  \multiput(-1,-1)(2,0){2}{\line(0,1){2}}
  \multiput(-1,-1)(0,2){2}{\line(1,0){2}}
  \linethickness{.3mm}
  \strecke{-0.8,-0.8}{0.8,-0.8}
  \strecke{0.8,-0.8}{-0.7,0.8}
  \strecke{-0.7,0.8}{-0.8,-0.8}
  \linethickness{.075mm}
  \strecke{0.8,-0.8}{-0.7,0.8}
  \strecke{-0.7,0.8}{-0.8,-0.8}
  \strecke{0.8,-0.8}{-0.5,0.8}
  \strecke{-0.5,0.8}{-0.8,-0.8}
  \strecke{0.8,-0.8}{-0.3,0.8}
  \strecke{-0.3,0.8}{-0.8,-0.8}
  \strecke{0.8,-0.8}{-0.1,0.8}
  \strecke{-0.1,0.8}{-0.8,-0.8}
  \strecke{0.8,-0.8}{0.1,0.8}
  \strecke{0.1,0.8}{-0.8,-0.8}
  \strecke{0.8,-0.8}{0.3,0.8}
  \strecke{0.3,0.8}{-0.8,-0.8}
  \strecke{0.8,-0.8}{0.5,0.8}
  \strecke{0.5,0.8}{-0.8,-0.8}
  \strecke{0.8,-0.8}{0.7,0.8}
  \strecke{0.7,0.8}{-0.8,-0.8}
  \strecke{0.8,-0.8}{0.9,0.8}
  \strecke{0.9,0.8}{-0.8,-0.8}
\end{picture}
```

For example, the command `\strecke{(0.8,-0.8)}{(0.3,0.8)}` invokes the LaTeX 2_ε command

- `\qbezier(0.8,-0.8)(0.8,-0.8)(0.3,0.8)`,

which cannot be replaced by a LaTeX 2_ε `\line` command as the direction vector $(-5, 16)$ would not be accepted.

By this "abuse" of the `\qbezier` command, one of the two severe drawbacks of the original `picture` environment can be comfortably circumvented. As the `\linethickness{`*thickness*`}` command applies to the `\qbezier` command, line segments of arbitrary slope and thickness can be drawn. What else would you want?

2.1.12 Multiple use of predefined partial pictures

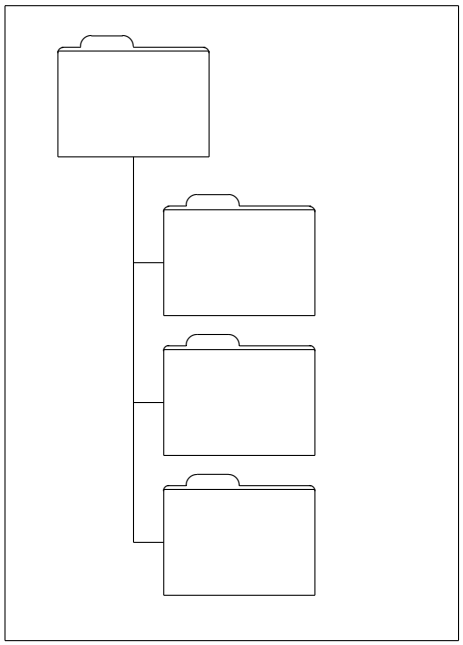


Fig. 14: Multiple use of predefined partial pictures

```
\setlength{\unitlength}{.5mm}
\begin{picture}(120, 168)
\newsavebox{\foldera}%           declaration
\savebox{\foldera}(40, 32)[l]{%  definition
  \multiput(0, 0)(0, 28){2}{\line(1, 0){40}}
  \multiput(0, 0)(40, 0){2}{\line(0, 1){28}}
  \put(1, 28){\oval(2, 2)[tl]}
  \put(1, 29){\line(1, 0){5}}
  \put(9, 29){\oval(6, 6)[tl]}
  \put(9, 32){\line(1, 0){8}}
  \put(17, 29){\oval(6, 6)[tr]}
  \put(20, 29){\line(1, 0){19}}
  \put(39, 28){\oval(2, 2)[tr]}
}
\newsavebox{\folderb}%           declaration
\savebox{\folderb}(40, 32)[l]{%  definition
  \put(0, 14){\line(1, 0){8}}
  \put(8, 0){\usebox{\foldera}}
}
\multiput(0, 0)(120, 0){2}{\line(0, 1){168}}
\multiput(0, 0)(0, 168){2}{\line(1, 0){120}}
\put(34, 26){\line(0, 1){102}}
\put(14, 128){\usebox{\foldera}}
\multiput(34, 86)(0, -37){3}{\usebox{\folderb}}
\end{picture}
```

A partial picture can be *declared* and *defined* by the commands

- `\newsavebox{`*name*`}` (declaration)
- `\savebox{`*name*`}(`*size*`)[`*position*`]{`*content*`}` (definition)

and then be arbitrarily often *used* by the command

- `\usebox{`*name*`}`.

The *name* argument refers to a LaTeX storage bin and therefore is of a command nature (which accounts for the backlashes in the current example). *size* is a numeric pair referring to the current value of `\unitlength`.

An object defined with the `\savebox` command (such as `\foldera` in the current example) can be used in the definition of another such element (`\folderb`).

The `\oval` command was applied because the connecting segments are two short to be drawn with the `line` command.

2.2 Positioning `picture` environments

2.2.1 Correcting positions

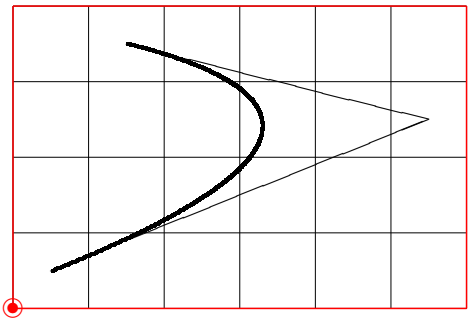


Fig. 15

```
\setlength{\unitlength}{1cm}
\begin{picture}(6, 4)
  \linethickness{0.075mm}
  \multiput(0, 0)(1, 0){7}{\line(0, 1){4}}
  \multiput(0, 0)(0, 1){5}{\line(1, 0){6}}
  \color{red}
  \put(0, 0){\circle*{.15}}
  \put(0, 0){\circle{.25}}
  \linethickness{0.2mm}
  \multiput(0, 0)(6, 0){2}{\line(0, 1){4}}
  \multiput(0, 0)(0, 4){2}{\line(1, 0){6}}
  \color{black}
  \put(.5, .5){\line(5, 2){5}}
  \put(5.5, 2.5){\line(-4, 1){4}}
  \linethickness{.5mm}
  \qbezier(.5, .5)(5.5, 2.5)(1.5, 3.5)
\end{picture}
```

Fig. 15 again demonstrates that the `\linethickness` command also applies to quadratic Bézier curves (while it does neither apply to a `\circle` nor to the arcs of an `\oval`).

We now change `\begin{picture}(6, 4)` to

- `\begin{picture}(6, 4)(-0.5, -1.25)`,

thus assigning the coordinates $(-0.5, -1.25)$ to the bottom left corner of the rectangle reserved by `\begin{picture}(6, 4)` (drawn in red color). Such a reassignment may be very convenient if you do not want to change all the coordinates of a picture which is not optimally located at the outset.

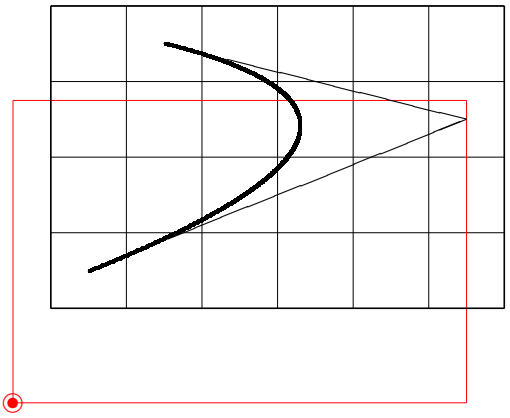


Fig. 16

```
\setlength{\unitlength}{1cm}
\begin{picture}(6, 4)(-0.5, -1.25)
  \linethickness{0.075mm}
  \multiput(0, 0)(1, 0){7}{\line(0, 1){4}}
  \multiput(0, 0)(0, 1){5}{\line(1, 0){6}}
  \color{red}
  \put(-0.5, -1.25){\circle*{.15}}
  \put(-0.5, -1.25){\circle{.25}}
  \multiput(-0.5, -1.25)(6, 0){2}
    {\line(0, 1){4}}
  \multiput(-0.5, -1.25)(0, 4){2}
    {\line(1, 0){6}}
  \color{black}
  \linethickness{0.2mm}
  \multiput(0, 0)(6, 0){2}{\line(0, 1){4}}
  \multiput(0, 0)(0, 4){2}{\line(1, 0){6}}
  \put(.5, .5){\line(5, 2){5}}
  \put(5.5, 2.5){\line(-4, 1){4}}
  \linethickness{.5mm}
  \qbezier(.5, .5)(5.5, 2.5)(1.5, 3.5)
\end{picture}
```

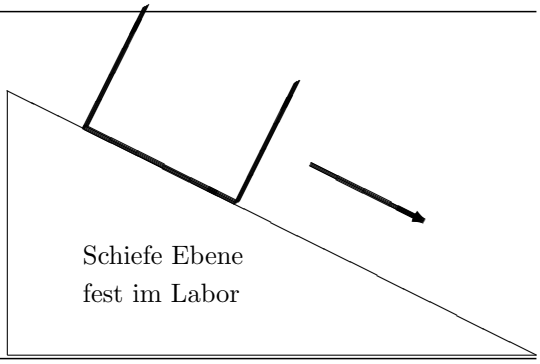
2.2.2 Side by side: text and picture

The strip between the horizontal lines:

Ein Behälter, mit Wasser teilweise gefüllt, gleitet reibungsfrei auf einer schiefen Ebene (siehe Zeichnung rechts). Die Wasseroberfläche sei stationär (keine Wellen); wie sieht die Wasseroberfläche aus? Äquivalenzprinzip. Sorgfältige Argumentation! Diskutiere die Kräfte

(1) parallel zur schiefen Ebene,

(2) normal zur schiefen Ebene.



---

is produced by the LaTeX source:

```
\begin{minipage}{7cm}
Ein Beh\"alter, mit Wasser teilweise gef\"ullt, gleitet reibungsfrei
auf einer schiefen Ebene (siehe Zeichnung rechts). Die Wasseroberfl\"ache
sei sta\-tio\-n\"ar (keine Wellen); wie sieht die Wasseroberfl\"ache aus?
\"Aquivalenzprinzip. Sorgf\"altige Argumentation! Diskutiere die Kr\"afte
\begin{enumerate}
\item parallel zur schiefen Ebene,
\item normal zur schiefen Ebene.
\end{enumerate}
\end{minipage} \hfill \begin{minipage}{7cm}
\setlength{\unitlength}{1mm}
  \begin{picture}(70,45)
    \put(0,0){\line(1,0){70}}
    \put(0,0){\line(0,1){35}}
    \put(70,0){\line(-2,1){70}}
    \linethickness{.75mm}
    \multiput(10,30)(.1,.05){8}{\line(1,2){8}}
    \multiput(30,20)(.1,.05){8}{\line(1,2){8}}
    \multiput(10,30)(.05,.1){6}{\line(2,-1){20}}
    \multiput(40,25)(.05,.1){6}{\vector(2,-1){15}}
    \put(10,12){Schiefe Ebene}
    \put(10,7){fest im Labor}
  \end{picture}
\end{minipage}
```

The picture illustrates, by the way, how oblique line segments can be drawn with arbitrary thickness by the use of the `\multiput` command.

Using a `minipage` for the left half allows an `enumerate` environment which would not be possible in a `parbox`.

A change of the argument of `\begin{picture}` has the effect of moving the picture up and down with respect to the leftside text. If we change to

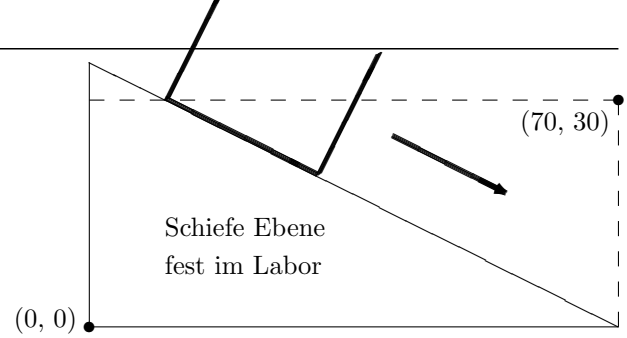
- `\begin{picture}(70, 30)`,

the effect is again shown in the strip between the horizontal lines:

Ein Behälter, mit Wasser teilweise gefüllt, gleitet reibungsfrei auf einer schiefen Ebene (siehe Zeichnung rechts). Die Wasseroberfläche sei stationär (keine Wellen); wie sieht die Wasseroberfläche aus? Äquivalenzprinzip. Sorgfältige Argumentation! Diskutiere die Kräfte

(1) parallel zur schiefen Ebene,

(2) normal zur schiefen Ebene.



The example proves that the space allocated to the picture is given by the `(70, 30)` argument, not by the real extension. If we change to

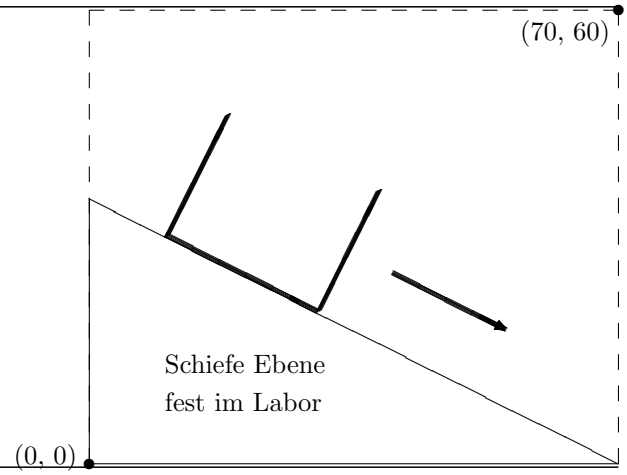
- `\begin{picture}(70, 60)`,

the picture is placed as if it had a size of `(70, 60)`:

Ein Behälter, mit Wasser teilweise gefüllt, gleitet reibungsfrei auf einer schiefen Ebene (siehe Zeichnung rechts). Die Wasseroberfläche sei stationär (keine Wellen); wie sieht die Wasseroberfläche aus? Äquivalenzprinzip. Sorgfältige Argumentation! Diskutiere die Kräfte

(1) parallel zur schiefen Ebene,

(2) normal zur schiefen Ebene.



The two minipages are centered vertically.

2.3 Applications

In this section, some "real life" examples of illustrations drawn in a `picture` environment are shown.

2.3.1 Falling mass

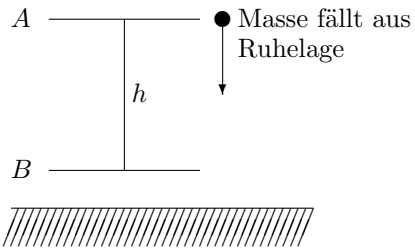


Fig. 17: Falling mass

