Vorstellung der LaTeX-Pakete PGF/TikZ

Was ist PGF?

PGF ist ein Makropaket zur Erstellung von Grafiken. Es wird z. B. für Grafiken im Paket *beamer* benutzt.

- ⊕ plattform- und formatunabhängig, d. h. geeignet für plain (pdf)TeX, (pdf)LaTeXund ConTeXt
- ⊕ Kann PostScript oder PDF erstellen
- ⊕ Benutzerfreundliche Schnittstelle Ti*k*Z
- ⊕ Modularer Aufbau, zusätzliche Makros für z. B. Pfeilspitzen, Hintergründe usw. über zusätzliche Bibliotheken
- ⊕ Die Syntax ist an METAPOST und PSTricks angelehnt

Was ist TikZ?

Ti*k*Z ist *kein* Zeichenprogramm! Es ist die benutzfreundliche Schnittstelle zu PGF.

Paketquellen/Dokumentation

Autor	Till Tantau
Quelle	`http://sourceforge.net/projects/pgf/` oder
	`http://tug.ctan.org/tex-archive/graphics/pgf`
Doku	`http://tug.ctan.org/graphics/pgf/doc/generic/pgf/version-for-pdftex/en/pgfmanual.pdf`

Einbindung in ein LaTeX-Dokument

```
\usepackage{tikz}
```

Und bei Bedarf
`\usetikzlibrary{arrows,automata,backgrounds,calendar}` usw.

Allgemeine Syntax

Syntax von Koordinatenangaben

- Angabe in runden Klammern als x, y-Wertepaar (TeX-Dimensionen): `(1cm,2pt)`
- Oder in Form von Polarkoordinaten: `(30:1cm)`. Das bedeutet „1 cm in Richtung 30 Grad“
- Sind keine Einheiten angegeben `(1,2)`, dann wird PGFs xy-Koordinatensystem zugrunde gelegt. Das heißt 1 cm in x-Richtung und 2 cm in y-Richtung

- Relative Koordinaten: Angabe einmal als `++(1cm,0pt)`; bedeutet 1 cm nach rechts vom letzten Punkt aus. Gleichzeitig wird der aktuelle Punkt auf diese Koordinaten gesetzt.

  Oder Angabe als `+(1cm,0pt)`; dann wird der aktuelle (Referenzpunkt) nicht verändert.

Syntax von Pfadangaben

Hierbei macht Ti*k*Z von der METAPOST-Syntax Gebrauch.

- Am Anfang steht der Pfad, allgemein als `\path[⟨optArg⟩]`. Der Befehl bewirkt keine Aktion, er reserviert nur den Platz, den das Objekt gezeichnet einnehmen würde.

- Nach Ermitteln des Pfades können darauf weitere Aktionen angewandt werden. So kann man z. B. den Pfad zeichnen (*draw, stroke*), (einheitlich) füllen (*fill*), schattieren (*shade*), mit einem Muster füllen (*pattern*) oder ausschneiden (*clip*).

- Ti*k*Z stellt eher allgemeine Befehle bereit, die intensiv mit optionalen Argumenten (Key=Value-Syntax) versorgt werden können. Das folgende Beispiel zeichnet auf 5(!) verschiedene Arten einen Kreis.

```
\begin{tikzpicture}
 \path            (0,0) circle (2ex);
 \path[draw]      (1,0) circle (2ex);
 \path[fill]      (2,0) circle (2ex);
 \path[fill=yellow,draw=red] (3,0) circle (2ex);
 \path[clip]         (4,0) circle (2ex);
 \fill[color=orange] (3.5,-.5) rectangle (4.5,.5);
\end{tikzpicture}
```

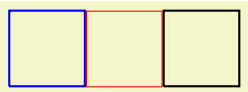
Die allgemeine Syntax mit `\path[⟨optArg⟩]` existiert z. B. für die oben genanten Pfad-Aktionen auch in Kurzform.

`\draw`	Steht innerhalb `{tikzpicture}` für `\path[draw]`.
`\fill`	Steht innerhalb `{tikzpicture}` für `\path[fill]`.
`\filldraw`	Steht innerhalb `{tikzpicture}` für `\path[fill,draw]`.
`\pattern`	Steht innerhalb `{tikzpicture}` für `\path[pattern]`.
`\shade`	Steht innerhalb `{tikzpicture}` für `\path[shade]`.
`\shadedraw`	Steht innerhalb `{tikzpicture}` für `\path[shade,draw]`.
`\clip`	Steht innerhalb `{tikzpicture}` für `\path[clip]`.
`\useasboundingbox`	Steht innerhalb `{tikzpicture}` für `\path[use as bounding box]`.

Beispiele für Inline-Grafiken

Um im Fließtext ein kleine Grafik oder auch nur eine Linie mittels Ti*k*Z zu erhalten, genügt ein kleiner Einzeiler. Beispiel für einen Kreis: `\tikz \draw[fill=orange] (0,0) circle (1ex);`  . Das untere Ende eines Elements wird immer auf der Grundlinie platziert. Der Parameter `baseline=⟨dimension⟩` legt fest, dass `⟨dimension⟩` auf der Grundlinie liegt. Da der Mittelpunkt des Kreises `(0,0)` ist, liegt dieser jetzt auf der Grundlinie `\tikz[baseline=0pt] \draw (0,0) circle (1ex);`.

