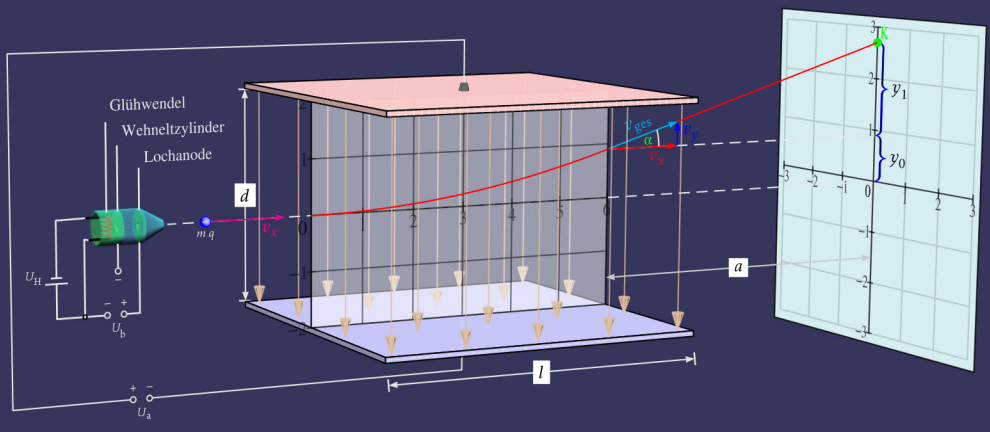
Herbert Voß

PSTricks

Grafik mit PostScript für TEX und LATEX

7. Auflage



lehmanns media

dante

PSTricks

Siebte, überarbeitete und erweiterte Auflage

Herbert Voß
Berlin

dante

lehmanns media

© 2016 Herbert Voß, Berlin
Siebte, überarbeitete und erweiterte Auflage
ISBN 978-3-86541-858-6
Umschlag: Herbert Voß
Satz: LaTeX (Libertine, Kp-Math und Beramono)
Verlag: Lehmanns Media, Berlin (www.lehmanns.de)
Druck: Dimograf – Bielsko-Biała – Polen

Inhaltsverzeichnis

Vorwort

»PSTricks–mehr als nur ein alter Hut«, war ein Vortrag auf der dante-Tagung in Darmstadt betitelt. [54] Er sollte den Teilnehmern vor Augen führen, dass PSTricks, als eines der ersten für Plain TEX entwickelten Pakete nichts an seiner Aktualität und vor allem seiner Professionalität verloren hat. Die Qualität der Grafikausgabe, die mit PSTricks erreicht werden kann, sucht sicherlich ihresgleichen. Dabei darf nicht vergessen werden, dass alles seine Grenzen hat, so auch PSTricks mit seinen vielfältigen Paketen, denn die Grafiken müssen komplett in TEX- und somit in Textform eingegeben werden. Keiner wird auf die Idee kommen, die Baupläne für eine Gasturbinenanlage oder das Layout für die nächste Prozessorgeneration in Textform zu erstellen. Dies ist auch oft gar nicht erforderlich für die Forschung und Lehre, denn hier gilt es sehr oft Grafiken zu erstellen, die in ihrem Anspruch zwar reduziert aber dennoch komplex genug sind, wie es beispielsweise für zu erstellende Arbeitsblätter, Veröffentlichungen (Aufsätze, Bücher), Studien-, Diplom- und Doktorarbeiten, usw. sehr oft der Fall ist. Hier bietet PSTricks fast unvorstellbare Möglichkeiten hinsichtlich der Professionalität in der Ausgabe, denn hinter PSTricks steht die zwar alte, aber im grafischen Bereich mächtige Programmiersprache PS. PSTricks verfügt mittlerweile über derartig viele verschiedene Pakete und somit Makros und Parameter, dass schon lange keiner mehr in der Lage ist, diese alle präsent zu haben. Hier soll diese Veröffentlichung helfen, denn die aktuelle Dokumentation von PSTricks ist nicht nur in die Jahre gekommen, sondern auch bislang unvollständig gewesen.

TEX lebt vom Enthusiasmus der Entwickler und der postiven Rückkopplung der Benutzer. So war es Timothy Van Zandt, der Anfang der 90-er Jahre ein paar Makros zur Unterstützung der Seminar-Klasse schrieb. Und wie so oft, es fing »ganz langsam an, aber dann…« Irgendwann korrelierten Enthusiasmus und »offizieller Broterwerb« im negativen Sinne, sodass Timothy die mehr oder weniger fertige Arbeit am sehr umfangreichen Grundgerüst von PSTricks einstellte. Etwas zu entwickeln ist eine Seite, aber es am Laufen zu halten eine nicht minder schwierige und zeitaufwendige Sache. Seit Jahren kümmerte sich Denis Girou darum, dass Bugs beseitigt und Fragen auf der PSTricks-eigenen Mailingliste umfassend und schnell beantwortet wurden. Nebenbei

entwickelte er noch eine Reihe weiterer Pakete. Ohne Denis wäre PSTricks heute nicht das was es ist.

Fast von Anfang an dabei war auch Rolf Niepraschk. Ohne ihn wäre diese Veröffentlichung erstens nicht so schnell entstanden und zweitens auch nicht mit so wenig Fehlern. Er übernahm neben Uwe Ziegenhagen und Hubert Gäßlein den wichtigen Part des Korrekturlesens und engagierte sich vehement, LaTeX, PSTricks und Farbe unter einen Hut zu bringen. Rolf hatte auch immer irgendwo die unvollständigen Uralt-Dokumentationen gespeichert und konnte sie auch unter LaTeX 2_ε zum Laufen bringen, was wahrlich nicht ganz einfach war. Immer, wenn es mit TeX an das Eingemachte ging, war Hubert gefragt, der mit seinen TeX-Kenntnissen alles auf den Punkt bringen konnte. Jens-Uwe Morawski erstellte das Cover und zeigte wieder einmal, was mit ConTeXt alles möglich ist. Uwe Siart stellte seine tabellarische Zusammenfassung der Makros zur Verfügung und der Berliner TeX-Stammtisch, sowie viele PSTricks-Anwender überprüften die bei der Arbeit an diesem Buch entdeckten Bugs und Ungereimtheiten von PSTricks.

Last but not least, was wäre der Mensch ohne seinen Verein. DANTE e.V. unterstützte die Veröffentlichung nicht nur dieses Buches, sondern war mit Klaus Hoeppner letztlich der Impulsgeber. Klaus war auch zusammen mit Christoph Kaeder von Lehmanns Fachbuchhandlung stets bemüht eine adäquate Veröffentlichungsform zu finden.

Von den Entwicklern der PSTricks-Pakete seien noch ausdrücklich Manuel Luque und Christophe Jorssen erwähnt. Ohne Manuel wären die 3D-Welten in dieser professionellen Weise nicht in ein PSTricks-Paket eingeflossen. Allen gilt es zu danken dafür, dass PSTricks das ist, was es ist: ein professionelles Werkzeug. Bleibt nur die Hoffnung, dass dieses Buch ebenfalls diesen Anspruch erfüllen möge.

Berlin, im Juni 2004 Herbert Voß

Vorwort zur 7. Auflage

Diese siebte Auflage ist mehr als nur ein verbesserter Nachdruck der sechsten Auflage; sie ist zum erstem Mal auch komplett in Farbe. Die Pakete pst-solarsystem, pst-intersect, pst-ode und pst-cie wurden neu aufgenommen. Inhaltlich gibt es zudem mehrere Verschiebungen, da zwischenzeitlich einige Makros in andere Pakete verschoben wurden.