```
\setlength{\unitlength}{1mm}
\begin{picture}(60,30)
  \put(0,29){$A$} \put(5,30){\line(1,0){20}}
  \put(28,30){\circle*{2}}
  \put(28,30){\vector(0,-1){10}}
  \put(30,29){Masse f\"allt aus}
  \put(30,25){Ruhelage}
  \put(0, 9){$B$}
  \put(5,10){\line(1,0){20}}
  \put(0,5){\line(1,0){40}}
  \multiput(1,5)(1,0){40}{\line(-2,-5){2}}
  \put(15,10){\line(0,1){20}} \put(16,19){$h$}
\end{picture}
```

2.3.2 Simultaneousness

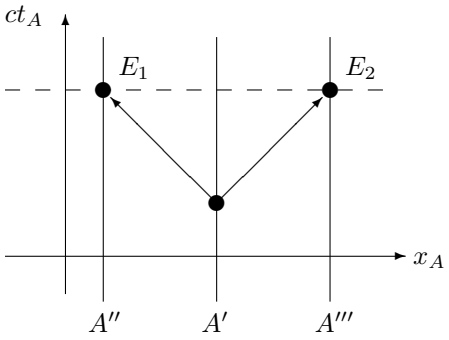


Fig. 18: Simultaneousness

```
\setlength{\unitlength}{1mm}
\begin{picture}(60,50)
  \put(0 ,15){\vector(1,0){53}}
  \put(54,14){$x_A$}
  \put( 8,10){\vector(0,1){37}}
  \put( 0,46){$ct_A$}
  \multiput(13,9)(15,0){3}{\line(0,1){35}}
  \put(11,5){$A''$}
  \put(26,5){$A'$}
  \put(41,5){$A'''$}
  \multiput(13,37)(30,0){2}{\circle*{2}}
  \put(28,22){\circle*{2}}
  \put(28,22){\vector(1,1){14}}
  \put(15,39){$E_1$}
  \put(28,22){\vector(-1,1){14}}
  \put(45,39){$E_2$}
  \multiput(0,37)(4,0){13}{\line(1,0){2}}
\end{picture}
```

2.3.3 Moving light clock

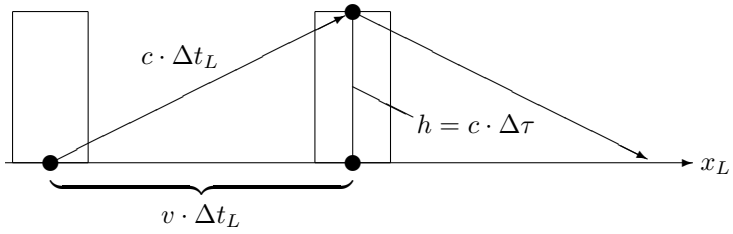


Fig. 19: Moving light clock

```
\setlength{\unitlength}{1mm}
\begin{picture}(96,38)
  \put(0,12){\vector(1,0){91}}
  \put(92,11){$x_L$}
  \put(6,10){$\underbrace{\rule{4cm}{0cm}}$}
  \put(26,5){\makebox(0,0){$v\cdot\Delta t_L$}}
  \multiput(1,12)(0,20){2}{\line(1,0){10}}
  \multiput(1,12)(10,0){2}{\line(0,1){20}}
  \multiput(41,12)(0,20){2}{\line(1,0){10}}
  \multiput(41,12)(10,0){2}{\line(0,1){20}}
  \multiput(6,12)(40,0){2}{\circle*{2}}
  \put(46,32){\circle*{2}}
  \put(46,12){\line(0,1){20}}
  \put(6,12){\vector(2,1){39}}
  \put(18,25){$c\cdot\Delta t_L$}
  \put(46,32){\vector(2,-1){39}}
  \put(46,22){\line(2,-1){8}}
  \put(54.5,16){$h=c\cdot\Delta\tau$}
\end{picture}
```

2.3.4 Rotation of axes

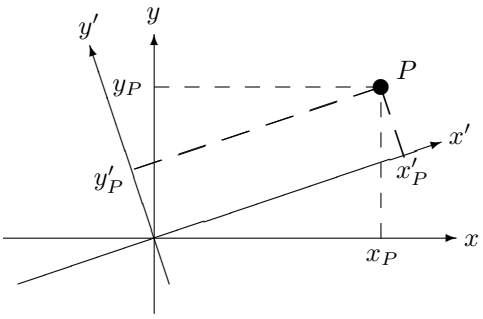


Fig. 20: Rotation of axes

```
\setlength{\unitlength}{1mm}
\begin{picture}(93,46)
  \put( 0,14){\vector(1,0){60}}
  \put(61,13){$x$}
  \put(20,4){\vector(0,1){37}}
  \put(19,43){$y$}
  \put(50,34){\circle*{2}}
  \put(52,35){$P$}
  \multiput(20,34)(4,0){8}{\line(1,0){2}}
  \put(14.5,33.5){$y_P$}
  \multiput(50,14)(0,4){5}{\line(0,1){2}}
  \put(48,11){$x_P$}
  \put( 2,8){\vector(3,1){56}}
  \put(59,26.5){$x'$}
  \multiput(50,34)(1.9,-5.7){2}
    {\line(1,-3){1.2}}
  \put(52,22){$x_P'$}
  \multiput(50,34)(-5.8,-1.933){6}
    {\line(-3,-1){3.6}}
  \put(12,21){$y_P'$}
  \put(22,8){\vector(-1,3){10.5}}
  \put(10,41){$y'$}
\end{picture}
```

2.3.5 Rapidity in the Special Theory of Relativity

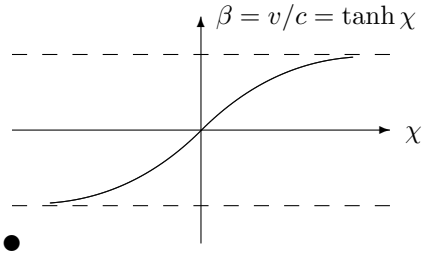


Fig. 21: Rapidity in the Special Theory of Relativity

Rapidity $\chi := \tanh^{-1}\dfrac{v}{c}$.

```
\setlength{\unitlength}{1cm}
\begin{picture}(6,4)(-2.5,-1.5)
  \put(-2.5,0){\vector(1,0){5}}
  \put(2.7,-0.1){$\chi$}
  \put(0,-1.5){\vector(0,1){3}}
  \multiput(-2.5,1)(0.4,0){13}
    {\line(1,0){0.2}}
  \multiput(-2.5,-1)(0.4,0){13}
    {\line(1,0){0.2}}
  \put(0.2,1.4)
    {$\beta=v/c=\tanh\chi$}
  \qbezier(0,0)(0.8853,0.8853)(2,0.9640)
  \qbezier(0,0)(-0.8853,-0.8853)(-2,-0.9640)
  \put(-2.5,-1.5){\circle*{0.2}}
\end{picture}
```

The intermediary control points of the two Bézier curves were calculated with formulas (1) in the section about quadratic Bézier curves. The positive branch is determined by $P_1 = (0,\, 0)$, $m_1 = 1$ and $P_2 = (2,\, \tanh 2)$, $m_2 = 1/\cosh^2 2$.

This example points out the usefullness of the second argument of the `\begin{picture}` command. The picture itself is defined in mathematically convenient coordinates, and the lower left corner is assigned the mathematical coordinates $(-2.5,\, -1.5)$ (black disk). If the second argument were omitted, it would default to $(0,\, 0)$; as a consequence, the origin of the graph would move to the center of the black disk.

2.3.6 Clocks in gravitational field

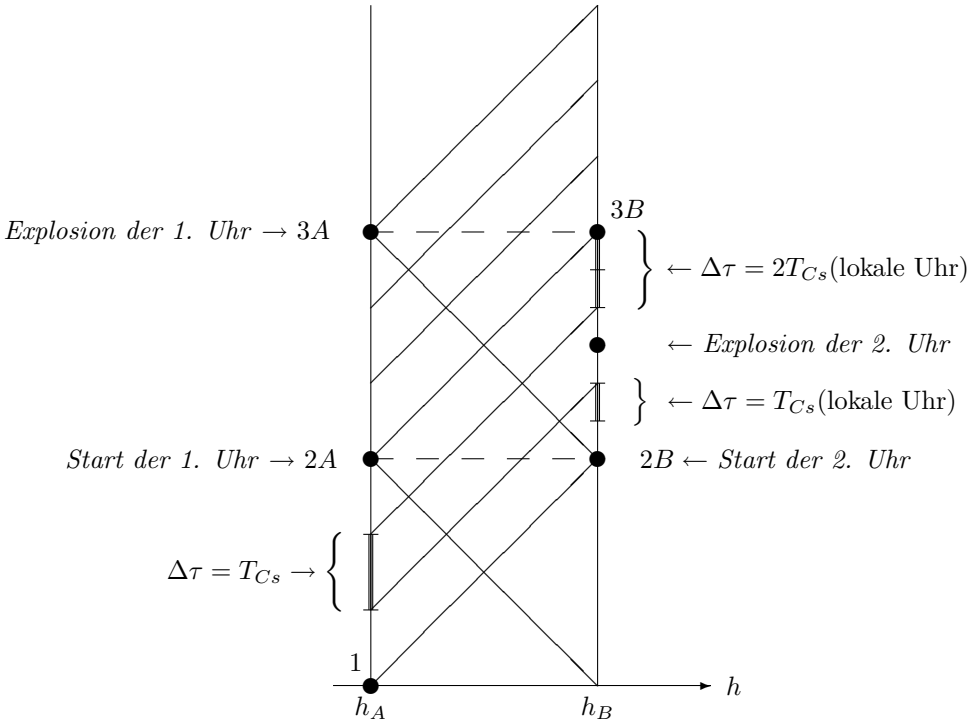


Fig. 22: Clocks in gravitational field