Relative Koordinatenangaben.



```
\tikz \draw[thick,blue] (0,0) -- (1,0) -- (1,1) -- (0,1) -- (0,0);
\tikz \draw[red] (0,0) -- ++(1,0) -- ++(0,1) -- ++(-1,0) -- ++(0,-1);
\tikz \draw[thick]  (0,0) -- +(1,0) -- +(1,1) -- +(0,1) -- +(0,0);
```

Im ersten Beispiel (blau) zeichnet man einfach eine Linie durch die tatsächlichen Koordinaten. Im zweiten Fall (rot) erfolgen die Koordinatenangaben immer relativ zu `(0,0)` und die aktuelle Koordinate wird immer aktualisiert und als Ausgangspunkt für die darauffolgende Koordinate genommen. Zuletzt dagegen erfolgen alle Koordinatenangaben relativ zur Startkoordinate `(0,0)`.

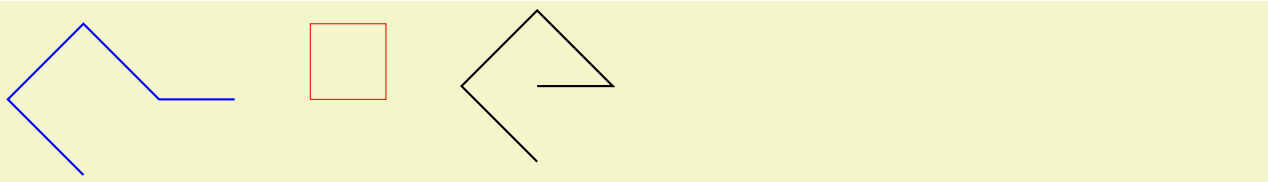
Ti*k*Z bietet einen einfachen Zugang zu Polarkoordinaten der Form `(⟨Winkel⟩:⟨Radius⟩)`, z. B. `(30:1cm)`. Das vorige Beispiel ließe sich auch wie folgt darstellen:

```
\tikz \draw[thick,blue] (2cm,0) -- (0:1cm) -- (90:1cm) -- (180:1cm) -- (270:1cm);
\tikz \draw[red] (3cm,0) -- ++(0:1cm) -- ++(90:1cm) -- ++(180:1cm) -- ++(270:1cm);
\tikz \draw[thick] (6cm,5) -- +(right:1cm) -- +(up:1cm) -- +(left:1cm) -- +(down:1cm);
```

Wie man sieht, haben wir jetzt verschiedene Effekte: In blau gehen wir nicht mehr den Pfad Punkt für Punkt ab, sondern beschreiben grob einen Kreis. Auch wenn der Ausgangspunkt nicht `(0,0)` ist, so beziehen sich doch die Polarkoordinaten darauf. Dasselbe Ergebnis zeigt die schwarze Figur, aber mit Koordinaten relativ zum Ausgangspunkt. Nur die rote Figur mit aktualisierten Koordinatenangaben erscheint in der gewünschten Form. Auch haben die verschiedenen Startpositionen mit dieser Syntax keine Wirkung und alle Objekte werden auf der Grundlinie platziert.

Die `\tikzpicture`-Umgebung

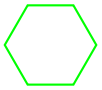
Für größere Zeichnungen oder um bestimmte Formatierungen anzuwenden, gibt es die Umgebung `\tikzpicture`. Aus dem vorigen Beispiel werden jetzt die Startpositionen des `\draw`-Befehls wirksam.



```
\begin{tikzpicture}
  \draw[thick,blue] (2cm,0) -- (0:1cm) -- (90:1cm) -- (180:1cm) -- (270:1cm);
  \draw[red] (3cm,0) -- ++(0:1cm) -- ++(90:1cm) -- ++(180:1cm) -- ++(270:1cm);
  \draw[thick] (6cm,5) -- +(right:1cm) -- +(up:1cm) -- +(left:1cm) -- +(down:1cm);
\end{tikzpicture}
```

Weitere Beispiele mit Syntax-Variationen

Als erstes zeichnen wir ein Sechseck. Die Farbangabe wirkt hier global. Wir lernen einen weiteren Zeichen-„Befehl“ kennen: `cycle`. Damit wird ein Pfad zum Ausgangspunkt geführt und somit *glatt* geschlossen.

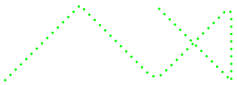


```
\begin{tikzpicture}[color=green,thick]
  \draw (60:4ex) -- (120:4ex) -- (180:4ex) -- (240:4ex) -- (300:4ex)
  -- (360:4ex) -- cycle;
\end{tikzpicture}
```

Das Beispiel soll mit abwechselnden Farben für jede Seite gezeichnet werden. Die Farbangabe muss lokal als optionales Argument von `\draw` für jede Seite gesetzt werden. Warum? Weil u. a. das `color`-Argument den Nebeneffekt hat, nicht so ganz lokal zu wirken, wie es andere Argumente machen (Beispiele weiter unten).

```
\begin{tikzpicture}[thick]     % globale Wahl der Linienst"arke
  \draw (0:4ex) -- (60:4ex);
  \draw [color=red] (4ex,4ex) (60:4ex) -- (120:4ex);
  \draw [color=blue] (4ex,4ex) (120:4ex) -- (180:4ex);
  \draw [color=green] (180:4ex) -- (240:4ex);
  \draw [color=yellow] (240:4ex) -- (300:4ex);
  \draw [color=magenta] (300:4ex) -- (360:4ex);
\end{tikzpicture}
```

Manche optionalen Elemente können auch im „laufenden“ Befehl geändert werden. Anders dagegen das Argument `dotted`, welches zwal lokal geschrieben, jedoch global wirkt.



```
\begin{tikzpicture}[color=green,style=thick]
  \draw (0,0) -- (1,1)
  [rounded corners,dotted] -- (2,0) -- (3,1)
  [sharp corners] -- (3,0) -- (2,1);
\end{tikzpicture}
```

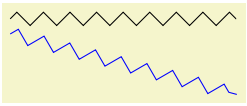
Lokale Anweisungen können in geschweifte Klammern gefasst werden, dann ist ihr Wirkungsbereich begrenzt. Im folgenden Beispiel wurde die Punktlinie geklammert, was zur Folge hat, dass `dotted` gänzlich ignoriert wird = unerwünscht!

```
\begin{tikzpicture}[color=green,style=thick]
  \draw (0,0) -- (1,1)
  {[rounded corners,dotted] -- (2,0) -- (3,1)}
  -- (3,0) -- (2,1);
\end{tikzpicture}
```

Eine saubere Trennung ist mit der Umgebung `scope` möglich (s. unten).