Dank geht an Uwe Siart, der wieder seine hilfreiche Zusammenstellung der Befehle zur Verfügung stellte und an Thomas Söll für die Titelgrafik. Christine Römer und Sebastian Hitziger haben mich auf Fehler in der sechsten Auflage hingewiesen. Alle Beispiele findet man wieder als eigenständige Programme auf CTAN:/info/examples/PSTricks_7_de.

Berlin, im Mai 2016 Herbert Voß

Kapitel 1

Einführung

Mit der Entwicklung von TEX wurde nicht unbedingt Wert auf die grafischen Fähigkeiten des Systems gelegt. TEX selbst verfügt daher auch nur über rudimentäre grafische Elemente, die mit der Einführung von LATEX erweitert wurden und erst seit neuestem durch das Paket `pict2e` [18] den normalen Ansprüchen Rechnung tragen. So entstand schon sehr früh die Idee, die grafischen Fähigkeiten der »alten« Programmiersprache PostScript zu nutzen, denn PostScript war ohnehin als Standard-Ausgabeformat für TEX vorgesehen, wenn man einmal vom »Zwischenformat« DVI absieht.

1.1 Die Geschichte

`PSTricks` gehört zu den älteren Paketen, die bereits für eine Anwendung unter Plain TEX geschaffen wurden.

> I started in 1991. Initially I was just trying to develop tools for my own use. Then I thought it would be nice to package them so that others could use them. It soon became tempting to add lots of features, not just the ones I needed. When this become so interesting that it interfered with my »day job«, I gave up the project »cold turkey«, in 1994.
>
> [Timothy Van Zandt]

Dieser Ablauf kann fast als der Standard für viele Paketentwickler angesehen werden, die fast ausnahmslos ehrenamtlich tätig waren und sind. Es fängt sehr häufig ganz bescheiden an und wird irgendwann zu einem Selbstläufer, sobald man sich mit seinen

Ideen an die TEX-Öffentlichkeit wendet. Andererseits ist die Weiterentwicklung von TEX von genau solchen selbstlosen Entwicklern abhängig, wenn sie weiter auf einem hohen softwaretechnischen Niveau bleiben will.

Nach Timothy Van Zandt übernahmen Sebastian Rahtz und Denis Girou die Aufgabe, `PSTricks` zu betreuen und Fehler zu beheben. Die Zahl der mittlerweile neu hinzugekommenen Pakete (siehe Kapitel 32) nimmt stetig zu, sodass `PSTricks` nach wie vor nicht als »alter Hut« bezeichnet werden kann.[54]

1.2 Der Kern

Der Kern von `PSTricks` befindet sich ähnlich wie TEX und LATEX in einem quasi eingefrorenen Zustand, woran sich auch in nächster Zeit nichts ändern wird. Zum Kern sind die in Tabelle 1.1 angegebenen Pakete zu zählen, die sich alle in dem CTAN-Verzeichnis CTAN: `/graphics/pstricks/generic/` befinden[1]. Der Verzeichnisname `generic` deutet schon daraufhin, dass es sich hier um Plain TEX-kompatible Pakete handelt, die entweder über den `\input`-Befehl oder für LATEX über den korrespondierenden `\usepackage`-Befehl eingebunden werden können. Die entsprechenden Style-Dateien befinden sich im Verzeichnis `CTAN:/graphics/pstricks/latex/`.

Die genannten Basispakete gehören zu jeder bekannten TEX-Distribution, sodass hier keine weiteren Schritte notwendig sind, um mit ihnen arbeiten zu können. Weitere Informationen zu diesen und anderen Paketen findet man im Kapitel 32. Bei den Dateien mit der Endung `.con` handelt es sich um spezielle Konfigurationsdateien, von denen nur `pstricks.con` entscheidend ist. Je nach verwendetem System ist eine der anderen Dateien in `pstricks.con` umzubenennen. Die meisten Distributionen kommen bereits mit einer korrekten Grundeinstellung, beispielsweise MiKTEX oder TEX Live.

Tabelle 1.1: Die Basispakete von `PSTricks`

Paketname	*Datum*	*Eigenschaft*
`Changes`	`2016`	log
`pst-fp.tex`	`2010`	Dezimalzahloperationen
`pst-key.tex`	`1998`	Key-Value-Interface
`pstricks.con`	`2010`	Konfigurationsdatei (DVIPS-Version)
`pstricks.tex`	`2010`	Das Basispaket
`pstricks97.tex`	`1999`	Version 1997 patch 14
	Konfigurationsdateien	
`distiller.cfg`	`2008`	
`dvips.cfg`	`2010`	Standard, siehe
`dvipsone.cfg`	`1994`	
`textures.cfg`	`1997`	
`vtex.cfg`	`2003`	
`xdvipdfmx.cfg`	`2015`	for XETEX

[1] CTAN: Comprehensive TEX Archive Network

1.3 Was es ist ...

PSTricks ist eine Ansammlung von PS-basierten TEX-Makros, die mit den meisten TEX-Formaten wie Plain TEX, LATEX, LuaTEX und ConTEXt kompatibel sind. PSTricks gibt ihnen die Möglichkeit Farbe, Grafiken, Transformationen, Bäume, Overlays usw. einzusetzen. Wichtige, grundlegende Informationen enthält die Datei README, die Teil des Pakets PSTricks ist. Sie enthält auch wichtige Informationen über aktuelle Probleme im Umgang mit PSTricks.

Die Zahl der Optionen und ihr Einsatz mit PSTricks-Makros ist mittlerweile wegen ihrer großen Zahl nur noch wenigen Anwendern vollständig geläufig, sodass diese Veröffentlichung eine Hilfe sein kann, den Überblick zu bekommen oder zu behalten. Der Index enthält daher auch jedes besprochene Makro samt seiner Optionen und kann neben dem Inhaltsverzeichnis sowie der Zusammenstellung der Optionen im Anhang (Abschnitt A.1 auf Seite 859) als Ausgangspunkt für die Suche von Informationen dienen.

Aus Gründen der Kompatibilität gibt es von fast jedem PSTricks-Paket eine TEX- (Endung .tex) und eine LATEX-Version (Endung .sty). Beide sind prinzipiell gleichwertig, wenn auch die LATEX-Styledatei in der Regel nichts anderes macht, als die TEX-Version mittels des \input Befehls zu laden. Die Datei pstricks.sty weicht davon ab, da sie einige grundlegende Tests ausführt (siehe Kapitel 2 auf Seite 9).