```
\setlength{\unitlength}{1mm}
\begin{picture}(120,100)(-8,0)
  \put(35,5){\vector(1,0){50}}
  \put(88,5){\makebox(0,0){$h$}}
  \put(40,2){\makebox(0,0){$h_A$}}
  \put(70,2){\makebox(0,0){$h_B$}}
  \multiput(40,5)(30,0){2}{\line(0,1){90}}
  \multiput(40,5)(0,10){7}{\line(1,1){30}}
  \multiput(40,35)(6,0){5}{\line(1,0){3}}
  \multiput(40,65)(6,0){5}{\line(1,0){3}}
  \multiput(40,5)(0,30){3}{\circle*{2}}
  \multiput(70,35)(0,15){3}{\circle*{2}}
  \multiput(40,35)(0,30){2}{\line(1,-1){30}}
  \multiput(39.75,15)(.25,0){3}{\line(0,1){10}}
  \multiput(39,15)(0,10){2}{\line(1,0){2}}
  \multiput(69.75,55)(.25,0){3}{\line(0,1){10}}
  \multiput(69.75,40)(.25,0){3}{\line(0,1){5}}
  \multiput(69,35)(0,5){7}{\line(1,0){2}}
  \put(97,60){\makebox(0,0){$
    \left\}\rule[-4mm]{0mm}{8mm}\right.\leftarrow
    \Delta\tau = 2T_{Cs}\mbox{(lokale Uhr)}
    $}}
  \put(98,50){\makebox(0,0){$\leftarrow$ \emph{Explosion der 2. Uhr}}}
  \put(96,42.5){\makebox(0,0){$\left\}\rule[-1.5mm]{0mm}{3mm}\right.
    \hskip1mm\leftarrow\Delta\tau = T_{Cs}\mbox{(lokale Uhr)}$}}
  \put(25,20){\makebox(0,0){$\Delta\tau = T_{Cs}\rightarrow
    \left\{\rule[-4mm]{0mm}{8mm}\right.$}}  %}
  \put(38,8){\makebox(0,0){1}}
  \put(17,35){\makebox(0,0){ \emph{Start der 1. Uhr} $\rightarrow$ 2$A$}}
  \put(13,65){\makebox(0,0)
              {\emph{Explosion der 1. Uhr} $\rightarrow$ 3$A$}}
  \put(93,35){\makebox(0,0)
              { 2$B \leftarrow$ \emph{Start der 2. Uhr}}}
  \put(74,68){\makebox(0,0){$3B$}}
\end{picture}
```

2.3.7 Line of simultaneousness

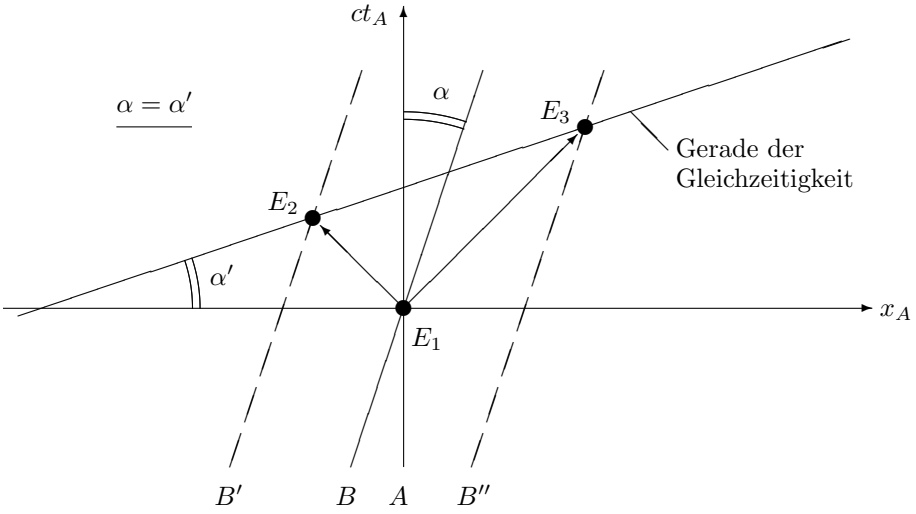


Fig. 23: Line of simultaneousness

```
\setlength{\unitlength}{1mm}
\begin{picture}(120,68)
  \put(0,28){\vector(1,0){115}} \put(116,27){$x_A$}
  \put(53,7){\vector(0,1){61}} \put(46,66){$ct_A$}
  \put(2,27){\line(3,1){110}}
  \multiput(53,28)(-12,12){2}{\circle*{2}}
  \put(77,52){\circle*{2}}
  \put(53,28){\vector(-1,1){11}} \put(53,28){\vector(1,1){23}}
  \put(54,23){$E_1$} \put(35,41){$E_2$} \put(71,53){$E_3$}
  \put(28,2){$B'$}   \multiput(30,7)(2,6){9}{\line(1,3){1.5}}
  \put(44,2){$B$}  \put(46,7){\line(1,3){17.5}}
  \put(51,2){$A$}
  \put(60,2){$B''$} \multiput(62,7)(2,6){9}{\line(1,3){1.5}}
  \put(83,54){\line(1,-1){5}}
  \put(89,48){Gerade der} \put(89,44){Gleichzeitigkeit}
  \put(15,54){$\alpha=\alpha'$}
  \put(15,52){\line(1,0){10}}
  % arc P1 = (54.0/31.0)  P2 = (53.0/28.0)  P3 = (53.0/29.0)  r = 25.0
  \qbezier(60.9057, 51.7171)(57.0569, 53.0)(53.0, 53.0)
  % arc P1 = (54.0/31.0)  P2 = (53.0/28.0)  P3 = (53.0/29.0)  r = 26.0
  \qbezier(61.2219, 52.6658)(57.2192, 54.0)(53.0, 54.0)
  % arc P1 = (6.0/28.0)  P2 = (5.0/28.0)  P3 = (8.0/29.0)  r = 20.0
  \qbezier(25.0, 28.0)(25.0, 31.2456)(23.9737, 34.3246)
  % arc P1 = (6.0/28.0)  P2 = (5.0/28.0)  P3 = (8.0/29.0)  r = 21.0
  \qbezier(26.0, 28.0)(26.0, 31.4078)(24.9223, 34.6408)
  \put(58.000,56.500){\makebox(0,0){$\alpha$}}
  \put(29.000,32.408){\makebox(0,0){$\alpha'$}}
\end{picture}
```

The inserts in the input file drawing the angle markings are produced, according to Section 2.1.8, with the Java program `arc` (see appendix for the source file `arc.java`).

2.3.8 Isotropic spherical waves

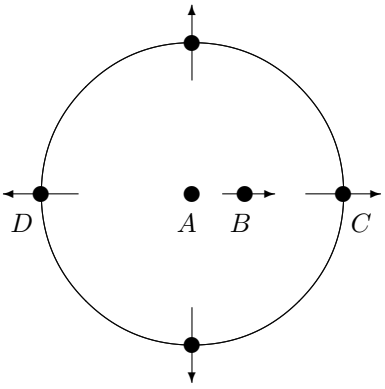


Fig. 24: Isotropic spherical waves

```
\setlength{\unitlength}{1mm}
\begin{picture}(50,55)
% Ellipse:  u = 25.0  v = 30.0
%           a = 20.0  b = 20.0  phi = 0.0 Grad
\qbezier(45.0, 30.0)(45.0, 38.2843)
  (39.1421, 44.1421)
\qbezier(39.1421, 44.1421)(33.2843, 50.0)
  (25.0, 50.0)
\qbezier(25.0, 50.0)(16.7157, 50.0)
  (10.8579, 44.1421)
\qbezier(10.8579, 44.1421)(5.0, 38.2843)
  (5.0, 30.0)
\qbezier(5.0, 30.0)(5.0, 21.7157)
  (10.8579, 15.8579)
\qbezier(10.8579, 15.8579)(16.7157, 10.0)
  (25.0, 10.0)
\qbezier(25.0, 10.0)(33.2843, 10.0)
  (39.1421, 15.8579)
\qbezier(39.1421, 15.8579)(45.0, 21.7157)
  (45.0, 30.0)
  \multiput(5,30)(20,0){3}{\circle*{2}}
  \multiput(25,10)(0,40){2}{\circle*{2}}
  \put(32,30){\circle*{2}}
  \put(29,30){\vector(1,0){7}}
  \put(10,30){\vector(-1,0){10}}
  \put(40,30){\vector(1,0){10}}
  \put(25,15){\vector(0,-1){10}}
  \put(25,45){\vector(0,1){10}}
  \put(1,25){$D$} \put(23,25){$A$}
  \put(30,25){$B$} \put(46,25){$C$}
\end{picture}
```

The circle is patched together from 8 quadratic Bézier curves, each covering a 45^0 angle. The `\qbezier` lines can be generated with the command

- `java bezierellipse 25 30 20 20`.

See the appendix for the source file `bezierellipse.java`.

2.4 Additional examples of quadratic Bézier curves

2.4.1 Catenary

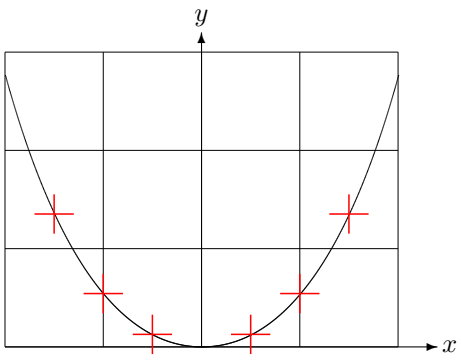


Fig. 25: Catenary

In Fig. 25, each symmetric half of the catenary $y = \cosh x - 1$ is approximated by a quadratic Bézier curve.

The right half of the curve ends in the point (2, 2.7622), the slope there having the value $m = 3.6269$. Using equation (1) in the section about quadratic Bézier curves, we can calculate the intermediate control points. They turn out to be (1.2384, 0) and (−1.2384, 0). The red crosses indicate points of the *real* catenary. The error is barely noticeable, being less than one percent.

```
\setlength{\unitlength}{1.3cm}
\begin{picture}(4.3, 3.6)(-2, 0)
  \put(-2, 0){\vector(1, 0){4.4}}
  \put(2.45, -.05){$x$}
  \put(0, 0){\vector(0, 1){3.2}}
  \put(0, 3.35){\makebox(0, 0){$y$}}
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(2.0/2.7622) m2=3.6269
  \qbezier(0.0, 0.0)(1.2384, 0.0)(2.0, 2.7622)
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(-2.0/2.7622) m2=-3.6269
  \qbezier(0.0, 0.0)(-1.2384, 0.0)(-2.0, 2.7622)
  \linethickness{.075mm}
  \multiput(-2, 0)(1, 0){5}{\line(0, 1){3}}
  \multiput(-2, 0)(0, 1){4}{\line(1, 0){4}}
  \color{red}
  \linethickness{.2mm}
  \put( .3, .12763){\line(1, 0){.4}}
  \put(.5, -.07237){\line(0, 1){.4}}
  \put(-.7, .12763){\line(1, 0){.4}}
  \put(-.5, -.07237){\line(0, 1){.4}}
  \put(.8,   .54308){\line(1, 0){.4}}
  \put(1,   .34308){\line(0, 1){.4}}
  \put(-1.2,  .54308){\line(1, 0){.4}}
  \put(-1,   .34308){\line(0, 1){.4}}
  \put(1.3,  1.35241){\line(1, 0){.4}}
  \put(1.5,  1.15241){\line(0, 1){.4}}
  \put(-1.7, 1.35241){\line(1, 0){.4}}
  \put(-1.5,  1.15241){\line(0, 1){.4}}
\end{picture}
```

Using the Java program `qbezier` (see appendix for the source file `qbezier.java`), we can generate the `qbezier` command lines of the input file with the command

- `java qbezier 0 0 0 2 2.7622 3.6269`,
- `java qbezier 0 0 0 -2 2.7622 -3.6269`

and insert them into the LaTeX file.

2.4.2 Forces on the catenary

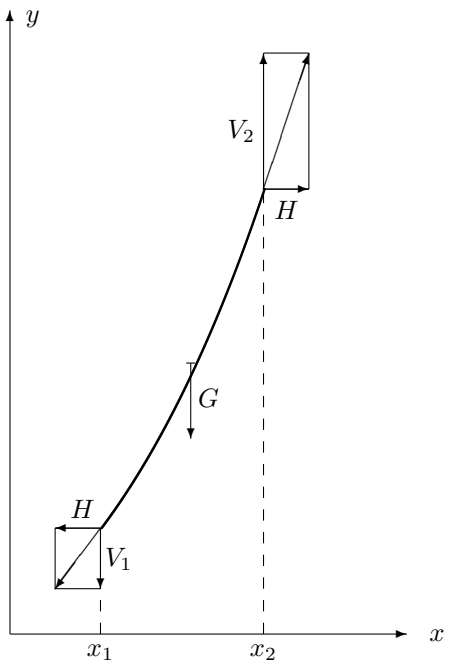


Fig. 26: Forces on the catenary

As the picture originally was too close to the caption, "Fig. 26: Forces on the catenary", it was pushed up somewhat by the optional argument `(0, -0.1)` of the `picture` environment.