Weitere Spezialitäten (Auswahl)

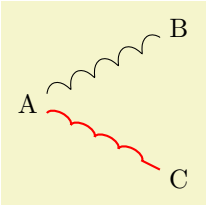
Schnecken (Snakes) via `\usetikzlibrary{snakes}`



```
\begin{tikzpicture}
  \draw[snake=zigzag] (0,0) -- (3,0);
  \draw[snake=zigzag,color=blue] (0,-0.2) -- (3,-1);
\end{tikzpicture}
```

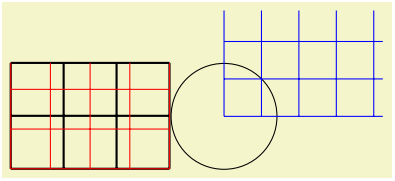
Knoten (nodes)

Wir nehmen gleich mal das vorige Beispiel und ändern die Schneckenform und benennen die Endpunkte. Damit haben wir einen Knoten erzeugt, der intern benannt werden und einen Namen bekommen kann (mindestens jedoch ein leeres Klammerpaar). Auf die internen Namen kann später auch Bezug genommen werden.




```
\begin{tikzpicture}
  \node (a) {A}  node (b) at (2,1) {B}  node (c) at (2,-1) {C};
  \draw[snake=coil] (a) -- (b);
  \draw[snake=bumps, segment length=20pt, red,thick] (a) -- (c);
\end{tikzpicture}
```

Gitter (grids)

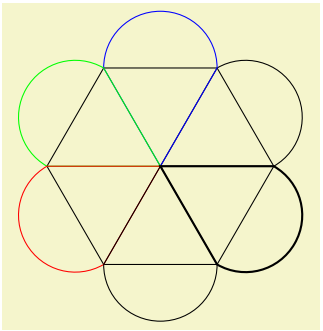


```
\begin{tikzpicture}[scale=0.7]
  \draw[thick] (0,0) grid [step=1]     (3,2);
  \draw[red]   (0,0) grid [step=.75cm] (3,2);
\end{tikzpicture}
\begin{tikzpicture}[scale=0.7]
  \draw       (0,0) circle (1);
  \draw[blue] (0,0) grid [step=(45:1)] (3,2);
\end{tikzpicture}
```

Die bounding box des zweiten Bildes wurde automatisch nach links erweitert.

Winkel und Kreisbögen

Mit dem Befehl `arc(⟨Startwinkel⟩:⟨Zielwinkel⟩:Radius)` bietet Ti*k*Z einen einfachen Zugang zu Polarkoordinaten. Die Koordinaten im folgenden Beispiel sind z. T. in der Form `(30:1cm)` angegeben, wobei die `30` die Winkelangabe und `1cm` der Radius ist.



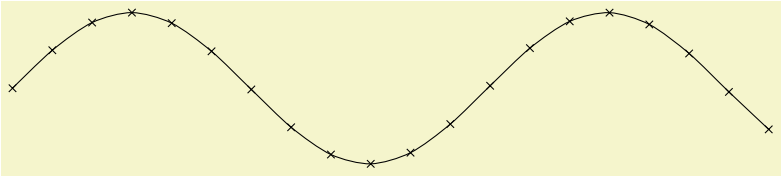
```
\begin{tikzpicture}[scale=1.5]
  \draw (60:1cm) -- (120:1cm) -- (180:1cm) --
  (240:1cm) -- (300:1cm) -- (360:1cm) -- cycle;
  \draw (0,0) -- (0:1cm) arc (-60:120:0.5cm) -- cycle;
  \begin{scope}[rotate=60, color=blue]
    \draw (0,0) -- (0:1cm) arc (-60:120:0.5cm) -- cycle;
  \end{scope}
  \begin{scope}[rotate=120, color=green]
    \draw (0,0) -- (0:1cm) arc (-60:120:0.5cm) -- cycle;
  \end{scope}
  \begin{scope}[rotate=180, color=red]
    \draw (0,0) -- (0:1cm) arc (-60:120:0.5cm) -- cycle;
  \end{scope}
  \begin{scope}[rotate=240]
    \draw (0,0) -- (0:1cm) arc (-60:120:0.5cm) -- cycle;
  \end{scope}
  \begin{scope}[rotate=300, black, thick]
    \draw (0,0) -- (0:1cm) arc (-60:120:0.5cm) -- cycle;
  \end{scope}
\end{tikzpicture}
```

In dem Beispiel wurde die Umgebung `scope` genutzt. Damit lassen sich Blöcke von Anweisungen festlegen, auf die bestimmte Operationen/Formatierungen angewandt werden können.