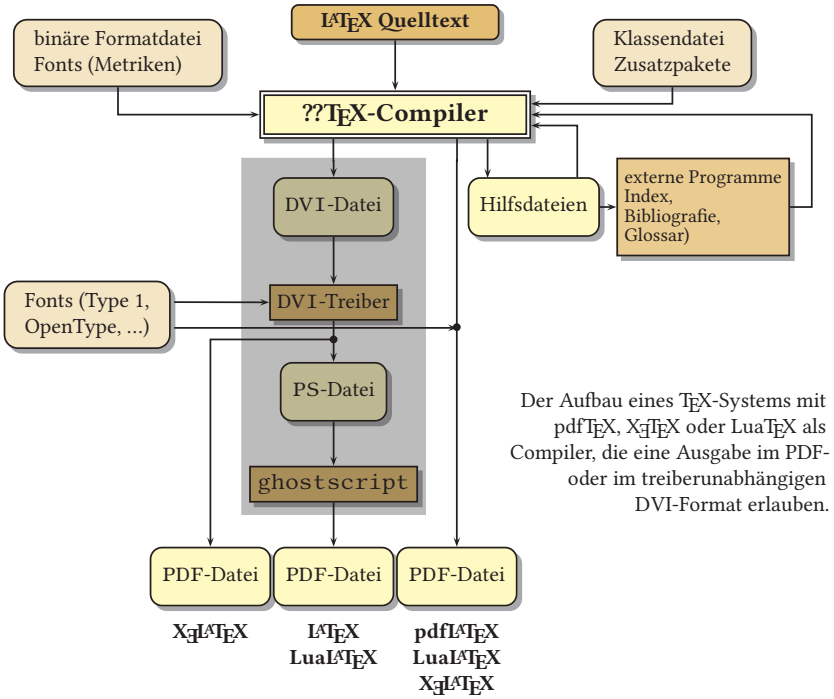
.sty und .tex

PSTricks verwendet sehr stark PS-Funktionen, die über den \special-Befehl vom dvips Programm, beziehungsweise dem DVIPS-Treiber an PS weitergereicht werden. Damit steht prinzipiell das komplette PS innerhalb von (LA)TEX zur Verfügung. Prinzipiell deshalb, weil eine Einschränkung gemacht werden muss; die Kommunikation zwischen TEX und PS ist einseitig, nämlich genau nur in dieser Richtung (Abschnitt 15.9 auf Seite 291). Nur mit zusätzlichen und relativ umständlichen Tricks ist es möglich, Informationen von PS wieder an TEX zurückzugeben. Dies impliziert vor allen Dingen auch die Fehlermeldungen von PS, über die man während eines TEX-Laufs keinerlei Kenntnis hat, lediglich der PS-Interpreter kann hier weitere Informationen liefern.

Das TEX-Logfile enthält *keinerlei* Informationen über eventuelle PS-Fehler, die erst beim Ausführen der PS-Datei auftreten! In Zweifelsfällen kann die explizite Ausführung der PS-Datei mit GhostScript hilfreich sein.

1.4 Was es nicht ist ...

PSTricks ist bezeichnenderweise eine Abkürzung für PS-Tricks, woraus folgt, dass eine Ausgabe im PS-Format die Regel ist (Dateiendung .ps). Welche Möglichkeiten dennoch vorhanden sind, das populäre PDF-Format zu erreichen, wird ausführlich im Anhang D auf Seite 897 erläutert. Der Standardablauf zur Erzeugung einer PDF-Ausgabe ist daher nicht der rechte Zweig in Abbildung 1.1 auf der nächsten Seite, sondern der linke, grau hinterlegte Bereich! Die Befehlssequenz ist daher immer latex⇒dvips⇒pdf2pdf, wobei diese Schritte durchaus von einem pdflatex intern ausgeführt werden können, ohne dass der Anwender dies explizit erkennt. Weitere Informationen dazu gibt es im Kapitel D auf Seite 897.



Der Aufbau eines TeX-Systems mit pdfTeX, XeTeX oder LuaTeX als Compiler, die eine Ausgabe im PDF- oder im treiberunabhängigen DVI-Format erlauben.

Abbildung 1.1: Schematischer Ablauf zur Erstellung eines PDF-Dokumentes.

Viele der PSTricks-Pakete weisen einen professionellen Charakter auf, können aber dennoch nicht mit Programmen wie AutoCAD, AutoSketch usw. verglichen werden. Die Anwendung von PSTricks macht prinzipiell nur Sinn, wenn die zu erstellende Grafik eine bestimmte, von Paket zu Paket variierende Komplexität nicht überschreitet. Der Anwender muss selbst entscheiden, wo er PSTricks einsetzen oder nicht einsetzen will.

1.5 Was noch zu sagen ist ...

Es wird versucht, zu jedem Abschnitt mindestens ein Beispiel anzugeben. Dies ist jedoch nicht ganz unkritisch, denn häufig sind die Makros, die für ein sinnvolles Beispiel benötigt werden, noch gar nicht behandelt worden. Für solche Fälle wird am Ende eines Kapitels oder Abschnitts eine Querverweisliste angegeben, die nicht nur auf die einzelnen verwendeten Makros, sondern auch auf ergänzende oder weiterführende Pakete verweist.

Erwartete Angaben zu den Parametern sind immer in kursiver Schrift zu finden, optionale Parameter sind zusätzlich mit einem *grauen Kasten* unterlegt. Bei den Eingaben treten folgende, in Tabelle 1.2 dargestellte Fälle auf.

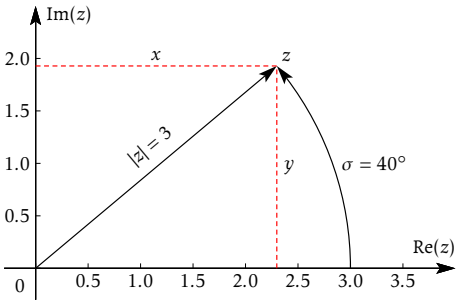
Die Verwendung von PSTricks, TeX und Farbe erfolgt nicht immer reibungslos. Um unnötige Irritationen zu vermeiden, sollte man bei der Verwendung von farbspezifischen

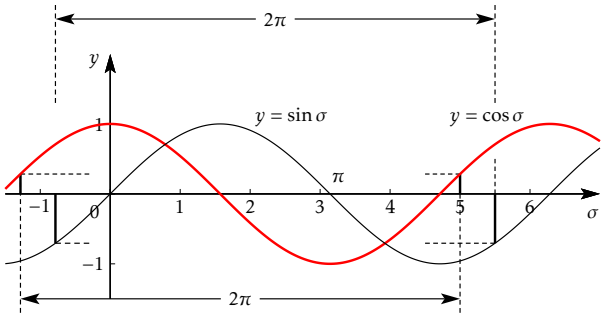
Tabelle 1.2: Konventionen bei der Eingabe

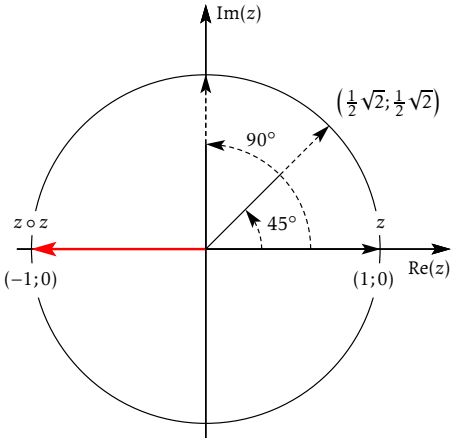
Beispiel	*Bedeutung*
Name	Name eines Parameters bzw. einer Option
Parameter	Feld für Optionen/Parameter
Pfeile	Angabe über die Art von Linien- oder Kurvenanfang bzw. Kurvenende
Text	beliebiger alphanumerischer Text
Material	beliebiges Material, wobei eventuell eine `\parbox` zu verwenden ist, wenn Zeilenumbrüche auftreten
Boolean	`false` oder `true`
Wert	Zahlenwert ohne Einheit
Wert Einheit	Zahlenwert *mit* Einheit
Wert *Einheit*	Zahlenwert *mit* oder *ohne* Einheit
Wert1 Wert2	Zwei Zahlenwerte durch Leerzeichen getrennt
Wert1 *Wert2*	Zweiter Zahlenwert optional
Winkel	Winkelangabe, entspricht einem Zahlenwert in Grad
Farbe	Farbname, der durch `PSTricks` oder durch `xcolor` bzw. `color` definiert sein muss
Länge	Ein mit `\newlength` definiertes Längenregister
x, y	Koordinatenpaar (Punkt)
x_1, y_1	1. Koordinatenpaar (Punkt) von mehreren
x_n, y_n	n-tes Koordinatenpaar (n-ter Punkt) von mehreren