```
\setlength{\unitlength}{3cm}
\begin{picture}(1.75, 2.75)(0, -0.1)
  \put(0,0){\vector(1,0){1.75}}
  \put(1.85, -0.03){$x$}
  \put(0,0){\vector(0,1){2.75}}
  \put(0.07, 2.70){$y$}
  \thicklines
  \qbezier(0.399, 0.467)(0.797, 0.998)
          (1.118, 1.962)
  \thinlines
  \multiput(0.399, 0)(0, 0.1){3}
           {\line(0,1){0.05}}
  \multiput(1.118, 0)(0, 0.1){20}
           {\line(0,1){0.05}}
  \put(0.399, -0.08){\makebox(0,0){$x_1$}}
  \put(1.118, -0.08){\makebox(0,0){$x_2$}}
  \multiput(0.399, 0.467)(0, -0.267){2}
           {\line(-1, 0){0.2}}
  \multiput(0.399, 0.467)(-0.2, 0){2}
           {\line(0, -1){0.267}}
  \put(0.399, 0.467){\vector(-1, 0){0.2}}
  \put(0.399, 0.467){\vector(0, -1){0.267}}
  \put(0.399, 0.467){\vector(-3, -4){0.2}}
  \put(0.32,0.55){\makebox(0,0){$H$}}
  \put(0.48, 0.33){\makebox(0,0){$V_1$}}
  \multiput(1.118, 1.962)(0, 0.6){2}
           {\line(1,0){0.2}}
  \multiput(1.118, 1.962)(0.2, 0){2}
           {\line(0,1){0.6}}
  \put(1.118, 1.962){\vector(1,0){0.2}}
  \put(1.118, 1.962){\vector(0,1){0.6}}
  \put(1.118, 1.962){\vector(1,3){0.2}}
  \put(1.22, 1.87){\makebox(0,0){$H$}}
  \put(1.02, 2.22){\makebox(0,0){$V_2$}}
  \put(0.797, 1.195){\vector(0,-1){0.333}}
  \put(0.777, 1.195){\line(1,0){0.04}}
  \put(0.83,1.0){$G$}
\end{picture}
```

2.4.3 Catenaries of constant length

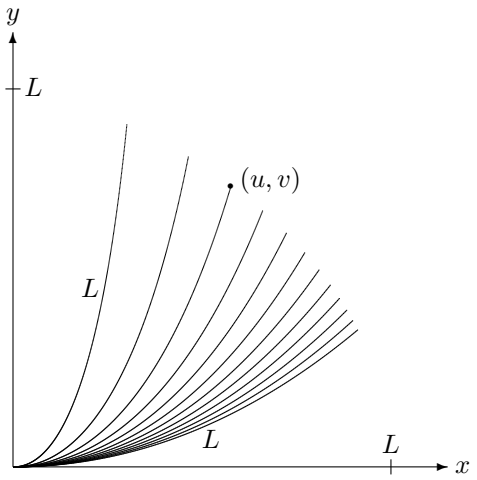


Fig. 27: Catenaries of constant length

From the equation

$$\frac{y}{a} = \cosh\frac{x}{a} - 1$$

of the catenary, it is seen that all catenaries can be obtained by scaling the curve $y = \cosh x - 1$ with factor a. A catenary of length L having its vertex in the origin and ending in the point $(u,\, v)$ satisfies (see `www.vsmp.ch`, Bulletin nr. 87) the equations

$$\begin{cases} v &= \sqrt{L^2 + a^2} - a, \\ u &= \dfrac{L^2 - v^2}{2v} \ln \dfrac{L + v}{L - v}, \\ m &= \sinh\dfrac{u}{a}. \end{cases}$$

From these equations, for $L = 1$ we get:

a	u	v	m
0.1	0.2998	0.9050	10
0.2	0.4625	0.8198	5
0.3	0.5757	0.7440	3.3333
0.4	0.6589	0.6770	2.5
0.5	0.7218	0.6180	2
0.6	0.7703	0.5662	1.6667
0.7	0.8081	0.5207	1.4286
0.8	0.8381	0.4806	1.25
0.9	0.8620	0.4454	1.1111
1	0.8814	0.4142	1
1.1	0.8972	0.3866	0.9091
1.2	0.9102	0.3620	0.8333

```
\setlength{\unitlength}{5cm}
\begin{picture}(1.2, 1.3)
  \put(0, 0){\vector(1, 0){1.15}}
  \put(1.17, -.015){$x$}
  \put(0, 0){\vector(0, 1){1.15}}
  \put(0, 1.19){\makebox(0, 0){$y$}}
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(0.2998/0.905) m2=10.0
  \qbezier(0.0,0.0)(0.2093,0.0)(0.2998,0.905)
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(0.4625/0.8198) m2=5.0
  \qbezier(0.0,0.0)(0.2985,0.0)(0.4625,0.8198)
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(0.5757/0.744) m2=3.3333
  \qbezier(0.0,0.0)(0.3525,0.0)(0.5757,0.744)
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(0.6589/0.677) m2=2.5
  \qbezier(0.0,0.0)(0.3881,0.0)(0.6589,0.677)
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(0.7218/0.618) m2=2.0
  \qbezier(0.0,0.0)(0.4128,0.0)(0.7218,0.618)
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(0.7703/0.5662) m2=1.6667
  \qbezier(0.0,0.0)(0.4306,0.0)(0.7703,0.5662)
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(0.8081/0.5207) m2=1.4286
  \qbezier(0.0,0.0)(0.4436,0.0)(0.8081,0.5207)
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(0.8381/0.4806) m2=1.25
  \qbezier(0.0,0.0)(0.4536,0.0)(0.8381,0.4806)
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(0.862/0.4454) m2=1.1111
  \qbezier(0.0,0.0)(0.4611,0.0)(0.862,0.4454)
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(0.8814/0.4142) m2=1.0
  \qbezier(0.0,0.0)(0.4672,0.0)(0.8814,0.4142)
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(0.8972/0.3866) m2=0.9091
  \qbezier(0.0,0.0)(0.4719,0.0)(0.8972,0.3866)
  % qbezier P1=(0.0/0.0) m1=0.0
  %         P2=(0.9102/0.362) m2=0.8333
  \qbezier(0.0,0.0)(0.4758,0.0)(0.9102,0.362)
  \put(0.5757, 0.744){\circle*{.015}}
  \put(0.6,    0.74){$(u,v)$}
  \put(0.5,    0.05){$L$}
  \put(0.18,    0.45){$L$}
  \put(1, -.02){\line(0, 1){.04}}
  \put(1, .06){\makebox(0, 0){$L$}}
  \put(-.02, 1){\line(1, 0){.04}}
  \put(.03, .98){$L$}
\end{picture}
```

Using equation (1) in the section about quadratic Bézier curves, we can calculate the intermediate control points. Or we can, by using the Java program `qbezier` described there (see appendix for the

source file `qbezier.java`), produce the `\qbezier` command lines and insert them into the LaTeX file.

2.4.4 Circles and ellipses

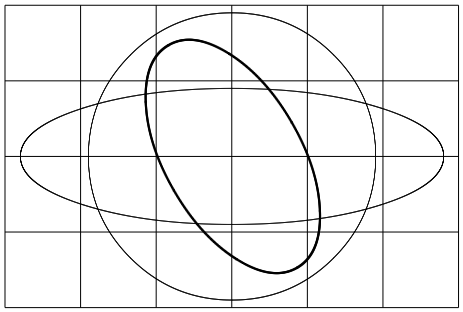


Fig. 28: Circles and ellipses

The circle ($a = b = 1.9$) was built from 8 arcs, each of 45 degrees, and each approximated by a quadratic Bézier curve. Circles and ellipses can then be obtained by rotating, scaling and translating.

Thus, the building block of all the circles and ellipses is the quadratic Bézier curve determined by the control points

$$\begin{cases} P_1 &= (1/0), \\ P_2 &= (1/\sqrt{2}-1), \\ P_3 &= \left(\frac{1}{2}\sqrt{2}\Big/\frac{1}{2}\sqrt{2}\right): \end{cases} \qquad (2)$$

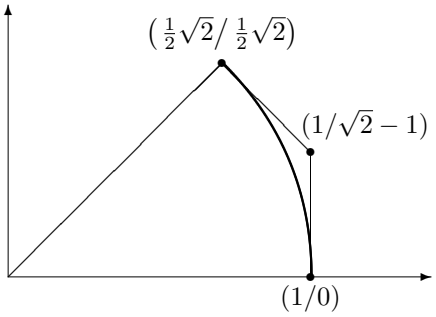


Fig. 29: 45 degrees arc

```
\setlength{\unitlength}{1cm}
\begin{picture}(6, 4)
  \linethickness{0.075mm}
  \multiput(0, 0)(1, 0){7}{\line(0, 1){4}}
  \multiput(0, 0)(0, 1){5}{\line(1, 0){6}}
  % Ellipse:  u=3.0  v=2.0  a=1.9  b=1.9
  %           phi=0.0 Grad
  \qbezier(4.9,2.0)(4.9,2.787)(4.3435,3.3435)
  \qbezier(4.3435,3.3435)(3.787,3.9)(3.0,3.9)
  \qbezier(3.0,3.9)(2.213,3.9)(1.6565,3.3435)
  \qbezier(1.6565,3.3435)(1.1,2.787)(1.1,2.0)
  \qbezier(1.1,2.0)(1.1,1.213)(1.6565,0.6565)
  \qbezier(1.6565,0.6565)(2.213,0.1)(3.0,0.1)
  \qbezier(3.0,0.1)(3.787,0.1)(4.3435,0.6565)
  \qbezier(4.3435,0.6565)(4.9,1.213)(4.9,2.0)
  % Ellipse:  u=3.0  v=2.0  a=2.8  b=0.9
  %           phi=0.0 Grad
  \qbezier(5.8,2.0)(5.8,2.3728)(4.9799,2.6364)
  \qbezier(4.9799,2.6364)(4.1598,2.9)(3.0,2.9)
  \qbezier(3.0,2.9)(1.8402,2.9)(1.0201,2.6364)
  \qbezier(1.0201,2.6364)(0.2,2.3728)(0.2,2.0)
  \qbezier(0.2,2.0)(0.2,1.6272)(1.0201,1.3636)
  \qbezier(1.0201,1.3636)(1.8402,1.1)(3.0,1.1)
  \qbezier(3.0,1.1)(4.1598,1.1)(4.9799,1.3636)
  \qbezier(4.9799,1.3636)(5.8,1.6272)(5.8,2.0)
  \thicklines
  % Ellipse:  u=3.0  v=2.0  a=0.9  b=1.7
  %           phi=30.0 Grad
  \qbezier(3.7794,2.45)(3.4273,3.0598)
          (2.9501,3.3592)
  \qbezier(2.9501,3.3592)(2.4728,3.6586)
          (2.15,3.4722)
  \qbezier(2.15,3.4722)(1.8272,3.2858)
          (1.8478,2.7228)
  \qbezier(1.8478,2.7228)(1.8685,2.1598)
          (2.2206,1.55)
  \qbezier(2.2206,1.55)(2.5727,0.9402)
          (3.0499,0.6408)
  \qbezier(3.0499,0.6408)(3.5272,0.3414)
          (3.85,0.5278)
  \qbezier(3.85,0.5278)(4.1728,0.7142)
          (4.1522,1.2772)
  \qbezier(4.1522,1.2772)(4.1315,1.8402)
          (3.7794,2.45)
  \end{picture}
```

The quadratic Bézier curve with control points (2) is given by the equation

$$\mathbf{r}(t) = (1-t)^2\begin{pmatrix}1\\0\end{pmatrix} + 2(1-t)t\begin{pmatrix}1\\\sqrt{2}-1\end{pmatrix} + t^2\begin{pmatrix}\frac{1}{2}\sqrt{2}\\\frac{1}{2}\sqrt{2}\end{pmatrix}.$$

From this, we get:

$$||\mathbf{r}(t)||^2 = 1 + (10 - 7\sqrt{2})(1-t)^2t^2, \qquad (3)$$

$$\left\|\mathbf{r}\left(\frac{1}{2}\right)\right\|^2 = \frac{1}{4}\sqrt{26-7\sqrt{2}} \approx 1.003\,135.$$

The error amounts to approximately 0.3 percent. From (3), we further conclude that $||\mathbf{r}(t)|| > 1$ for $0 < t < 1$. Letting $t = s + \frac{1}{2}$, we can see the symmetry with respect to $t = \frac{1}{2}$:

$$||\mathbf{r}(s)||^2 = 1 + (10 - 7\sqrt{2})\left(s^2 - \frac{1}{4}\right)^2 \qquad \left(-\frac{1}{2} \le s \le \frac{1}{2}\right).$$

The Java program `bezierellipse` (see appendix for the source file `bezierellipse.java`) generates a file, `ellipse.tex`, which can be pasted into a `picture` environment. Running the program with

- `java bezierellipse <u> <v> <a> <b> <phi>`

produces a file representing the ellipse with center (u/v) and radii a und b, and rotated by `<phi>` degrees. Thus, the `\qbezier` lines in the present example can be produced by invoking

- `java bezierellipse 3 2 1.9 1.9 0`
- `java bezierellipse 3 2 2.8 0.9 0`
- `java bezierellipse 3 2 0.9 1.7 30`

2.5 Enhancing the `picture` environment with packages `epic` und `eepic`

The packages `epic` and `eepic` enhance the `picture` environment. They are extensively described in Goossens, Mittelbach, Samarin[1].

2.5.1 Two applications of `epic`: `matrixput` and `putfile`

The two following examples require the command `\usepackage{epic}` in the preamble of the LaTeX file.

The `\matrixput` command

The `\matrixput` command is the natural generalization of the `\multiput` command to two dimensions.

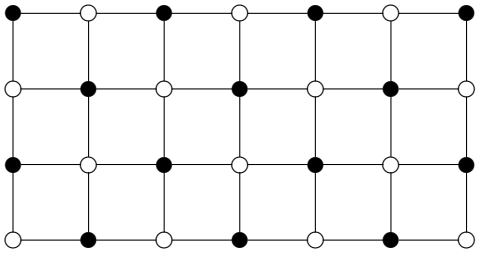


Fig. 30