Funktionen zeichnen

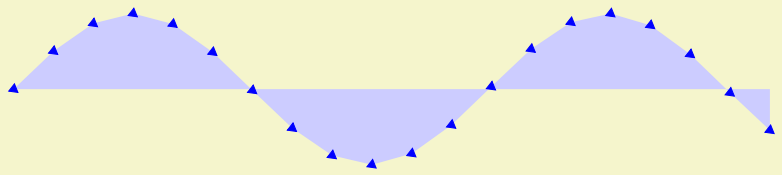
Kann man machen. Diese Einschränkung gilt, da Ti*k*Z eben *kein* Zeichenprogramm ist und PSTricks vieles besser kann. Dennoch kommt man auch mit Ti*k*Z zum Ziel. Am Ende des Vortrags habe ich noch ein paar Beispiele mit Ti*k*Z bzw. PSTricks gegenübergestellt.

In Ti*k*Z gibt man die Koordinaten entweder direkt im Zeichenbefehl an oder lädt sie aus einer externen Datei, wie hier:



```
\tikz \draw plot[mark=x,smooth] file {pgfmanual-sine.table};
```

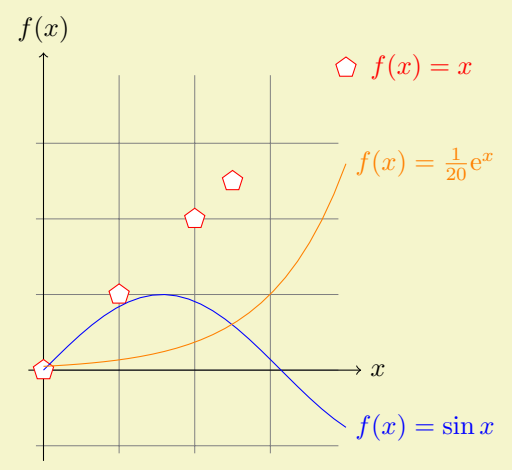
Als nächstes auch mal gefüllt.



```
\tikz \fill[fill=blue!20]
  plot[mark=triangle*,mark options={color=blue,rotate=112}]
    file{pgfmanual-sine.table} |- (0,0);
```

Es ist weiterhin möglich, Funktionen mit Hilfe von GNUPLOT zur Laufzeit erstellen zu lassen und einzubinden (nicht demonstriert).

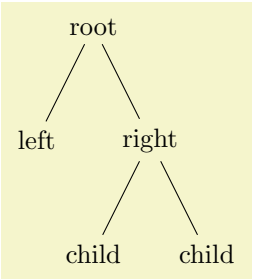
Natürlich kann man die Funktion fast direkt als Formel angeben, nämlich als Koordinatengleichung. Im folgenden Beispiel ist der darzustellende Bereich kontinuierlich angegeben.




```
\begin{tikzpicture}[domain=0:4]    % Zeichenbereich
  % Gitter zeichnen
  \draw[very thin,color=gray] (-0.1,-1.1) grid (3.9,3.9);
  % Achsen zeichnen
  \draw[->] (-0.2,0) -- (4.2,0) node[right] {$x$};
  \draw[->] (0,-1.2) -- (0,4.2) node[above] {$f(x)$};
  % Funktionen zeichnen
  \draw[color=red,samples at={0,1,2,2.5,4},mark=pentagon*,mark options={fill=white,draw=red},
    mark size=4pt] plot[only marks] (\x,\x) node[right] at (4.2,4) {$f(x) =x$};
  \draw[color=blue] plot (\x,{sin(\x r)}) node[right] {$f(x) = \sin x$};
  \draw[color=orange] plot (\x,{0.05*exp(\x)}) node[right] {$f(x) = \frac{1}{20} \mathrm e^x$};
\end{tikzpicture}
```

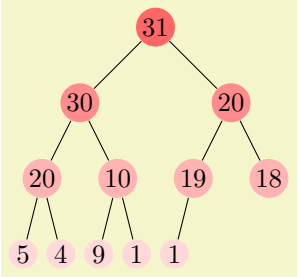
Verschiedene Diagrammtypen, Bäume

Bauen wir uns eine kleine Hierarchie:



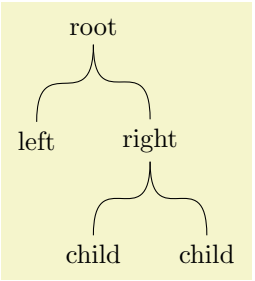

```
\begin{tikzpicture}
  \node {root} %[grow oder grow'=up/left/north/east/north east/...]
    child {node {left}}
    child {node {right}
      child{node{child}}
      child{node{child}}
    };
\end{tikzpicture}
```

Wir mögen aber Farbe und so kommt hier ein umfangreicheres Beispiel mit bisher noch nicht erläuterten Optionen:




```
\begin{tikzpicture}[level distance=10mm]
  \tikzstyle{every node}=[fill=red!60,circle,inner sep=1pt]
  \tikzstyle{level 1}=[sibling distance=20mm,
    set style={{every node}+=[fill=red!45]}]
  \tikzstyle{level 2}=[sibling distance=10mm,
    set style={{every node}+=[fill=red!30]}]
  \tikzstyle{level 3}=[sibling distance=5mm,
    set style={{every node}+=[fill=red!15]}]
  \node {31}
     child {node {30}
       child {node {20}
         child {node {5}}
         child {node {4}}
       }
       child {node {10}
         child {node {9}}
         child {node {1}}
       }
     }
     child {node {20}
       child {node {19}
         child {node {1}}
         child[fill=none] {edge from parent[draw=none]}
       }
       child {node {18}}
     };
\end{tikzpicture}
```