Makros nur die Syntax des Pakets `xcolor` bzw. `color` verwenden und auf die `PSTricks`-eigene verzichten. Dies ist allerdings nur für LaTeX-Anwender der Fall, für plainTeX ist ausschließlich die `PSTricks`-Syntax zu verwenden. Mit dem Paket `miniltx` können auch plainTeX-Anwender die LaTeX-Syntax verwenden. Weitere Informationen dazu findet man in der aktuellen TeX-Installation unter `$TEXMF/tex/plain/graphics-pln/`.

01-05-1


Im(z)
2.0
1.5
1.0
0.5
0
x
z
|z| = 3
y
σ = 40°
Re(z)
0.5
1.0
1.5
2.0
2.5
3.0
3.5


2π
y
y = sin σ
y = cos σ
1
π
−1
0
1
2
3
4
5
6
σ
−1
2π


Im(z)
(½√2; ½√2)
90°
45°
z ∘ z
z
(−1; 0)
(1; 0)
Re(z)

Kapitel 2

Erste Schritte

PSTricks-Pakete werden in der üblichen Weise in ein Dokument eingebunden, wobei Tabelle 2.1 eine Zusammenstellung für die zugrundeliegenden Systeme zeigt.

Syntax	*System*
\input *pstricks*	für Plain TEX
\usepackage [*Optionen*] {*pstricks*}	für LATEX
\usemodule[*pstric*]	für ConTEXt

Tabelle 2.1: Laden von PSTricks-Paketen für verschiedene TEX-Systeme.

In der Regel sind fast alle PSTricks-Pakete Plain TEX-kompatibel, sodass die entsprechenden LATEX-Stildateien im Prinzip nichts anderes machen als die korrespondierende TEX-Datei zu laden. Eine Ausnahme bildet pstricks.sty, da es insbesondere wegen des Farbmanagements zusätzlich zum Laden von pstricks.tex mehrere Tests durchführt und anschließend einige Modifikationen vornimmt. Es stehen für pstricks.sty folgende Optionen zur Verfügung, wobei alle anderen Optionen an color oder xcolor weitergereicht werden:

noxcolor Anstelle von xcolor wird das Paket color geladen, der Vorläufer des xcolor Pakets.

plain Es wird nur ein \input{*pstricks*} ausgeführt, sämtliche Modifikationen an den Makros zur Handhabung von Farben unterbleiben. Es stehen dann nur die PSTricks-spezifischen Makros zur Definition von Farben zur Verfügung (siehe Abschnitt 2.1 auf der nächsten Seite)

`distiller`	Redefiniert die Makros für die Transparenz von Linien und Flächen für die Anwendung von Adobes Distiller.
`97`	Lädt die Version von 1997, womit ein Großteil aller neuen Funktionen nicht zur Verfügung steht.
`pdf`	Lädt automatisch das Paket `auto-pst-pdf`, sodass ein Dokument mit pdflatex und der zugehörigen Option `shell-escape` übersetzt werden kann. Eingehende Informationen gibt es im Abschnitt D.2 auf Seite 898.

Es existiert ein `pst-all` Paket, welches alle so genannten Basispakete von `PSTricks` lädt, wobei diese Zusammenstellung allerdings historisch bedingt ist und keiner inneren Logik folgt. Andererseits macht es keinen Sinn, diese Liste derart zu erweitern, dass wirklich »all« Pakete geladen werden; es sind mittlerweile zu viele und negative Abhängigkeiten sind nicht immer ausgeschlossen, die eine Fehlersuche erschweren.

Tabelle 2.2: Paketreihenfolge in `pst-all`

```
[ ... ]
\ProvidesPackage{pst-all}[2008/01/01 the main pstricks tools]
\RequirePackage{pstricks}  % important
% this loads the xcolor package and pstricks in the right order
% and does some modification to the color handling. Look at the
% doc for the options.
%
\RequirePackage{pst-plot}         \RequirePackage{pst-node}
\RequirePackage{pst-tree}         \RequirePackage{pst-grad}
\RequirePackage{pst-coil}         \RequirePackage{pst-text}
\RequirePackage{pst-3d}           \RequirePackage{pst-eps}
\RequirePackage[tiling]{pst-fill} \RequirePackage{pstricks-add}
\RequirePackage{multido}
\endinput
```

Bei größeren Projekten empfiehlt es sich, nicht `pst-all` zu benutzen, sondern die Pakete einzeln zu laden. Auf diese Weise können Fehler bei einem TEX-Lauf leichter behoben werden, indem gezielt einzelne Pakete nicht geladen werden. Weiteres dazu wird im nächsten Abschnitt behandelt werden.

2.1 Farben

TEX kennt bekanntermaßen selbst keine Farben, was in der Vergangenheit zur Entwicklung verschiedener Pakete führte, die nicht unbedingt immer fehlerfrei eingesetzt werden können. `PSTricks` greift auch hier auf die Möglichkeiten von PS zurück und definiert seine eigenen Farben, je fünf verschiedene Grauwerte und Farben (Tabelle 2.3).

Tabelle 2.3: Die vordefinierten Grauwerte und Farben von `PSTricks`.

Grauwerte	`black`, `darkgray`, `gray`, `lightgray`, `white`
Farben	`red`, `green`, `blue`, `cyan`, `magenta`, `yellow`

Diese können ohne jegliches Zusatzpaket innerhalb von `PSTricks` mit den oben angegebenen Namen als gleichnamiges Makro benutzt werden, da sie in jedem Fall auf PS-Ebene

definiert werden (vergleiche dazu Beispiel 02-01-1). Dennoch sollte das Folgende für ein Arbeiten unter LaTeX beachtet werden:

Die Kurzformen für die Farbsetzung, wie beispielsweise `\red`, werden zwar nach wie vor von `PSTricks` unterstützt, sollten aber für LaTeX-Anweder durch die Befehle `\color` beziehungsweise `\textcolor` ersetzt werden.

Neue Farben können durch die folgenden `PSTricks`-Makros definiert werden:

`\newgray{`*anotherGray*`}{`*Wert*`}`
`\newrgbcolor{`*anotherRGB*`}{`*Wert1 Wert2 Wert3*`}`
`\newhsbcolor{`*anotherHSB*`}{`*Wert1 Wert2 Wert3*`}`
`\newcmykcolor{`*anotherCMYK*`}{`*Wert1 Wert2 Wert3 Wert4*`}`

Diese Definitionen stehen nicht unbedingt in Einklang mit dem Paket `xcolor`, sodass `pstricks.sty` einige Modifikationen vornimmt, um dem LaTeX- Benutzer ein reibungsloses Arbeiten mit den Makros von `xcolor` zu ermöglichen.