```
\setlength{\unitlength}{1mm}
\begin{picture}(60, 30)
  \matrixput(0,  0)(20,0){4}(0,20){2}
            {\circle{2}}
  \matrixput(10,10)(20,0){3}(0,20){2}
            {\circle{2}}
  \matrixput(10, 0)(20,0){3}(0,20){2}
            {\circle*{2}}
  \matrixput(0, 10)(20,0){4}(0,20){2}
            {\circle*{2}}
  \matrixput(1,  0)(10,0){6}(0,10){4}
            {\line(1,0){8}}
  \matrixput(0,  1)(10,0){7}(0,10){3}
            {\line(0,1){8}}
\end{picture}
```

The `\putfile` command

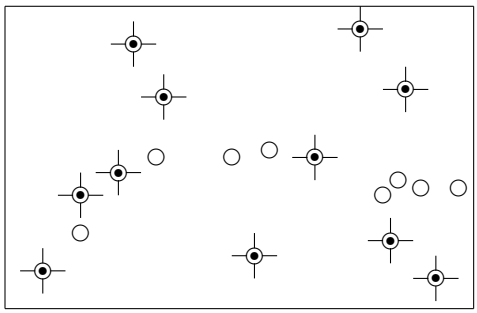


Fig. 31

```
\setlength{\unitlength}{1mm}
\begin{picture}(62, 40)
\newsavebox{\punkt}
\savebox{\punkt}(0, 0)[l]{%
  \put(0, 0){\circle*{1}}
  \put(0, 0){\circle{2}}
  \multiput(-3, 0)(4, 0){2}{\line(1, 0){2}}
  \multiput(0, -3)(0, 4){2}{\line(0, 1){2}}
}
  \multiput(0, 0)(62, 0){2}{\line(0, 1){40}}
  \multiput(0, 0)(0, 40){2}{\line(1, 0){62}}
  \putfile{data/datafile1}{\circle{2}}
  \putfile{data/datafile2}{\usebox{\punkt}}
\end{picture}
```

The slash in the command `\putfile{data/datafile1}` may have to be replaced by the backslash on a Windows system. The two files `datafile1` and `datafile2` contain one coordinate pair on each line:

```
datafile1: | datafile2:
     10 10 | 5 5
     20 20 | 10 15
     30 20 | 15 18
     35 21 | 17 35
     50 15 | 21 28
     55 16 | 33 7
     52 17 | 41 20
     60 16 | 47 37
           | 51 9
           | 53 29
           | 57 4
```

2.5.2 Line segments and circles in the `eepic` package

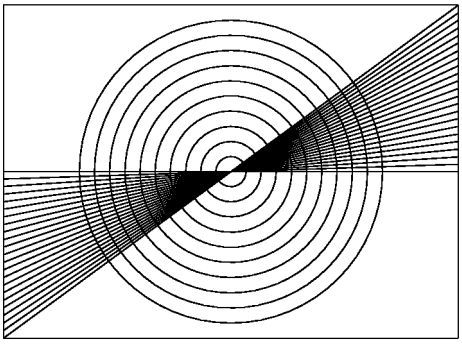


Fig. 32: Line segments and circles in the `eepic` package

This example requires the command `\usepackage{eepic}` in the preamble of the LaTeX file.

With the use of the `eepic` package, line segments of any slope (defined by two integers), and circles of any radius can be drawn within the `picture` environment.

Remark. `pdflatex` cannot deal directly with commands of the `eepic` package. The picture was generated with the commands

`latex` | `dvips` | `ps2pdf`

and subsequently converted into a `png` picture. This was then imported with the `includegraphics` command.

```
\setlength{\unitlength}{1mm}
\begin{picture}(60, 44)
  \multiput(0, 0)(60, 0){2}{\line(0, 1){44}}
  \multiput(0, 0)(0, 44){2}{\line(1, 0){60}}
  % requires eepic
  \put(30, 22){\circle{40}}
  \put(30, 22){\circle{36}}
  ......................
  ......................
  \put(30, 22){\circle{8}}
  \put(30, 22){\circle{4}}
  % direction vectors must consist of
  % integers
  \put(30, 22){\line(30, 0){30}}
  \put(30, 22){\line(30, 1){30}}
  \put(30, 22){\line(30, 2){30}}
  ..........................
  ..........................
  \put(30, 22){\line(30, 20){30}}
  \put(30, 22){\line(30, 21){30}}
  \put(30, 22){\line(30, 22){30}}
  % direction vectors must consist of
  % integers
  \put(30, 22){\line(-30, -0){30}}
  \put(30, 22){\line(-30, -1){30}}
  \put(30, 22){\line(-30, -2){30}}
  ..........................
  ..........................
  \put(30, 22){\line(-30, -20){30}}
  \put(30, 22){\line(-30, -21){30}}
  \put(30, 22){\line(-30, -22){30}}
\end{picture}
```

3 The pspicture Environment of the pstricks Package

Unlike epic and eepic, the pstricks package brings with it its own environment, pspicture, with drawing commands different from those of the picture environment. A detailed description can be found in Goossens, Rahtz, Mittelbach[2].

Remark. pdflatex cannot deal directly with commands of the pstricks package. The pictures in Section 3 were generated with the commands

latex | dvips | ps2pdf

and subsequently converted into png pictures. These were then imported with the \includegraphics command.

3.1 Line segments and circles

3.1.1 Regular polygon

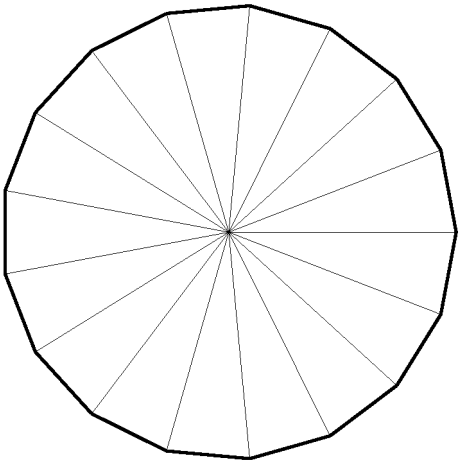


Fig. 33: Regular polygon

This example illustrates how line segments of arbitrary slopes can be drawn within the pspicture environment. The command \psset{unit=3cm} sets the length unit to 3 cm. (The pstricks default is 1 cm.) The arguments $(0, 0)$ and $(2, 2)$ of \begin{pspicture} determine the upper left corner and the lower right corner of the picture. Line width ist set by

\psset{linewidth=1pt}

for the periphery, and by

\psset{linewidth=.1pt}

for the diagonals.

```
\psset{unit=3cm}
\begin{pspicture}(0, 0)(2, 2)
  \psset{linewidth=1pt}
  \psline(2.0000,1.0000)(1.9325,1.3612)
  \psline(1.9325,1.3612)(1.7390,1.6737)
  \psline(1.7390,1.6737)(1.4457,1.8952)
  \psline(1.4457,1.8952)(1.0923,1.9957)
  \psline(1.0923,1.9957)(0.7263,1.9618)
  \psline(0.7263,1.9618)(0.3974,1.7980)
  \psline(0.3974,1.7980)(0.1498,1.5264)
  \psline(0.1498,1.5264)(0.0170,1.1837)
  \psline(0.0170,1.1837)(0.0170,0.8163)
  \psline(0.0170,0.8163)(0.1498,0.4736)
  \psline(0.1498,0.4736)(0.3974,0.2020)
  \psline(0.3974,0.2020)(0.7263,0.0382)
  \psline(0.7263,0.0382)(1.0923,0.0043)
  \psline(1.0923,0.0043)(1.4457,0.1048)
  \psline(1.4457,0.1048)(1.7390,0.3263)
  \psline(1.7390,0.3263)(1.9325,0.6388)
  \psline(1.9325,0.6388)(2.0000,1.0000)
  \psset{linewidth=.1pt}
  \psline(2.0000,1.0000)(1, 1)
  \psline(1.9325,1.3612)(1, 1)
  \psline(1.7390,1.6737)(1, 1)
  \psline(1.4457,1.8952)(1, 1)
  \psline(1.0923,1.9957)(1, 1)
  \psline(0.7263,1.9618)(1, 1)
  \psline(0.3974,1.7980)(1, 1)
  \psline(0.1498,1.5264)(1, 1)
  \psline(0.0170,1.1837)(1, 1)
  \psline(0.0170,0.8163)(1, 1)
  \psline(0.1498,0.4736)(1, 1)
  \psline(0.3974,0.2020)(1, 1)
  \psline(0.7263,0.0382)(1, 1)
  \psline(1.0923,0.0043)(1, 1)
  \psline(1.4457,0.1048)(1, 1)
  \psline(1.7390,0.3263)(1, 1)
  \psline(1.9325,0.6388)(1, 1)
\end{pspicture}
```

The points $P_i \quad (i = 0, \ldots, 16)$ are determined by

$$\begin{cases} x_i &= 1 + \cos\left(i \cdot \dfrac{2\pi}{17}\right), \\ x_i &= 1 + \sin\left(i \cdot \dfrac{2\pi}{17}\right). \end{cases}$$

3.1.2 Triangle

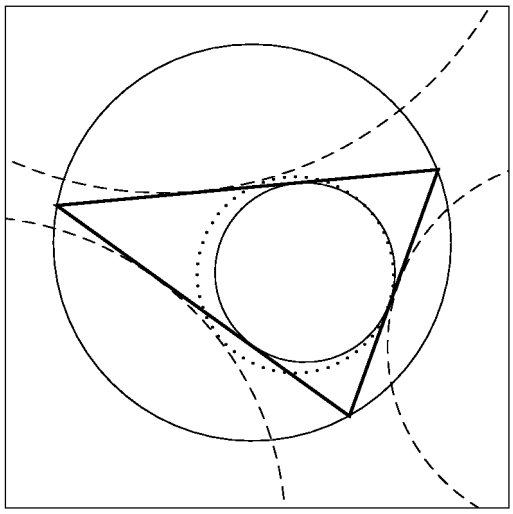


Fig. 34: Triangle

```
\psset{unit=.45pt}
\begin{pspicture}*(420, 420)
  \psset{linewidth=.4pt}
  % Frame
  \pspolygon(0, 0)(420, 0)(420, 420)(0, 420)
  % Umkreis und Inkreis
  \pscircle(206, 222){166.18}
  \pscircle(250, 197){75.42}
  % Ankreise
  \pscircle[linestyle=dashed](476, 135){157.68}
  \pscircle[linestyle=dashed](-32, -22){266.53}
  \pscircle[linestyle=dashed](132, 579){315.92}
  % triangle
  \psset{linewidth=1pt}
  \pspolygon(43, 253)(361, 283)(287, 77)
  % nine-points circle (Feuerbach)
  \pscircle[linestyle=dotted](242, 195){83.09}
\end{pspicture}
```

This example illustrates that `pstricks` allows the drawing of circles with arbitrary radii. The $*$ after {`pspicture`} causes the picture to be clipped to the frame determined by `(420, 420)`. The upper left corner not being mentioned, it defaults to $(0, 0)$.

In the picture, a triangle is drawn with the inscribed and the circumscribed circle (solid), the three adjacent circles (dashed), and the nine point (Feuerbach) circle (dotted).

3.1.3 Loops and calculations

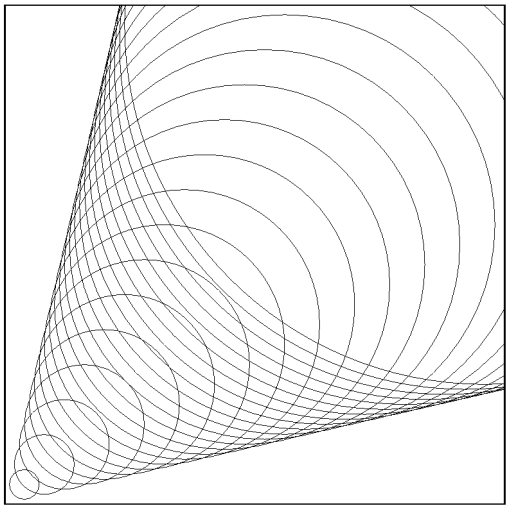


Fig. 35: Loops and calculations