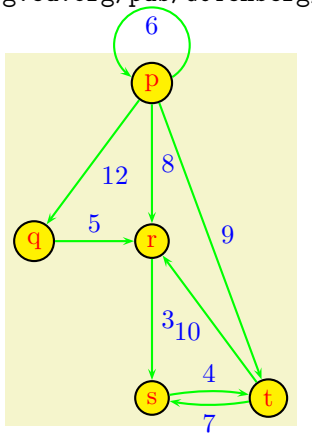
Der Parameter `silbling distance` bestimmt den Abstand der Child nodes zueinander. Diese gedachte Linie verläuft senkrecht zur Wachstumsrichtung des Baumes. Die Wachstumsrichtung wird über `grow` angegeben, `grow'` kehrt sie um. Den Abstand zwischen den Ebenen bestimmt der Parameter `level distance`. Im vorigen Beispiel wird mittels `\tikzstyle` für jede Ebene des Baumes der Abstand der Child nodes sowie deren Farben festgelegt. Die Option `edge from parent` bestimmt die Verbindung einer Child Node mit ihrer Parent Node (Definition global oder lokal möglich, siehe nächstes Beispiel).




```
\begin{tikzpicture}[edge from parent path=
  {(\tikzparentnode.south) .. controls +(0,-1) and +(0,1)
                           .. (\tikzchildnode.north)}]
  \node {root}
    child {node {left}}
    child {node {right}
      child {node {child}}
      child {node {child}}
    };
\end{tikzpicture}
```

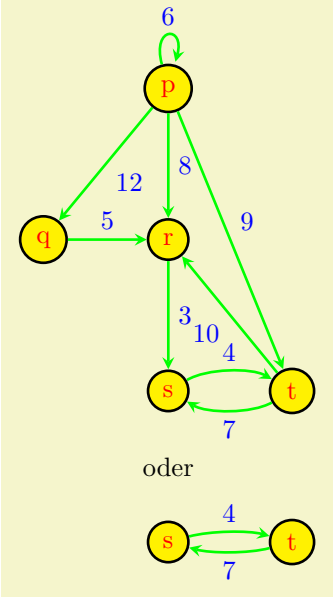
Vergleich PSTricks und TikZ

Das folgende Beispiel, erstellt mit PSTricks, entstammt einer Präsentation von Denis Girou `http://www.gutenberg.eu.org/pub/GUTenberg/publicationsPDF/16-girou.pdf`.




```
\begin{psmatrix}[fillstyle=solid,
 fillcolor=yellow,mnode=circle,colsep=1]
 & \red p \\
 \red q & \red r \\
 & \red s & \red t
\end{psmatrix}
\psset{linecolor=green,arrows=->,labelsep=1mm,shortput=nab}
\blue
\nccircle{1,2}{0.5cm}^{6}
\ncline{1,2}{2,1}^{12}
\ncline{1,2}{2,2}^{8}
\ncline{1,2}{3,3}^{9}
\ncline{2,1}{2,2}^{5}
\ncline{2,2}{3,2}^{3}
\ncline{3,3}{2,2}^{10}
\ncarc[arcangle=10]{3,3}{3,2}^{7}
\ncarc[arcangle=10]{3,2}{3,3}^{4}
```

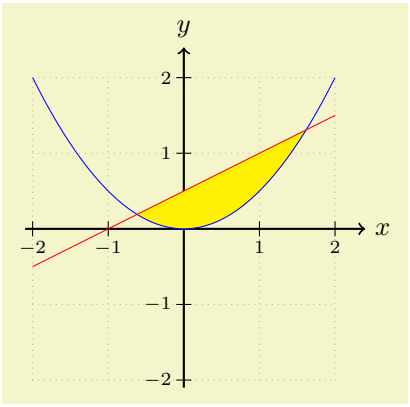
Das gleiche Beispiel, jetzt mit Ti*k*Z gezeichnet. Das Grundgrüst stammt aus dem Handbuch und ist so verändert, dass es dem PSTricks-Beispiel so nah wie möglich kommt.




```
\begin{tikzpicture}[%
 >=stealth,                 % Aussehen der Pfeilspitzen
 ->,                        % Pfeile als Verbindungslinien
 looseness=.7,              % Kr"ummung der Pfeile mit Option 'bent'
 auto,                      % Position des Ankers f"ur Node Labels
 color=green,               % Farbe aller Linien
 text=red,                  % Textfarbe in den Matrix-Nodes
 line width=1pt             % Linienst"arke f"ur alle Elemente
]
\matrix [%
 matrix of nodes,           % Elemente der Matrix sind Nodes
 column sep={1cm},          % Abstand der Spalten
 row sep={2cm,between origins}, % Zeilenabstand, vom Nodemittelpunkt aus
 nodes={circle,             % Stil der Nodes: Kreis um die Matrix-Nodes
  fill=yellow,              % F"ullfarbe
  draw,                     % Nodes zeichnen -> ja
  anchor=center},           % Zentrierung der Nodes
 ampersand replacement=\&] % wg. PSTricks '&' umdefinieren
 {                            % Matrixelemente angeben
         \& |(p)| p  \&          \\
  |(q)| q \& |(r)| r  \&          \\
         \& |(s)| s  \& |(t)|  t \\
         \& |(s2)| s \& |(t2)| t \\
 };
 \tikzstyle{every node}=[color=blue]   % Stil der Node-Beschriftung der
                                       % Verbindungslinien
 \draw (p) --              (q) node [midway] {12};
 \draw (p) --              (r) node [midway] {8};
 \draw (p) --              (t) node [midway] {9};
 \draw (q) --              (r) node [midway] {5};
 \draw (r) --              (s) node [midway] {3};
 \draw (t) --              (r) node [midway] {10};
 \draw (s) to [bend left]  (t) node [midway] {4};  % gebogene Linie
 \draw (t) to [bend left]  (s) node [midway] {7};  % gebogene Linie
 \draw (p) to [loop above] ()  node [midway] {6};  % Schleife "uber 'p'
 % explizite Angabe des Aus- und Eintrittswinkels erm"oglicht bessere
 % Darstellung der gebogenen Pfeile als "uber loseness
 \path (s) -- (s2) node [below of=s,color=black] {oder};
 \draw (s2) to[out=15,in=165]  (t2) node [midway] {4};
 \draw (t2) to[out=195,in=345] (s2) node [midway] {7};
\end{tikzpicture}
```