Um Komplikationen zu vermeiden, sollten unbedingt folgende Punkte beachtet werden:

Die Datei `pstricks.sty` sollte stets *vor* allen `PSTricks`-basierten Paketen geladen werden. `pstricks.sty` selbst lädt bereits `pstricks.tex` *und* `xcolor` beziehungsweise `color`, sodass für LaTeX ein `\usepackage{`*pstricks*`}` ausreicht! Dies ist für TeX-Anwender uninteressant, da sie ohnehin die `PSTricks`-eigenen Farbdefinitionen verwenden müssen. Das RGB-Farbmodell wird von allen PS-Implementationen bestens unterstützt, HSB und CMYK nicht in allen Level-1-Implementationen von PS, wobei mittlerweile jeder Drucker Level 2 oder auch Level 3 unterstützen sollte.

Insbesondere im Zusammenhang mit der Dokumentenklasse `prosper`, die von `seminar` abgeleitet ist, gibt es wiederholt Probleme im Zusammenhang mit `PSTricks` und der Definition von Farben, was primär darauf zurückzuführen ist, dass diese Pakete sich nicht vollständig auf das `xcolor` bzw. `color` Paket beziehen, sondern ein eigenes Farbmanagement aufweisen, welches nicht kollisionsfrei zu `PSTricks` ist (siehe Abschnitt 13.2.6 auf Seite 169). Die Verwendung einer neueren und konfliktfreien Klasse wie `beamer`[1] oder insbesondere dem `prosper`-Nachfolger `powerdot`[2] ist in jedem Fall die bessere Variante (siehe hierzu auch Kapitel 34.2 auf Seite 813).

02-01-1

Das ist jetzt rot unter Verwendung der PSTricks-Syntax, jetzt nicht mehr.

```
\input{pstricks}% kein Farbpaket laden!

Das ist jetzt {\red rot unter Verwendung der
PSTricks-Syntax}, jetzt nicht mehr.
```

2.1.1 xcolor

Zwischen den Paketen `xcolor` von Uwe Kern und `color` von David Carlisle existiert ein wesentlicher Unterschied bei der Handhabung der Option `dvips`, denn `color` aktiviert grundsätzlich die `dvipsnames` Option, sobald einer der Treiber dvips, oztex oder xdvi ausgewählt wird. Dies kann zu Problemen führen, wenn man das Dokument mit pdftex übersetzt, welches dann undefinierte Farben reklamiert. Dies ist der Grund warum

[1] `CTAN:/macros/latex/contrib/beamer/`
[2] `CTAN:/macros/latex/contrib/powerdot/`

xcolor grundsätzlich vom Anwender eine explizite Angabe der Option dvipsnames erwartet, wenn die entsprechenden vordefinierten Farben benutzt werden sollen.

\usepackage[*dvipsnames,prologue*]{*xcolor*}

Tabelle 2.4: Zusammenstellung der Paketoptionen von xcolor.

Option	*Bedeutung*
natural	(Standard) Benutze alle Farben innerhalb ihres Modells, mit Ausnahme von RGB (konvertiert to rgb), HSB (konvertiert to hsb), und Gray (konvertiert to gray).
rgb	Konvertiert alle Farben in das rgb-Modell.
cmy	Konvertiert alle Farben in das cmy-Modell.
cmyk	Konvertiert alle Farben in das cmyk-Modell.
hsb	Konvertiert alle Farben in das hsb-Modell.
gray	Konvertiert alle Farben in das gray-Modell.
RGB	Konvertiert alle Farben in das RGB-Modell (und danach ins rgb-Modell).
HTML	Konvertiert alle Farben in das HTML-Modell (und danach ins rgb-Modell).
HSB	Konvertiert alle Farben in das HSB-Modell (und danach ins hsb-Modell).
Gray	Konvertiert alle Farben in das Gray-Modell (und danach ins to gray-Modell).
dvipsnames	Lädt die vordefinierten DVIPS-Farben.
svgnames	Lädt die vordefinierten SVG-Farben
x11names	Lädt die vordefinierten Unix/X11-Farben
prologue	schreibt die Liste der Farbnamen (dvipsnames) in den PS-Header, was für die Dokumentenerstellung via DVIPS wichtig ist.
kernelfbox	Benutze das ursprüngliche LaTeX-Makro.
table	Lädt das colortbl Packet, um farbige Tabellenzeilen zu ermöglichen.
hyperref	Unterstützung des hyperref Packets.
showerrors	(Standard) Gibt eine Meldung bei undefinierten Farben aus.
hideerrors	Gibt nur eine Fehlermeldung aus, wenn die undefinierte Farbe benutzt wird und setzt sie dann auf Schwarz.

Tabelle 2.5: Unterstützte Farbmodelle (L, M, N sind natürliche Zahlen).

Name	*Grundfarben*	*Parameterbereich*	*Standard*
rgb	red, green, blue	$[0,1]^3$	
cmy	cyan, magenta, yellow	$[0,1]^3$	
cmyk	cyan, magenta, yellow, black	$[0,1]^4$	
hsb	hue, saturation, brightness	$[0,1]^3$	

gray	gray	$[0,1]$	
RGB	Red, Green, Blue	$\{0,1,\ldots,L\}^3$	$L = 255$
HTML	RRGGBB	{000000,...,FFFFFF}	
HSB	Hue, Saturation, Brightness	$\{0,1,\ldots,M\}^3$	$M = 240$
Gray	Gray	$\{0,1,\ldots,N\}$	$N = 15$
wave	lambda (nm)	$[363,814]$	

Anwendung vordefinierter Farben

Analog zu den Farbdefinitionen von PSTricks gibt es auch für `xcolor` einige vom Farbmodell unabhängige vordefinierte Farben. Alle Treiber unterstützen logischerweise die Farben `black` und `white` sowie bei einer RGB-Unterstützung `red`, `green` und `blue`. Wird das CMYK-Modell unterstützt, dann noch zusätzlich die Farben `cyan`, `magenta` und `yellow`.

```
\color{Name}
\textcolor{Name}{Text}
```

Die erste Form arbeitet prinzipiell wie ein Schalter und definiert eine neue aktuelle Vordergrundfarbe bis entweder ein neuer `\color`-Befehl kommt oder eine lokale Gruppe beendet wird. Die zweite Form ist immer dann angebracht, wenn man nur kurze Textpassagen in einer anderen Farbe setzen will. Sie ist prinzipiell identisch zur Befehlsfolge {`\color{`*name*`}` ... *Text* ...}.

Definition von Farben über numerische Werte

Immer dann, wenn vordefinierte Farben nicht ausreichen, kann man Farben über numerische Werte bezogen auf ein bestimmtes Farbmodell definieren. Beide Makros erlauben ein optionales Argument für das Farbmodell, welches dann den Parameter nicht als Namen, sondern durch Komma getrennte Farbwerte erwartet.

```
\color [Modell] {Spezifikation}
\textcolor [Modell] {Spezifikation}{Text}
```

Das Beispiel aus dem vorhergehenden Abschnitt kann dann alternativ auch über die explizite Festlegung der Farbanteile im jeweiligen Modell geschrieben werden.

02-01-2

Nun ist der Text grün und nun ein wenig magenta und jetzt blau und nun wieder grün.

```
\usepackage[noxcolor]{pstricks}% color statt xcolor

Nun ist der Text \color[rgb]{0,1,0} grün und
nun \textcolor[cmyk]{0,1,0,0}{ein wenig magenta}
und {\color[rgb]{0,0,1}jetzt blau} und nun
wieder grün.
```

Das folgende Beispiel benutzt Farben aus verschiedenen Modellen, sowie einige der erwähnten vordefinierten Farben.