```
\newcounter{i} \setcounter{i}{1}
\newcounter{j} \setcounter{j}{0}
\newcounter{r} \setcounter{r}{0}
\usepackage{pstricks}
\usepackage{ifthen}
\usepackage{calc}
\psset{unit=.66mm}
\begin{document}
\begin{pspicture}*(100, 100)
  \pspolygon(0, 0)(100, 0)(100, 100)(0, 100)
  \psset{linewidth=.1pt}
  \whiledo{\value{i}<25}{%
    \setcounter{j}{\value{i}*4}
    \setcounter{r}{\value{i}*3}
    \pscircle(\value{j},\value{j}){\value{r}}
    \stepcounter{i}
  }
\end{pspicture}
\end{document}
```

As this example shows, with the `calc` and the `ifthen` packages, LaTeX assumes the capability of running through loops, taking decisions, and doing calculations. The $*$ after {`pspicture`} causes the picture to be clipped to the frame determined by `(100, 100)`. The upper left corner not being mentioned, it defaults to $(0, 0)$.

3.1.4 A remark about `latex2html`

Unlike the `latex` command, the `latex2html` command expects all `\psset` commands to be placed *within* the `pspicture` environment. Otherwise, `\psset` is considered unknown. Thus the arguments of the `\begin{pspicture}` command always have to be given in the default unit, which is 1 cm.

3.2 More applications

The following examples are taken from Goossens, Rahtz, Mittelbach[2], with slight variations. They are intended to give some hints as to the versatility of the `pstricks` package, and the related packages,

`pstcol`, `pst-grad`, `pst-text`, `pst-node`, `pst-tree`.

While the `latex2html` command is capable of processing `pstricks` commands, it cannot process `pstcol`, `pst-grad`, `pst-text`, `pst-node`, or `pst-tree` commands. Thus, we get the following overview:

	latex	latex2html	pdflatex
epic	+	+	+
eepic	+	+	-
pstricks	+	+	-
pstcol,...	+	-	-

3.2.1 Tribox

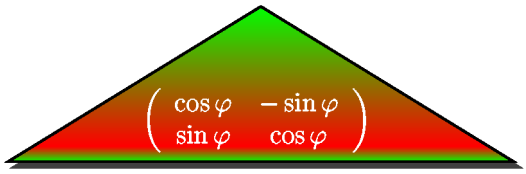


Fig. 36: Tribox

```
\usepackage{pstricks, pstcol, pst-grad}

\pstribox[shadow=true, fillstyle=gradient,
          gradbegin=green, gradend=red]
  {\color{white}$\left(\begin{array}{cc}
    \cos\varphi & -\sin\varphi \\
    \sin\varphi &  \cos\varphi
    \end{array}\right)$
  }
```

3.2.2 Curved text

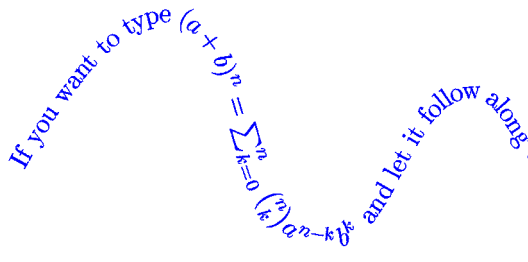


Fig. 37: Curved text

```
\usepackage{pstricks, pstcol, pst-text}

\begin{pspicture}(-4, -3.2)(3, 0.2)
  \psset{linecolor=white}     % hidden
  \pstextpath
    {\pscurve(-4, -2)(-2, 0)(0, -3)(2, -1)%
      (3, -2)(5, -3)}
    {\color{blue}               % font color
      If you want to type
      $(a+b)^n=\sum_{k=0}^n {n\choose k}
       a^{n-k}b^k$ and let it follow along
       a curving line \ldots}
\end{pspicture}
```

3.2.3 Logo

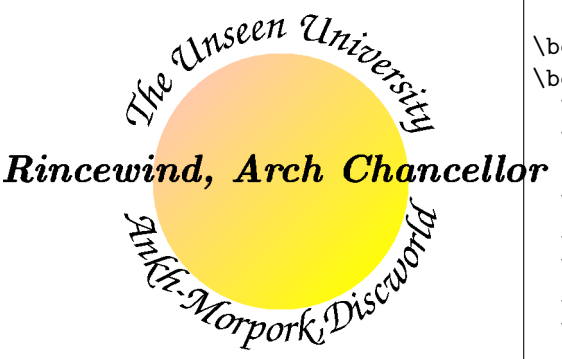


Fig. 38: Logo

```
\usepackage{pstricks,
            pstcol, pst-text, pst-grad}
\definecolor{pink}{rgb}{1, .75, .8}

\begin{document}
\begin{pspicture}(-3, -2.2)(3, 2.2)
  \psset{linestyle=none}
  \newcommand{\curly}[1]{{\fontfamily{pzc}%
    \fontsize{17}{17}\itshape#1}}
  \pstextpath[c]{\psarcn(0, 0){2}{180}{0}}
  {\curly{The Unseen University}}
  \pstextpath[c]{\psarc(0, 0){2}{180}{0}}
  {\curly{Ankh-Morpork,Discworld}}
  \pscircle[fillstyle=gradient, gradangle=45,
    gradbegin=pink, gradend=yellow](0, 0){1.7}
  \rput[B](0, 0){{\Large\itshape\bfseries
    Rincewind, Arch Chancellor
  }}
\end{pspicture}
\end{document}
```

3.2.4 Knots

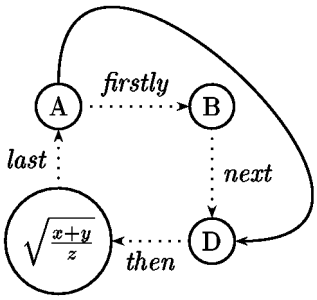


Fig. 39: Knots

```
\usepackage{pstricks, pst-node}
\psset{arrows=->, labelsep=3pt, mnode=circle}
\begin{document}
  \begin{psmatrix}[rowsep=20pt, colsep=28pt]
    A                       & B \\
    $\sqrt{\frac{x+y}{z}}$ & D
    \psset{linestyle=dotted}
    \ncline{1, 1}{1, 2}\naput{\emph{firstly}}
    \ncline{1, 2}{2, 2}\naput{\emph{next}}
    \ncline{2, 2}{2, 1}\naput{\emph{then}}
    \ncline{2, 1}{1, 1}\naput{\emph{last}}
    \nccurve[ncurv=2,
      linestyle=solid, angleA=90]{1, 1}{2, 2}
  \end{psmatrix}
\end{document}
```

3.2.5 Family tree

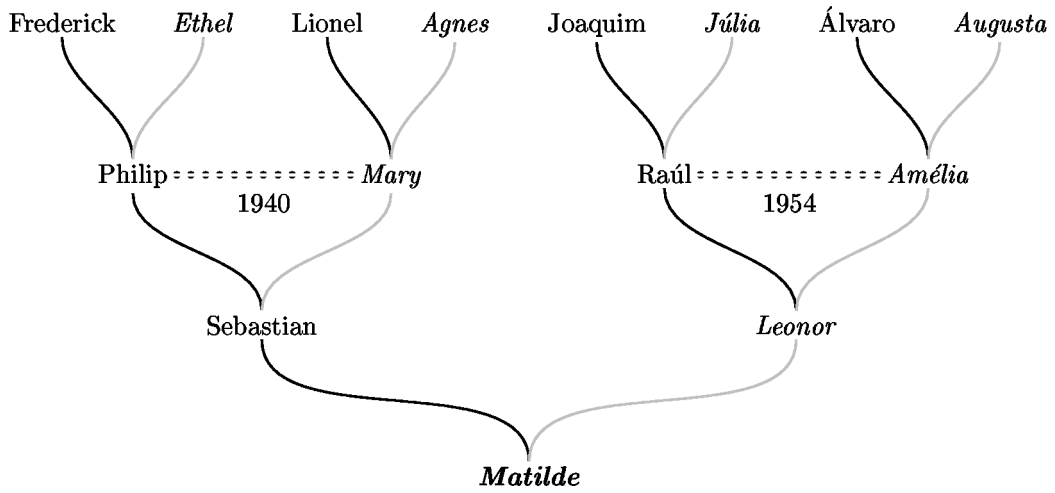


Fig. 40: Family tree