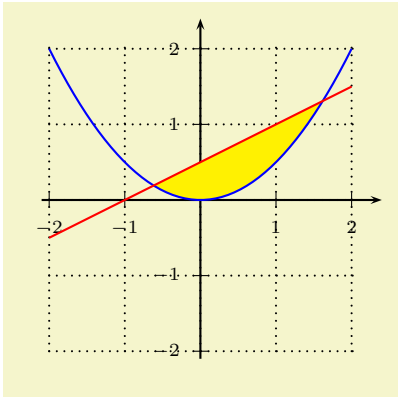
Nachfolgend ein Beispiel mit mathematischen Funktionen und einer Füllung ihrer Schnittmenge. Vorgabe ist dieses Mal die Zeichnung mit Ti*k*Z. Nicht so elegant ist die händische Ermittlung der Schnittpunkte beider Funktionen.

```
%%%% Beginn Globale Einstellungen f"ur TikZ %%%%
% Zeichenbereich f"ur Funktionen
\tikzstyle{every picture}=[domain=-2:2]
% Gitterstil. Zum Stil 'help lines' wird Linienart 'dotted' hinzugef"ugt
\tikzstyle{help lines}+=[dotted]
%%%% Beginn Globale Einstellungen f"ur PSTricks %%%%
\renewcommand{\pshlabel}[1]{$\scriptstyle#1$}
\renewcommand{\psvlabel}[1]{$\scriptstyle#1$}
```



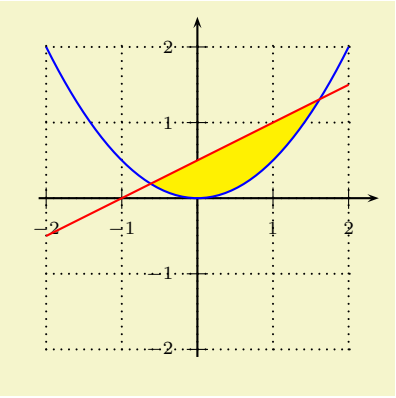

```
\begin{tikzpicture}[scale=1]
 % Gitter zeichnen
 \draw[style=help lines,step=1cm] (-2,-2) grid (2,2);
 % Achsen zeichnen
 \draw[->,thick] (-2.1,0) -- (2.4,0) node[right] {$x$};
 \draw[->,thick] (0,-2.1) -- (0,2.4) node[above] {$y$};

 % F"ullbereich zeichnen. Schnittpunkte selbst berechnet.
 % Ohne domain-Argument im plot-Befehl wird Kurve ab
 % linkem Rand gef"ullt
 \path[fill=yellow] ++(-0.618,0.191) --
  plot[smooth,domain=-0.618:1.618]
  (\x,{0.5*\x*\x}) --(1.618,1.309) -- (-0.618,0.191);

 % Achsen beschriften, erst hier, da fill "uberschreiben
 \foreach \x in {-2,-1,1,2}
  \draw (\x,-.1) -- (\x,.1) node[below=4pt] {$\scriptstyle\x$};
 \foreach \y in {-2,-1,1,2}
  \draw (-.1,\y) -- (.1,\y) node[left=4pt]  {$\scriptstyle\y$};

 % Funktionen zeichnen
 \draw[blue] plot[smooth] (\x,{0.5*\x*\x});
 \draw[red]  plot         (\x,{0.5*\x}+0.5);
\end{tikzpicture}
```

PSTricks ist dran. Hier ebenfalls Zeichnen der Füllfläche unter expliziter Angabe der Schnittpunkte beider Funktionen.



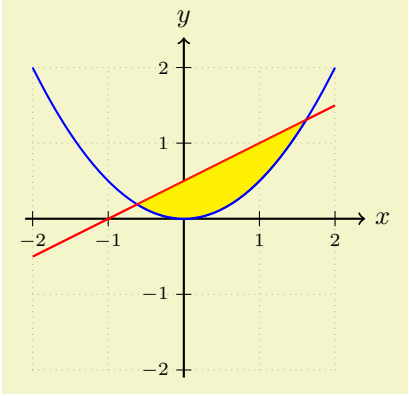

```
\begin{pspicture}(-2.5,-2.5)(2.5,2.5)
 \psgrid[subgridcolor=gray,subgriddiv=1,griddots=10,gridlabels=0]%
   (-2,-2)(2,2)
 \psaxes[%
   Dx=1%
   ,Dy=1%
   ,ticksize=-.1 .1%
   ,subticks=0%
   ]%
  {->}(0,0)(-2.1,-2.1)(2.4,2.4)
 \pscustom{%
  \gsave
  \psplot{-0.618}{1.618}{x 2 exp 0.5 mul}
  \psline(-0.618,0.191)
  \fill[fillstyle=solid,fillcolor=yellow]
  \grestore
 }
 \psset{linecolor=Blue}
 \psplot{-2}{2}{x 2 exp 0.5 mul}
 \psset{linecolor=Red}
 \psplot{-2}{2}{x 0.5 mul 0.5 add}
 \end{pspicture}
```

PSTricks zum Zweiten. Elegantes Füllen der Schnittmenge über `psclip`.