– magenta cmyk und schwarz
– vordefiniertes blau grauer Text

```
\usepackage{pstricks}% laedt xcolor

-- \textcolor[cmyk]{0,1,0,0}{magenta cmyk} und schwarz\\
-- \color[gray]{0.2}%
      \textcolor{blue}{vordefiniertes blau} grauer Text
```

02-01-3

Grundsätzlich gilt zu beachten, dass die Verwendung von Farben in bestimmten Modellen nicht unbedingt dem Austausch von Dokumenten förderlich ist; das Endergebnis in PS- oder PDF-Form hängt stark von den verwendeten Treibern ab. Die sichere Variante ist die Definition eigener Farben innerhalb der Präambel und anschließende Anwendung nur über den Farbnamen.

Definieren von Farben

Die Farben `black`, `white`, `red`, `green`, `blue`, `cyan`, `magenta` und `yellow` sollten von jedem beliebigen Treiber definiert werden, sodass der Anwender sie direkt über ihre Namen benutzen können sollte. In der Regel definieren Pakete noch weitere Farben über die man sich in der entsprechenden Dokumentation informieren kann. Mit `xcolor` stehen mehrere Makros für weitere Farbdefinitionen zur Verfügung:

```
\providecolor [Typ] {Name}{Modell}{Farbspezifikation}
\providecolorset [Typ] {Modell}{Präfix}{Suffix}{Setspezifikation}
\definecolor [Typ] {Name}{Modell}{Farbspezifikation}
\colorlet{Name} [num Modell] {Farbe}
\definecolorset [Typ] {Modell}{Präfix}{Suffix}{Setspezifikation}
\definecolorseries{Name}{Basismodell} [Methode] {b-Modell}
                  {b-Spezif.} [s-Modell] {s-Spezif.}
```

Weiterhin definiert `xcolor` die in PS als *current color* bezeichnete aktuelle Farbe durch den Punkt, auf den in derselben Weise zugegriffen werden kann, wie über normale Farbnamen. Im folgenden Beispiel wird mehrere Male hintereinander die Farbe auf 80 % (`\color{.!80}`) ihres aktuellen Wertes verändert, was letztlich zur Farbe Weiß führen würde. Die folgende `\colorbox` wiederum benutzt die komplementäre aktuelle Farbe (`-.`) für den Hintergrund um den Index `\iCol` lesbar zu gestalten.

1 2 3 4 5 6 7 8 9 10 11

```
\usepackage{pstricks,multido}

\newcommand\CBox[1]{%
  \color{.!80}\colorbox{.}{{\color{-.}#1}}}
\multido{\iCol=1+1}{11}{\CBox{\strut\large\iCol}}
```

02-01-4

Im Gegensatz zu Makronamen dürfen Farbnamen ohne weiteres Ziffern enthalten, was insbesondere für Farbnamen wie *Grau30* hilfreich sein kann. `xcolor` erlaubt weitere Zeichen, wovon man allerdings keinen Gebrauch machen sollte, um für andere Pakete oder zukünftige Entwicklungen kompatibel zu bleiben.

Die Wirkungsweise der beiden Makros `\providecolor` und `\providecolorseries` ist analog zum bekannten LaTeX Makro `\providecommand`. Wenn eine Farbe oder eine Farbserie bereits unter dem angegebenen Namen existiert, erfolgt keine Neudefiniton, womit ein Überschreiben verhindert werden kann.

```
\definecolor{MyOrange}{cmyk}{0,0.42,1,0}
\definecolor[named]{Blue}{rgb}{0,0,0.8}
\providecolor{MyGrey}{gray}{0.75}
\definecolor{MyBlack}{named}{Black}
\colorlet{MyRGBO}[rgb]{MyOrange}
```

Mit der Definition obiger Farben stehen *MyOrange*, *Blue*, *MyGrey*, *MyBlack* und *MyRGBO* allgemein zur Verfügung und können zusätzlich zu den vordefinierten benutzt werden, wie im folgenden Beispiel zu sehen ist, welches über die Paketoptionen `dvipsnames` und `prologue` die zusätzlichen Farbnamen des dvips Programms verfügbar macht. `\colorlet` benutzt die vorher im CMYK-Farbmodell definierte Farbe *MyOrange* als Grundlage. Dazu wird diese in das RGB-Modell umgerechnet und dann als *MyRGBO* bezeichnet.

02-01-5

```
\usepackage[dvipsnames,prologue]{xcolor}
\definecolor{MyOrange}{cmyk}{0,0.42,1,0}
\definecolor[named]{Blue}{rgb}{0,0,0.8}
\definecolor{MyGrey}{gray}{0.75}
\definecolor{MyBlack}{named}{Black}
\colorlet{MyRGBO}[rgb]{MyOrange}
\newcommand*\col[1]{\color{#1}\rule{3cm}{5mm}}

{\col{MyOrange}}\\{\col{Blue}}\\
{\col{MyGrey}}\\{\col{MyBlack}}\\{\col{MyRGBO}}
```

Im Zusammenhang mit `PSTricks` würde man die Optionen erst dem Paket `pstricks` übergeben, welches dann diese an `xcolor` weiterreicht, wobei die Option `table` nur bei farbigen Tabellen notwendig ist, wie beispielsweise in Tabelle 2.4 und 2.5 auf Seite 12.

```
\usepackage[dvipsnames,prologue,table]{pstricks}
```

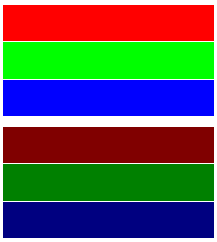
Farbserien (colorsets) sind immer dann hilfreich, wenn man Farbverläufe anwenden möchte. `xcolor` unterstützt die Definition derartiger Serien mit drei Makros, wobei zwei nichts weiter als eine erweiterte Version von `\definecolor` sind.

```
\definecolorset{rgb}{}{}{red,1,0,0;green,0,1,0;blue,0,0,1}
\providecolorset{rgb}{}{H}{red,0.5,0,0;green,0,0.5,0;blue,0,0,0.5}
```

Das erste Beispiel definiert die drei RGB-Basisfarben und das zweite drei neue Basisfarben mit dem Suffix *H*: *redH*, *greenH* und *blueH*. `\providecolorset` hätte man auch durch `\definecolorset` ersetzen können, denn es existierten noch keine entsprechenden Farbnamen. Umgekehrt wäre das nicht möglich gewesen, denn die drei RGB-Basisfarben `red`, `green` und `blue` sind sowohl durch `PSTricks` als auch `xcolor` bereits definiert.

```
\usepackage{xcolor}
\definecolorset{rgb}{}{}{red,1,0,0;green,0,1,0;blue,0,0,1}
\providecolorset{rgb}{}{H}{red,0.5,0,0;green,0,0.5,0;blue,0,0,0.5}
\newcommand*\col[1]{\color{#1}\rule{3cm}{5mm}}
```

```
{\col{red}}\\    {\col{green}}\\    {\col{blue}}\\[4pt]
{\col{redH}}\\   {\col{greenH}}\\   {\col{blueH}}
```