```
\usepackage{pstricks, pstcol, pst-node, pst-tree}
\renewcommand\psedge{\nccurve}
\newcommand{\Female}[2][]{{\psset{linecolor=lightgray}\TR[#1]{\emph{#2}}}}
\newcommand{\Male}[2][]{{\psset{linecolor=black}\TR[#1]{#2}}}
\psset{nodesep=2pt, angleA=90, angleB=-90}
\begin{document}
  \pstree[treemode=U]{\Female{{\bfseries Matilde}}}{%
    \pstree{\Male{Sebastian}}{%
      \pstree{\Male[name=P]{Philip}}{\Male{Frederick}\Female{Ethel}}
      \pstree{\Female[name=W]{Mary}}{\Male{Lionel}\Female{Agnes}}}
    \pstree{\Female{Leonor}}{%
      \pstree{\Male[name=R]{Ra\'ul}}{\Male{Joaquim}\Female{J\'ulia}}
      \pstree{\Female[name=A]{Am\'elia}}{\Male{\'Alvaro}\Female{Augusta}}}}
    \psset{doubleline=true, linestyle=dotted}
    \ncline{P}{W}\nbput{1940}
    \ncline{R}{A}\nbput{1954}
\end{document}
```

4 MetaPost

MetaPost is a twin of Donald Knuth's METAFONT, which was written for the purpose of generating new fonts. MetaPost was derived from METAFONT by John D. Hobby, who writes about it:

> MetaPost is a programming language much like Knuth's METAFONT except that it outputs PostScript programs instead bitmaps. Borrowed from METAFONT are the basic tools for creating and manipulating pictures. These include numbers, coordinate pairs, cubic splines, affine transformations, text strings, and boolean quantities. Additional features integrating text and graphics and accessing special features of PostScript such as clipping, shading, and dashed lines. Another feature borrowed from METAFONT is the ability to solve linear equations that are given implicitly, thus allowing many programs to be written in a largely declarative style. By building complex operations from simpler ones, MetaPost achieves both power and flexibility.
>
> MetaPost is particularly well-suited to generating Fig.s for technical documents where some aspects of a picture may be controlled by mathematical or geometrical constraints that are best expressed symbolically. In other words, MetaPost is not meant to take the place of a freehand drawing tool or even an interactive graphics editor.

A good way to start working with MetaPost is by using Hobby[3], whereas the complete background can be found in Knuth[5]. Hoenig[4] and Goossens, Rahtz, Mittelbach[2] also contain many valuable examples.
An online *tutorial* is found under `http://www.ursoswald.ch`.

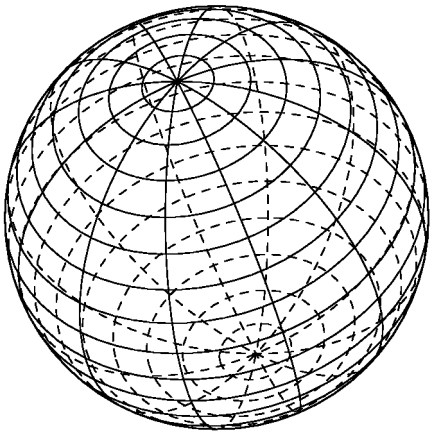


Fig. 41: Sphere

4.1 Examples

The following examples are meant to suggest the versatility of MetaPost.

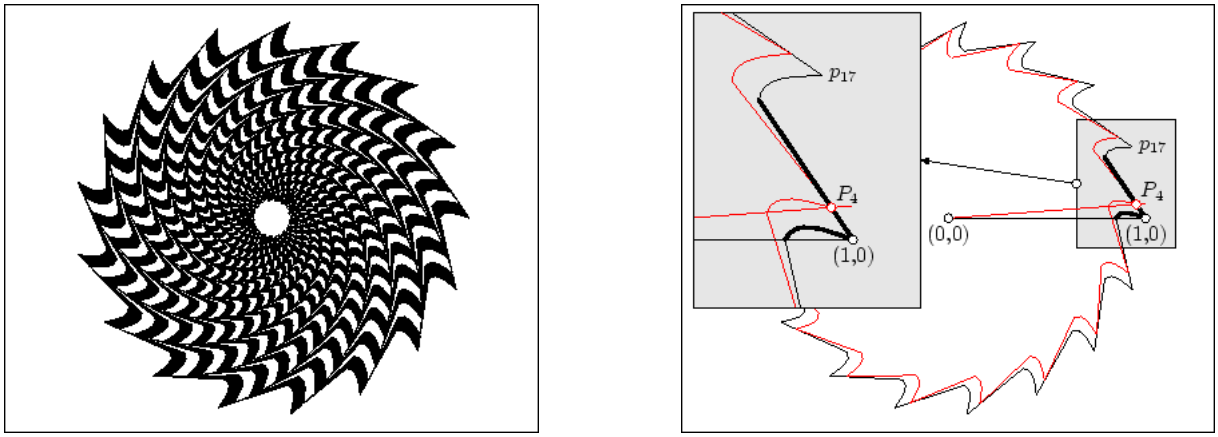


Fig. 42: Regular polygons

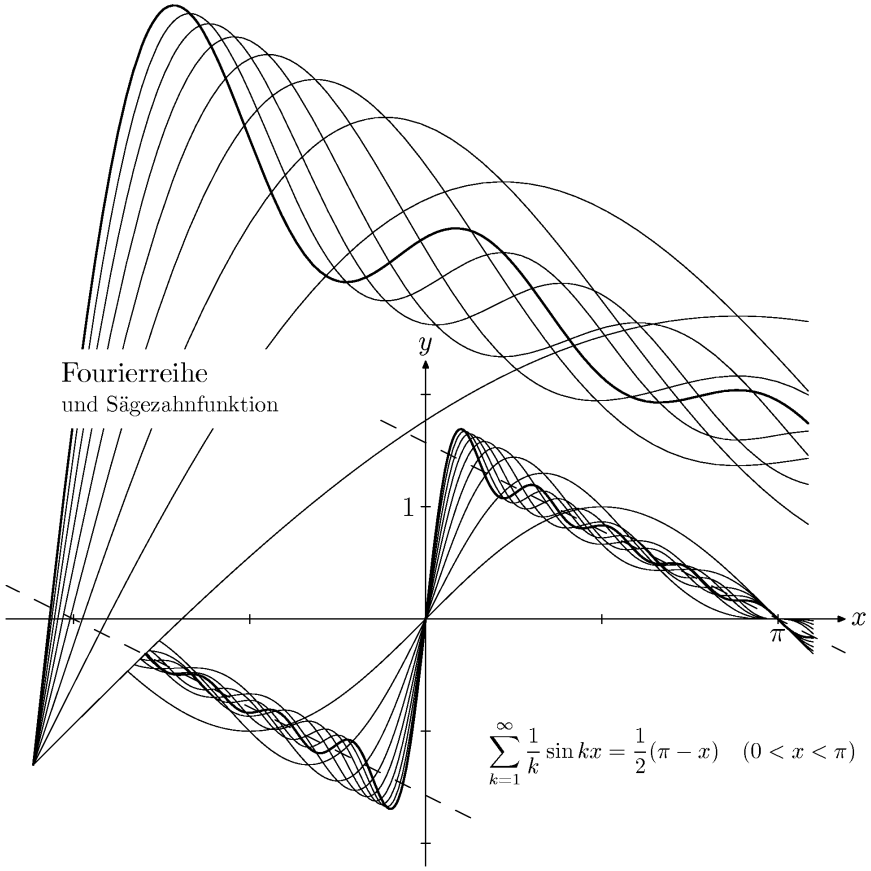


Fig. 43: Fourier series

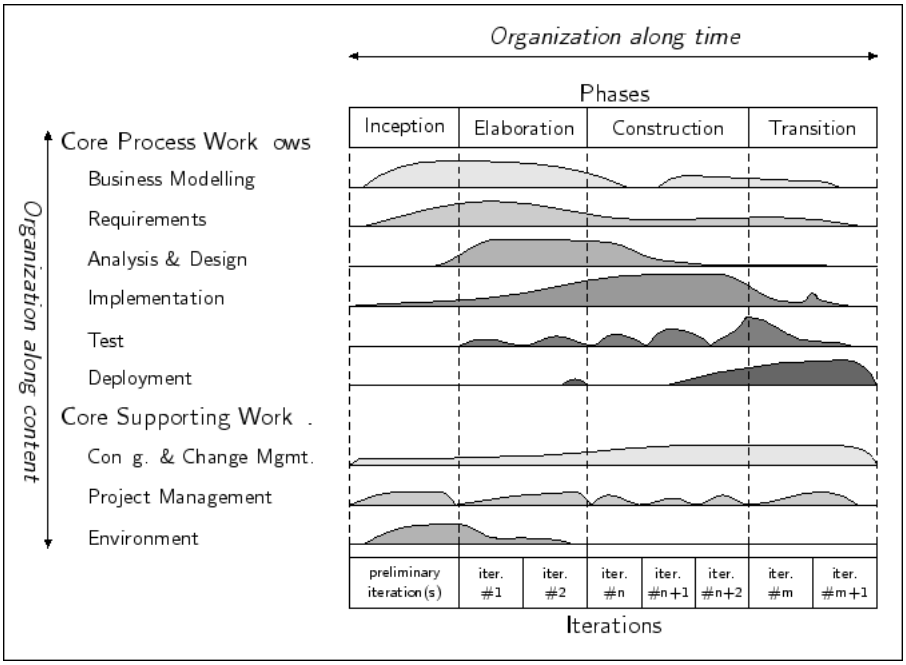


Fig. 44: Organization

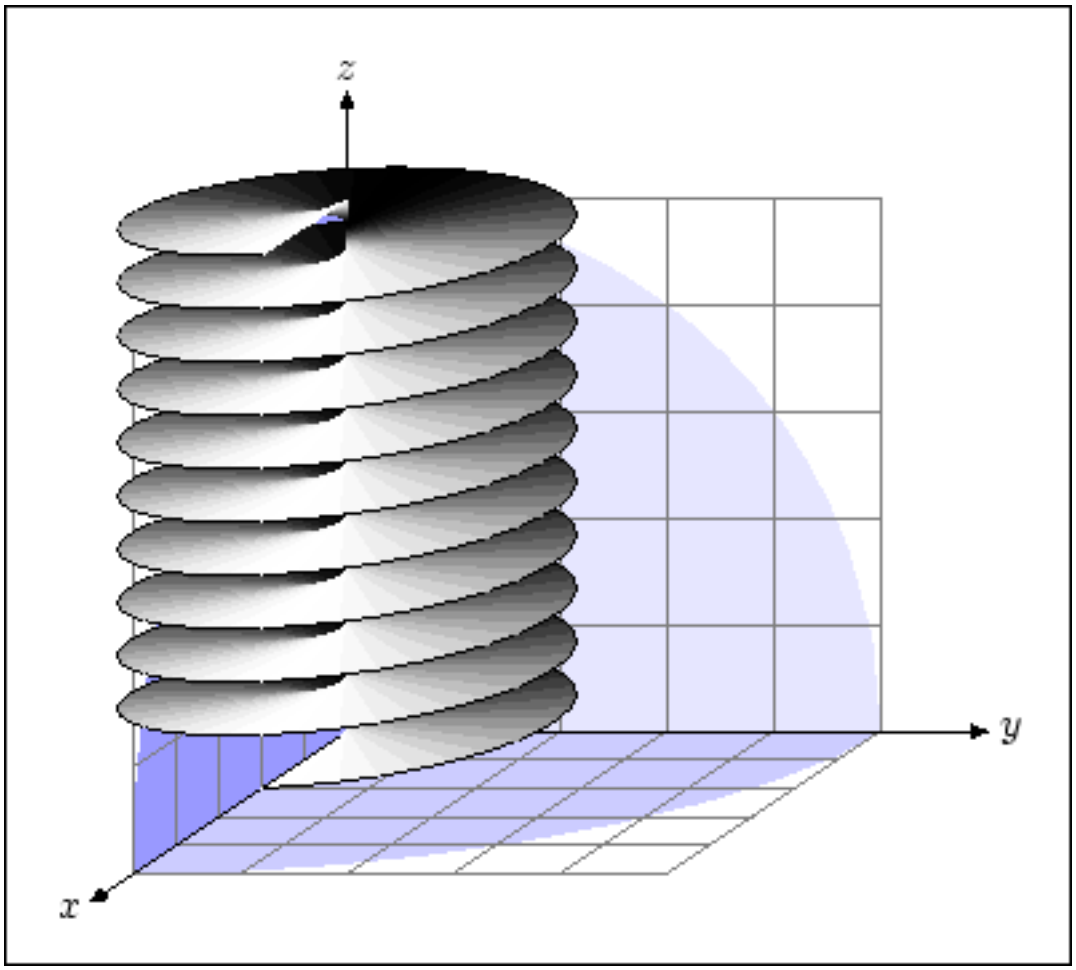


Fig. 45: Spiral

4.2 A source file

The file `Transform.mp` (see appendix) can give a taste of `MetaPost` programming. It generates the two pictures of Fig. 42. The section between `beginfig(1)` and the subsequent `endfig;` defines the left picture, and the section between `beginfig(2)` and the second `endfig;` defines the picture on the right.

On the command

- `mpost Transform`

(or `mpost Transform.mp`), `MetaPost` generates two files, `Transform.1` and `Transform.2`, which can be imported in a LaTeX document with the commands

- `\includegraphics{Transform.1}`,
- `\includegraphics{Transform.2}`.

The LaTeX file can be processed by both `latex` and `latex2html`. If it is to be processed by `pdflatex`, the line

- `\DeclareGraphicsRule{*}{mps}{*}{}`

must be entered in the preamble.

Remark. On certain systems, `mp` is to be used instead of `mpost`.

5 Appendix

A qbezier.java

```
import java.io.*;

class qbezier {

  static void main(String[] args) {
    if(args.length != 6) {
      System.out.println("Gebrauch: java qbezier x1 y1 m1 x2 y2 m2");
      return;
    }
    File outfile = new File("qbezier.tex");
    try {
      double x1 = Double.parseDouble(args[0]);
      double y1 = Double.parseDouble(args[1]);
      double m1 = Double.parseDouble(args[2]);
      double x2 = Double.parseDouble(args[3]);
      double y2 = Double.parseDouble(args[4]);
      double m2 = Double.parseDouble(args[5]);
      if(m1 == m2) {
        System.out.println("Keine Auswertung mglich:");
        System.out.println("m1 und m2 mssen verschieden sein");
        return;
      }
      String outstring = "% qbezier P1=("+x1+"/"+y1+") m1="+m1+" P2=("+x2+"/"+y2+") m2="+m2;
      outstring += "\n\\qbezier";
      double x = (m2*x2-m1*x1+y1-y2) / (m2-m1);
      double u = y1+m1*(x-x1);
      double v = y2+m2*(x-x2);
      double y = 0.5*(u+v);
      outstring += "(" + runde(x1) + ", " + runde(y1) + ")";
      outstring += "(" + runde(x)  + ", " + runde(y)  + ")";
      outstring += "(" + runde(x2) + ", " + runde(y2) + ")";
      FileWriter out = new FileWriter(outfile.getPath(), false);
      out.write(outstring);
      out.close();
    }
    catch(NumberFormatException e) {
      System.out.println("Eingabe nicht korrekt");
      System.out.println("Korrekte Eingabe: x1 y1 m1 x2 y2 m2");
    }
    catch(IOException e) {
    }
  }

  static String runde(double t) {
    String s = "";
    int T = (int)(Math.round(10000 * t));
    double d = T/10000.0;
    s = String.valueOf(d);
    return s;
  }
}
```

B bezierellipse.java

```
import java.io.*;

class bezierellipse {
  /*
   *  16.11.2002 Urs Oswald osurs@bluewin.ch
   *  erzeugt Datei "ellipse.tex"
   *  ueberschreibt allanfalls vorhandene Datei dieses Namens
   *  Ellipse wird aus 8 quadratischen Bezierkurven zusammengesetzt
   *  Gebrauch: java bezierellipse [u [v [a [b [phi]]]]]
   */
  static double[] uvabp  = new double[5];
  static double[] xx     = new double[16];
  static double[] yy     = new double[16];
  static double   sc45   = .5f * Math.sqrt(2);
  static{                                          // Einheitskreis
    uvabp[0] = 0.0;                                // u = 0
    uvabp[1] = 0.0;                                // v = 0
    uvabp[2] = 1.0;                                // a = 1
    uvabp[3] = 1.0;                                // b = 1
    uvabp[4] = 0.0;                                // phi = 0
  }

  static void main(String[] args) {
    /*
     *  args[0], args[1]:  Mittelpunkt (u, v)
     *  args[2], args[3]:  Halbachsen  a, b
     *  args[4]:           Rotationswinkel phi in Grad
     *
     */
    double[] xy;
    for(int i=0;i<5;i++)                           // Aufgrund des Arguments aendern:
      if(args.length > i)                          // Mittelpunkt (u,v)
uvabp[i] = Double.parseDouble(args[i]);   // Halbachsen  a, b und Rotation p