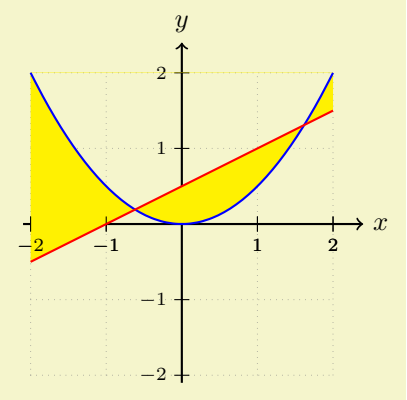

```
\begin{pspicture}(-2.5,-2.5)(2.5,2.5)
\psgrid[subgridcolor=gray,subgriddiv=1,griddots=10,gridlabels=0]%
  (-2,-2)(2,2)
\psaxes[Dx=1,Dy=1]{->}(0,0)(-2.1,-2.1)(2.4,2.4)
\psclip{%
  \pscustom[linestyle=none]{%
    \psplot{-2}{2}{x 2 exp 0.5 mul}
    \lineto(2,2)
  }
  \pscustom[linestyle=none]{%
    \psplot{-2}{2}{x 0.5 mul 0.5 add}
    \lineto(2,-2)
    \lineto(-2,-2)
  }
}
\psframe*[linecolor=yellow](-2,-2)(2,2)
\endpsclip

\psset{linecolor=Blue}
\psplot{-2}{2}{x 2 exp 0.5 mul}
\psset{linecolor=Red}
\psplot{-2}{2}{x 0.5 mul 0.5 add}
\end{pspicture}
```

Nochmaliges Zeichnen mittels Ti*k*Z, hier mit `clip`-Option. Nach Anwenden von `clip` wird leider der Zeichenbereich für alle nachfolgenden Elemente beschnitten. Daher werden die beiden Funktionen ein zweites Mal gezeichnet.




```
\begin{tikzpicture}[%
  scale=1.00          % Skalierung
]
% Gitter zeichnen
\draw[style=help lines,step=1cm] (-2,-2) grid (2,2);
% Achsen zeichnen, zuerst hier, da clip die Zeichenfl"ache beschneidet
\draw[->,thick] (-2.1,0) -- (2.4,0) node[right] {$x$};
\draw[->,thick] (0,-2.1) -- (0,2.4) node[above] {$y$};
% Achsen beschriften
\foreach \x in {-2,-1,1,2}
  \draw (\x,-.1) -- (\x,.1) node[below=5pt] {$\scriptstyle\x$};
\foreach \y in {-2,-1,1,2}
  \draw (-.1,\y) -- (.1,\y) node[left=5pt]  {$\scriptstyle\y$};
% Funktionen zeichnen
\draw[blue,thick] plot[smooth] (\x,{0.5*\x*\x});
\draw[red,thick]  plot[smooth] (\x,{0.5*\x}+0.5);
% F"ullbereich mittels clip zeichnen.
\path[clip] plot (\x,{0.5*\x}+0.5) -- (2,-2);
\filldraw[color=blue,fill=yellow] plot[smooth] (\x,{0.5*\x*\x});
% Funktionen nochmal zeichnen, da clip
\draw[blue,thick] plot[smooth] (\x,{0.5*\x*\x});
\draw[red,thick]  plot[smooth] (\x,{0.5*\x}+0.5);
\end{tikzpicture}
```

Noch ein Versuch mittels Ti*k*Z mit `filldraw`-Option:




```
\begin{tikzpicture}[%
   ,scale=1.00          % Skalierung
   ,fill=yellow         % Farbe der F"ullung
  ]
  % Gitter zeichnen
  \draw[style=help lines,step=1cm] (-2,-2) grid (2,2);
  % Achsen zeichnen, zuerst hier, da clip die Zeichenfl"ache beschneidet
  \draw[->,thick] (-2.1,0) -- (2.4,0) node[right] {$x$};
  \draw[->,thick] (0,-2.1) -- (0,2.4) node[above] {$y$};
  % Achsen beschriften
  \foreach \x in {-2,-1,1,2}
    \draw (\x,-.1) -- (\x,.1) node[below=5pt] {$\scriptstyle\x$};
  \foreach \y in {-2,-1,1,2}
    \draw (-.1,\y) -- (.1,\y) node[left=5pt]  {$\scriptstyle\y$};
  % F"ullfl"ache zeichnen, dabei beide Funktionen so zeichnen,
  % dass even odd rule zum Tragen kommt
  \fill[even odd rule] plot (\x,{0.5*\x}+0.5) -- (2,2) --
    plot[smooth] (\x,{0.5*\x*\x}) -- (-2,2) -- (-2,-0.5);
  % x-Achse nochmal beschriften
  \foreach \x in {-2,-1,1,2}
    \draw (\x,-.1) -- (\x,.1) node[below=5pt] {$\scriptstyle\x$};
  % Funktionen nochmal zeichnen, da clip
  \draw[blue,thick] plot[smooth] (\x,{0.5*\x*\x});
  \draw[red,thick]  plot[smooth] (\x,{0.5*\x}+0.5);
\end{tikzpicture}
```

Fazit

TikZ

- Portabel (PDF/PostScript)
- Zusatzfunktionen über Bibliotheken
- Zusammenhängende Dokumentation
- Einfache Installation
- Wenige, allgemeine Grundbefehle
- Steuerung über optionale Argumente