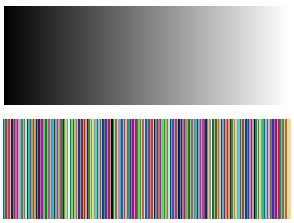
02-01-6

Eine ganze Farbserie lässt sich mit `\definecolorseries` festlegen, wobei der Anwender sowohl Farbanfang und Farbende als auch die Anzahl der Zwischenschritte vorgeben kann, beispielsweise eine Serie von Schwarz nach Weiß in 200 Schritten:

```
\definecolorseries{testA}{rgb}{last}{black}{white}
\resetcolorseries[200]{testA}%  defines the series with 200 colors
```

Mit dem Makro `\definecolorseries` wird diese Serie »logisch« definiert, während `\resetcolorseries` dies erst »physikalisch« durchführt, indem es die Zwischenschritte berechnet und alle Teilfarben intern in einer Art Feld definiert. Dabei kann man jederzeit auf einzelne Farben des Feldes zugreifen:

`\testA!![`*Index*`]`



02-01-7

```
\usepackage{xcolor,multido}
\definecolorseries{testB}{rgb}{step}[rgb]{%
    0.95,0.85,0.55}{0.17,0.47,0.37}
\resetcolorseries[200]{testB}
\definecolorseries{testA}{rgb}{last}{black}{white}
\resetcolorseries[200]{testA}% 200 Farbschritte
\linethickness{0.004\linewidth}

\multido{\nC=1+1}{200}{\hspace{0.004\linewidth}%
    \color{testA!![\nC]}\line(0,1){40}}\\[5pt]
\multido{\nC=1+1}{200}{\hspace{0.004\linewidth}%
    \color{testB!![\nC]}\line(0,1){40}}
```

Wie die Zwischenwerte berechnet werden, hängt von der gewählten Methode ab:

- {*b-Modell*}{*b-Spezif.*} Angabe der ersten Farbe.
- {*s-Modell*}{*s-Spezif.*} Berechnung der Zwischenschritte in Abhängigkeit von `method`:
  - `step`, `grad`: Das optionale Argument ist bedeutungslos und {*s-Spezif.*} ist ein Vektor, dessen Dimension durch {*Basismodell*} festgelegt ist, beispielsweise `[hsb]{0.1,-0.2,0.3}`.
  - `last`: Angabe der letzten Farbe, beispielsweise `[rgb]{0.1,0.5,0.5}`.

Wie die Berechnung der Zwischenschritte genau erfolgt, kann der Dokumentation zu `xcolor` entnommen werden. [41] Die einfachste Methode eine Farbserie zu definieren ergibt sich bei Verwendung der `last`-Option, wie es auch in den folgenden beiden Beispielen gezeigt wird. Das erste bezieht sich auf das CMYK und das zweite auf das HSB-Modell.

```
\usepackage{xcolor,multido}
\definecolorseries{testC}{cmyk}{last}{white}[cmyk]{1,0,0,0}
\resetcolorseries[10]{testC}
\definecolorseries{testM}{cmyk}{last}{white}[cmyk]{0,1,0,0}
\resetcolorseries[10]{testM}
\definecolorseries{testY}{cmyk}{last}{white}[cmyk]{0,0,1,0}
\resetcolorseries[10]{testY}
\definecolorseries{testK}{cmyk}{last}{white}[cmyk]{0,0,0,1}
\resetcolorseries[10]{testK}
\setlength{\fboxsep}{2mm}

\makebox[30mm][l]{cyan (C):}%
\multido{\nColr=0+1}{10}{\colorbox{testC!![\nColr]}{\strut0.\nColr}}\\
\makebox[30mm][l]{magenta (M):}%
\multido{\nColr=0+1}{10}{\colorbox{testM!![\nColr]}{\strut0.\nColr}}\\
\makebox[30mm][l]{yellow (Y):}%
\multido{\nColr=0+1}{10}{\colorbox{testY!![\nColr]}{\strut0.\nColr}}\\
\makebox[30mm][l]{black (K):}%
\multido{\nColr=0+1}{10}{\colorbox{testK!![\nColr]}{\strut0.\nColr}}
```

02-01-8

cyan (C):	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
magenta (M):	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
yellow (Y):	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
black (K):	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9

```
\usepackage{xcolor,multido}
\definecolorseries{testH}{hsb}{last}[hsb]{0,1,1}[hsb]{1,1,1}
\resetcolorseries[10]{testH}
\definecolorseries{testS}{hsb}{last}[hsb]{.1,0,1}[hsb]{.1,1,1}
\resetcolorseries[10]{testS}
\definecolorseries{testB}{hsb}{last}[hsb]{1,1,0}[hsb]{1,1,1}
\resetcolorseries[10]{testB}
\setlength{\fboxsep}{2mm}

\makebox[30mm][l]{Hue (H):}%
\multido{\nColr=0+1}{10}{\colorbox{testH!![\nColr]}{\strut0.\nColr}}\\
\makebox[30mm][l]{Saturation (S):}%
\multido{\nColr=0+1}{10}{\colorbox{testS!![\nColr]}{\strut0.\nColr}}\\
\makebox[30mm][l]{Brightness (B):}%
\multido{\nColr=0+1}{10}{\colorbox{testB!![\nColr]}{\strut\color{white}0.\nColr}}\\
```

02-01-9

Hue (H):	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Saturation (S):	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Brightness (B):	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9

Zwischen der Definition einer Farbserie und einer einzelnen Farbe gibt es einen wesentlichen Unterschied: `\definecolor` wirkt grundsätzlich lokal zur Gruppe und `\definecolorseries` grundsätzlich global.

Farbspezifikation

Der eigentliche Vorteil von `xcolor` gegenüber dem alten `color` Paket wird offensichtlich, wenn man die vielfältigen Möglichkeiten der Farbspezifikation bei `xcolor` betrachtet.

- »specification by name«, wobei die aktuelle Farbe den Punkt (`.`) als »Namen« hat.
- »specification by expression« mit der Syntax:
  PräfixName`!`*Wert1*`!`*Name1*`!` ... `!` *WertN*`!`*NameN**Suffix*

Präfix Ist dieses ein Minuszeichen (»−«), dann erfolgt vor Anwendung eine Umwandlung in die Komplementärfarbe.

Name Modell- und Farbparameter von *Name* bilden eine temporäre Farbe `\temp`.

Wert1`!`*Name1* Die neue Farbe ergibt sich aus *Wert1* % der Farbe *temp* und (100-*Wert1*) % von der Farbe *Name1* und wird dann wieder als neue temporäre Farbe `\temp` gespeichert. Dieser Schritt wird für alle weiteren *!Wert!Name* wiederholt. Ist ein *Suffix* definiert, dann ist die temporäre Farbe die zur Farbserie *Name* korrespondierende.

Suffix Kann eine der Formen `!!+`, `!!++`, `!!+++`, usw. annehmen. Die Anzahl der Pluszeichen (+) bezieht sich auf die zugrundeliegende Farbserie.