    String str = "% Ellipse:";
    str += "  u = " + uvabp[0];
    str += "  v = " + uvabp[1];
    str += "  a = " + uvabp[2];
    str += "  b = " + uvabp[3];
    str += "  phi = " + uvabp[4] + " Grad";

    uvabp[4] = Math.PI*uvabp[4]/180;               // Rotationswinkel in Radian
    xx[0] = 1; yy[0] = 0;                          // P0
    xx[1] = 1; yy[1] = Math.sqrt(2) - 1;           // P1

    for(int i=1;i<8;i++) {                         // Berechnung der Punkte P2,...,P15
      double phi = i * Math.PI / 4;                // des Einheitskreises
      xy = rotate(xx[0], yy[0], phi);
      xx[2*i] = xy[0]; yy[2*i] = xy[1];
      xy = rotate(xx[1], yy[1], phi);
      xx[2*i+1] = xy[0]; yy[2*i+1] = xy[1];
    }
    for(int i=0;i<16;i++) {                        // Streckung, Rotation, Verschiebung
      xx[i] = xx[i] * uvabp[2];                    // x-Streckung
      yy[i] = yy[i] * uvabp[3];                    // y-Streckung
```

```
      xy = rotate(xx[i], yy[i], uvabp[4]);        // Rotation
      xx[i] = xy[0] + uvabp[0];                   // Translation (u, 0)
      yy[i] = xy[1] + uvabp[1];                   // Translation (0, v)
    }
    for(int i=0;i<15;i+=2)
      str+="\n\\qbezier"
        +punkt(i)+punkt(i+1)+punkt((i+2)%16);     // Anfuegen der 8 Zeilen
      File outfile = new File("ellipse.tex");
      try {
outfile.createNewFile();                  // in "ellipse.tex" schreiben
FileWriter out =
  new FileWriter(outfile.getPath(), false);
out.write(str);
out.close();
      }
      catch(IOException e) {
      }
  }

  static double[] rotate(double x, double y, double phi) {
    /*
     *  phi im Bogenmass
     */
    double[] xy = new double[2];
    xy[0] = x * Math.cos(phi) - y * Math.sin(phi);
    xy[1] = x * Math.sin(phi) + y * Math.cos(phi);
    return xy;
  }
  static String runde(double t) {
    String s = "";
    int T = (int)(Math.round(10000 * t));
    double d = T/10000.0;
    s = String.valueOf(d);
    return s;
  }
  static String punkt(int i) {
    String s;
    s="(" + runde(xx[i]) + ", " + runde(yy[i]) + ")";
    return s;
  }
}
```

C arc.java

```
import java.io.*;

class arc {
  static double[] xx;
  static double[] yy;
  static double r;

  static void main(String[] args) {
    /*
     *  Urs Oswald osurs@bluewin.ch, 18.11.2002
     *  Eingabe: x1 y1 x2 y2 x3 y3 r
     *  schreibt arc.tex: LaTeX-Datei fuer Bogen
     *  mit Radius r im Winkel P1-P2-P3 (Scheitel: P2)
     *  ueberschreibt allfaellig schon vorhandene Datei desselben Namens
     */
    if(args.length < 7) {
      System.out.println("Schreibt arc.tex fr Bogen mit Radius r im Winkel P1-P2-P3");
      System.out.println("Gebrauch: java arc x1 y1 x2 y2 x3 y3 r");
      return;
    }
    String outstring = "";
    double a, rot;
    double[] xy;
    xx = new double[4];                          // P3: mittlerer (Kontroll-)Punkt
    yy = new double[4];
    xx[1] = Double.parseDouble(args[0]);         // P1 --> 1
    yy[1] = Double.parseDouble(args[1]);
    xx[0] = Double.parseDouble(args[2]);         // P2 --> 0
    yy[0] = Double.parseDouble(args[3]);
    xx[2] = Double.parseDouble(args[4]);         // P3 --> 2
    yy[2] = Double.parseDouble(args[5]);
    r    = Double.parseDouble(args[6]);          // r
    outstring += "% arc\n";
    outstring += "% P1 = ("+xx[1]+"/"+yy[1]+")";
    outstring += "  P2 = ("+xx[0]+"/"+yy[0]+")";
    outstring += "  P3 = ("+xx[2]+"/"+yy[2]+")";
    outstring += "  r = " + r ;
    xx[1] = xx[1] - xx[0]; yy[1] = yy[1] - yy[0]; // Verschiebung des Scheitels
    xx[2] = xx[2] - xx[0]; yy[2] = yy[2] - yy[0]; // in den Ursprung
    a = Math.sqrt(xx[1]*xx[1] + yy[1]*yy[1]);    // Einheitsvektor (xx[1], yy[1])
    xx[1] = xx[1] / a; yy[1] = yy[1] / a;
    a = Math.sqrt(xx[2]*xx[2] + yy[2]*yy[2]);    // Einheitsvektor (xx[2], yy[2])
    xx[2] = xx[2] / a; yy[2] = yy[2] / a;

    rot = 0.5*( angle(xx[1], yy[1])              // rot: Neigungswinkel der
              +angle(xx[2], yy[2]));             //      Winkelhalbierenden

    xy  = rotate(xx[1], yy[1], -rot);            // Rotation: Ziel ist
    xx[1] = xy[0]; yy[1] = xy[1];                // Symmetrie zur x-Achse
    xy  = rotate(xx[2], yy[2], -rot);
    xx[2] = xy[0]; yy[2] = xy[1];
    xx[3] = 1 / xx[1]; yy[3] = 0;                // qbezier-Punktfolge: P1 P3 P2
    for(int i=1;i<4;i++) {
      xy = rotate(xx[i], yy[i], rot);            // Zurueckrotieren
      xx[i] = r * xy[0] + xx[0];                 // Strecken und Zurueckschieben
```

```
      yy[i] = r * xy[1] + yy[0];
    }
    outstring += "\n\\qbezier"
      + punkt(1)+punkt(3)+punkt(2);               // Anfuegen der 3 Punkte
    File outfile = new File("arc.tex");           // in Datei schreiben
    try{
      FileWriter out =
        new FileWriter(outfile.getPath(), false); // ueberschreibt allfaellig schon
      out.write(outstring);                       // bestehende Datei "arc.tex"
      out.close();
    }
    catch(IOException e) {
    }
  }
  static double angle(double x, double y) {
    /*
     *   Resultat: Radian, 0 <= phi < 2PI
     */
    double phi = 0.0;
    double a = Math.sqrt(x*x + y*y);
    x = x/a;
    y = y/a;
    phi = Math.acos(x);
    if(y<0)
      phi = 2*Math.PI-phi;
    return phi;
  }
  static double[] rotate(double x, double y, double phi) {
    /*
     *  phi im Bogenmass
     */
    double[] xy = new double[2];
    xy[0] = x * Math.cos(phi) - y * Math.sin(phi);
    xy[1] = x * Math.sin(phi) + y * Math.cos(phi);
    return xy;
  }
  static String runde(double t) {
    String s = "";
    int T = (int)(Math.round(10000 * t));
    double d = T/10000.0;
    s = String.valueOf(d);
    return s;
  }
  static String punkt(int i) {
    String s;
    s="(" + runde(xx[i]) + ", " + runde(yy[i]) + ")";
    return s;
  }
}
```

D Transform.mp

```
u:=25;                    % 25 = 25bp = 25 PostScript points = 30/72 in
wi:=10;                   % width  in units u
he:=8;                    % height in units u
hoehe:=he*u;              % height
breite:=wi*u;             % width

R:=3.7;                   % maximum radius in units u
n:=17;                    % number of edges
phi:=360/(n*10);
phi:=360/(n*5);           % rotation angle

def draw_point(expr P, col) =
  unfill fullcircle scaled 1.5mm shifted P withcolor white;
  draw   fullcircle scaled 1.5mm shifted P withcolor col;
enddef;

transform t, Rot, T;
path q, p[];
pair P[];

Rot:=identity rotated(360/n);


% ----- Calculations in mathematical coordinates ----------------------------------
P1=(.85, 0);
P2=(1, 0);
P3=P1 transformed Rot;
p0:=P1{1, 2}..{3, -1}P2--P3;                    % p0: first path element
for i=1 upto n-1:                               % get remaining path elements
  p[i]:=p[i-1] transformed Rot;                  % p[1],..., p[n-1]
endfor                                          % by rotating p0
%
% path (n path elements)
%
p17:=p0 for i=1 upto n-1: &p[i] endfor ..cycle;   % put path elements together

q:=((0, 0)--(1,0)) rotated phi;

P4=p0 intersectionpoint q;
% ----- End of calculations in mathematical coordinates -----------------------------


%
% transform t maps mathematical coordinates on MetaPost coordinates
%
t:=identity scaled (R*u) shifted .5(breite, hoehe);


beginfig(1)    % ========================= figure 1 ==================================
  draw (0, 0)--(breite, 0)--(breite, hoehe)--(0, hoehe)--cycle;
  T:=identity;
  for i=0 upto 49:
    p98:=p17 transformed T transformed t;
    if i mod 2 = 0:
      fill p98;
```

```
    else:
      unfill p98;
    fi
    draw p98;
    T:=T zscaled P4;
  endfor
endfig;        % ====================== end of figure 1 ============================


beginfig(2)    % ======================== figure 2 ================================
  %
  % Box (in mathematical coordinates)
  %
  path kasten;
  kasten=(.65, -.15)--(1.15, -.15)--(1.15, .5)--(.65, .5)--cycle;
  z98=point 3.5 of kasten transformed t;
  fill kasten transformed t withcolor .9white;
  draw kasten transformed t;

  draw (0, 0)--(breite, 0)--(breite, hoehe)--(0, hoehe)--cycle;

  draw p17 transformed t;
  draw p17 zscaled P4 transformed t withcolor red;
  draw q rotated -phi transformed t;
  draw q transformed t withcolor red;

  pickup pencircle scaled 2;
  draw p0 transformed t;
  label.rt(btex $p_{99}$ etex, point 3 of p17 transformed t);

  pickup pencircle scaled .5;
  draw_point((0,0) transformed t, black);
  label.bot(btex (0,0) etex, (0,0) transformed t);

  draw_point((1,0) transformed t, black);
  label.bot(btex (1,0) etex, (1,0) transformed t);

  draw_point(P4 transformed t, red);
  label.urt(btex $P_4$ etex, P4 transformed t);

  %
  % renew definition of transform t in order ============
  % to draw the enlarged kasten at the left     ============
  %
  t:=identity scaled (2.3R*u) shifted (-.35breite-1.8u, .5hoehe-.4u);

  kasten:=kasten transformed t;
  fill kasten withcolor .9white;
  draw kasten;
  z99=point 1.5 of kasten;

  draw p0     transformed t;
  draw p1     transformed t cutafter  kasten;
  draw p[n-1] transformed t cutbefore kasten;

  draw p0     zscaled P4 transformed t withcolor red;
  draw p1     zscaled P4 transformed t cutafter  kasten withcolor red;
```

```
  draw p[n-1] zscaled P4 transformed t cutbefore kasten withcolor red;

  draw q rotated -phi transformed t cutbefore kasten;
  draw q transformed t cutbefore kasten withcolor red;

  pickup pencircle scaled 2;
  draw p0 transformed t;
  label.rt(btex $p_{99}$ etex, point 3 of p17 transformed t);

  pickup pencircle scaled .5;

  draw_point((1,0) transformed t, black);
  label.bot(btex (1,0) etex, (1,0) transformed t);

  draw_point(P4 transformed t, red);
  label.urt(btex $P_4$ etex, P4 transformed t);

  pickup pencircle scaled 1;
  drawarrow z98--z99;
  pickup pencircle scaled .5;
  draw_point(z98, black);


endfig;        % ====================== end of figure 2 ============================

end
```

References

[1] M. Goossens, F. Mittelbach, A. Samarin *The LaTeX Companion*

[2] M. Goossens, S. Rahtz, F. Mittelbach *The LaTeX Graphics Companion*

[3] John D. Hobby, *User's Manual*, `http://cm.bell-labs.com/who/hobby/`

[4] Alan Hoenig, *TeX Unbound*, OXFORD UNIVERSITY PRESS, 1998, ISBN 0-19-509685-1

[5] Donald E. Knuth. *The METAFONTbook*. ADDISON-WESLEY, 1992, ISBN 0-201-13445-4 und ISBN 0-201-13444-6 (soft)

[6] Leslie Lampert. *LaTeX – A Document Preparation System*. ADDISON-WESLEY, 1994 (13th Printing November 2001), ISBN 0201529831

[7] Helmut Kopka. *LaTeX – Eine Einführung*. ADDISON-WESLEY, 1992, ISBN 3-89319-434-7

[8] Helmut Kopka. *LaTeX – Erweiterungsmöglichkeiten*. ADDISON-WESLEY, 1992, ISBN 3-89319-356-1

[9] Keith Reckdahl, *Using Imported Graphics in LaTeX2e*,
`ftp://ftp.dante.de/tex-archive/info/epslatex.ps`,
`ftp://ftp.dante.de/tex-archive/info/epslatex.pdf`