PSTricks

- Mehr spezialisiertere Befehle als Ti*k*Z
- Größerer Funktionsumfang, da PostScript als Programmiersprache
- Zusatzfunktionen über Zusatzpakete
- keine umfassende Dokumentation (außer PSTricks-Buch)
- Zusatzpakete verstreut im TDS-Baum

TEX-Resourcen für ein reines Ti*k*Z-Dokument:

```
Here is how much of TeX's memory you used:
 13134 strings out of 94982
 217223 string characters out of 1180600
 280362 words of memory out of 1500000
 16034 multiletter control sequences out of 10000+50000
 11143 words of font info for 40 fonts, out of 1200000 for 2000
 62 hyphenation exceptions out of 8191
 49i,14n,59p,829b,1329s stack positions out of
 5000i,500n,6000p,200000b,5000s
PDF statistics:
 123 PDF objects out of 1000 (max. 8388607)
 25 named destinations out of 1000 (max. 131072)
 17 words of extra memory for PDF output out of 10000 (max. 10000000)
```

TEX-Resourcen für ein Dokument mit Ti*k*Z und PSTricks:

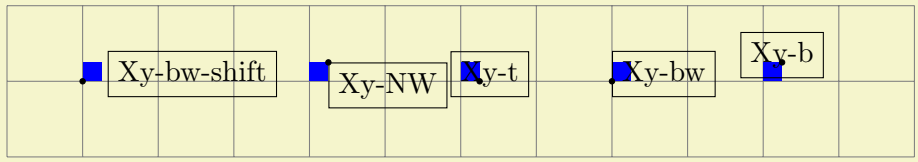
```
Here is how much of TeX's memory you used:
 11462 strings out of 94983
 186911 string characters out of 1180623
 327132 words of memory out of 1500000
 14443 multiletter control sequences out of 10000+50000
 6591 words of font info for 25 fonts, out of 1200000 for 2000
 62 hyphenation exceptions out of 8191
 58i,13n,55p,1435b,1107s stack positions out of
 5000i,500n,6000p,200000b,5000s
```

Für den Vergleich Ti*k*Z/PSTricks wurden beide Pakete gemeinsam geladen (Nebenwirkungen?).

Erstellung des Dokuments: Für den Ti*k*Z-Teil wurde direkt pdfLATEX eingesetzt, der Teil mit PSTricks wurde mittels `latex`, `dvips`, `ps2pdf` erstellt. Beide Teildokumente wurden mit dem Programm `pdftk` zusammengefügt.

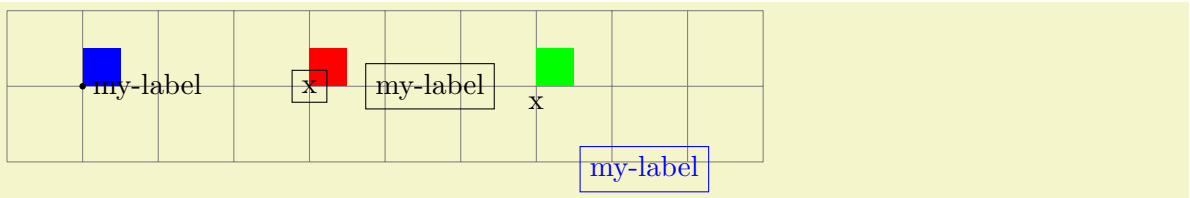
label-Spielereien

Anfrage in dctt zur Positionierung von Beschriftungen für nodes. Der schwarze Kreis kennzeichnet den Bezugspunkt der Beschriftung.




```
\begin{tikzpicture}
  \tikzstyle{every node}=[draw] % Rahmen um die label
  \draw[gray] (0,-1) grid (12,1);
  \fill[blue] (1,0) node[black,anchor=base west,xshift=2ex]%
     {Xy-bw-shift} rectangle (1.25,.25);
  \filldraw (1,0) circle (1pt);
  \fill[blue] (4,0) rectangle (4.25,.25) node[black,anchor=north west] {Xy-NW};%
  \filldraw (4.25,.25) circle (1pt);
  \fill[blue] (6,0) node[black,anchor=text] {Xy-t} rectangle (6.25,.25);%
  \filldraw (6.25,0) circle (1pt);
  \fill[blue] (8,0) node[black,anchor=base west] {Xy-bw} rectangle (8.25,.25);%
  \filldraw (8,0) circle (1pt);
  \fill[blue] (10,0) rectangle (10.25,.25) node[black,anchor=base] {Xy-b};%
  \filldraw (10.25,.25) circle (1pt);
\end{tikzpicture}
```

Problem: Label ragen in das blaue Rechteck hinein.
Lösung: label mit Angabe einer `label distance`:




```
\begin{tikzpicture}[label distance=5mm]
  %\tikzstyle{every node}=[draw]
  \tikzstyle{every label}=[draw] % Rahmen um das label

  \draw[gray] (0,-1) grid[gray] (10,1); 2x gray sonst ist Gitter schwarz!?
  % Standard-node-Beschriftung
  \fill[blue] (1,0) node[black,anchor=west] {my-label} rectangle (1.5,.5);%
  \filldraw (1,0) circle (1pt);
  % Erzeugen eines labels, rechts vom Bezugspunkt x
  \fill[red]  (4,0) node[draw,black,label={[black]right:my-label}] {x}
    rectangle (4.5,0.5);%
  % labels rechts unten und Nodebeschriftung darunter
  \fill[green]  (7,0) node[black,anchor=north,label={[blue]below right:my-label}] {x}
    rectangle (7.5,0.5);%
\end{tikzpicture}
```