Tabelle 2.6 auf der nächsten Seite zeigt einige Beispiele für Farbausdrücke und ihren äquivalenten Ausdruck als RGB-Code. Die zweite Hälfte der Tabelle zeigt die Anwendung der komplementären Farben, die jeweils zu denen der ersten Hälfte addiert werden. Farbe plus Komplementärfarbe addiert, ergibt bekanntlich Weiß mit dem RGB-Wert »1 1 1«. Wenn nur noch eine Farbe übrig ist, wird die letzte folgendermaßen berechnet:

$$\vec{c} = (1-p)\cdot\vec{e} + p\cdot(x,y,z) \tag{2.1}$$

wobei p der numerische Prozentwert ist, $\vec{e}$ der Einheitsvektor und (x,y,z) das gegebene RGB-Tripel. Für *red!75* erhält man dann $\vec{c} = (1-0.75)\cdot\vec{e} + 0.75\cdot(1,0,0) = (1, 0.25, 0.25)$, was allgemein als 75 % von `red` bezeichnet wird. Die dritte Tabellenzeile mit dem Farbausdruck `red!75!blue!100` wird zu einem Farbvektor aus der Addition von $\vec{c} = 0.75(1,0,0) + (1-0.75)(0,0,1) = (0.75, 0, 0.25)$. Oder mit anderen Worten: 75 % von `red` und 25 % von `blue`. Ein Wert von 100 ändert nicht die Farbe, denn daraus folgt die Vektoraddition $\vec{c} = 1\cdot(x_1,y_1,z_1) + 0\cdot(x_2,y_2,z_2) = (x_1,y_1,z_1)$. Zum besseren Verständnis soll noch einmal ausführlich die Farbberechnung der vierten Zeile aus Tabelle 2.6 gezeigt werden.

$$\begin{aligned}
(\text{red!75!blue}) &= 0.75(1,0,0) + (1-0.75)(0,0,1) && (2.2)\\
&= 0.75, 0, 0.25 = \vec{c}_{\text{temp}} && (2.3)\\
(0.75,0,0.25)!40 &= (1-0.4)\vec{e} + 0.4\cdot(0.75,0,0.25) && (2.4)\\
&= 0.6\vec{e} + (0.3,0,0.1) && (2.5)\\
&= (0.9, 0.6, 0.7) = \vec{c} && (2.6)
\end{aligned}$$

Farbausdruck	*RGB-Tripel*		
red	1	0	0
red!75	1	0.25	0.25
red!75!blue!100	0.75	0	0.25
red!75!blue!40	0.9	0.6	0.7
red!75!blue!40!yellow!50	0.95	0.8	0.55
-red	0	1	1
-red!75	0	0.75	0.75
-red!75!blue!100	0.25	1	0.75
-red!75!blue!40	0.1	0.4	0.3
-red!75!blue!40!yellow!50	0.05	0.2	0.45

Tabelle 2.6: Farbausdrücke und ihr entsprechendes RGB-Tripel.

Das folgende Beispiel zeigt eine Anwendung der Farbausdrücke innerhalb von PSTricks und Tabelle 2.7 zeigt weitere Beispiele bei Anwendung des \fcolorbox Makros, beispielsweise für die letzte Zeile:

```
\fcolorbox{black}{-red!75!green!50!blue!25!gray}{\phantom{aa}}
```

02-01-10


```
\usepackage{pstricks}

\psframebox[linecolor={red!70!green},
 fillcolor=yellow!90!cyan, fillstyle=solid,
 doubleline=true,doublesep=5pt,framesep=10pt,
 doublecolor=-yellow!90!cyan]{%
  \Large PST\textcolor{%
   red!72.75}{PST}\color{-green}PST}
```

Tabelle 2.7: Beispiele für Farbausdrücke.

red	-red
red!75	-red!
red!75!green	-red!75!green
red!75!green!50	-red!75!green!50
red!75!green!50!blue	-red!75!green!50!blue
red!75!green!50!blue!25	-red!75!green!50!blue!25
red!75!green!50!blue!25!gray	-red!75!green!50!blue!25!gray

Zusätzlich zu den »normalen« Farbausdrücken, kennt xcolor noch die »erweiterten« Farbausdrücke, die sich an einer Farbmischung orientieren:

core model,*suml*:*expr1*,*fac1*;*expr2*,*fac2*;…;*exprN*,*facN*

Jede Farbe in dem Ausdruck enthält einen Faktor, der die Gewichtung der Farbe darstellt. Für eine derartige erweiterte Farbdefinition wird jede Farbe *k* in das Basismodell

umgerechnet und danach mit dem Faktor multipliziert:

$$\frac{\mathrm{dec}_i}{\sum_{i=0}^{n} \mathrm{fac}_i}, \qquad \text{mit} \quad i \in [1;n] \tag{2.7}$$

Tabelle 2.8 zeigt ähnlich zu Tabelle 2.7 einige Beispiele zu den möglichen Farbausdrücken.

Tabelle 2.8: Beispiele für erweiterte Farbausdrücke.

`red`	`-red`
`rgb:red,1;white,2`	`cmyk:red,2;white,2`
`rgb:red,5;green,2;yellow,10`	`cmyk:red,5;green,2;yellow,10`
`rgb,11:red,5;green,2;cyan,1`	`cmyk,11:red,5;cyan,2;cyan,1`

Anwendung vordefinierter Farben (»named colors«)

Die so genannten »named colors«, die der zugrundeliegende Farbtreiber, beispielsweise dvips zur Verfügung stellt, erlauben eine vereinfachte Anwendung bei der Neudefinition von Farben: `\color[`*named*`]{`*SpringGreen*`}` wählt die vordefinierte Farbe `SpringGreen`.

Alternativ kann auch eine neue Farbe mit gleicher Eigenschaft definiert werden; dann braucht die Option `named` nicht jedes Mal angegeben zu werden. Formal wird nichts anderes als ein Aliasname für diese Farbe erzeugt, beispielsweise:

```
\definecolor{MyGreen}{named}{SpringGreen}
```

Danach kann einfach `\color{`*MyGreen*`}` verwendet werden. Sämtliche »named colors« können über die Paketoption angefordert werden, womit automatisch vom Paket entsprechende Farben definiert werden. Allerdings muss dem `xcolor`-Paket noch gesagt werden, dass es die so genannte Headerdatei `xcolor.pro` mit in die PS-Ausgabe schreiben soll, falls man mit dvips arbeitet. Dies kann über die Paketoption `prologue` erreicht werden. Eine Zusammenstellung der verfügbaren Farbnamen für das `dvipsnames`- als auch das `svgnames`-Farbmodell zeigen die Tabellen 35.5 und 35.6 auf Seite 820.

Farben in Boxen

Farbdefinitionen, die innerhalb einer `\savebox` erfolgen, werden grundsätzlich beim Schreiben in die Box ebenfalls in dieser gespeichert und sind somit lokal und können nicht mehr von außen überschrieben werden.

```
\usepackage{xcolor}
\newsavebox{\X}
\sbox{\X}{[Schwarz] und \color[cmyk]{0,0.6,0.8,0}[Orange]}

Start mit \usebox{\X}, und zurück zu Schwarz.\\
\color{green}Start mit Grün, siehe \usebox{\X} und wieder Grün.
```

Start mit [Schwarz] und [Orange], und zurück zu Schwarz.
Start mit Grün, siehe [Schwarz] und [Orange] und wieder Grün.

02-01-11

Es kommt immer wieder zu Problemen, wenn derartige Boxen mit Plain TEX weiterverarbeitet werden. Man befindet sich immer auf der sicheren Seite, wenn ausschließlich LATEX-Makros verwendet werden. Die Probleme werden eingehend in [73] und [65] besprochen.

2.1.2 Seitenhintergrundfarbe

`\pagecolor{`*Name*`}`

Die Hintergrundfarbe der gesamten Seite kann mit `\pagecolor` verändert werden, wobei dieselbe Syntax wie bei `\color` verwendet wird. Sowohl die aktuelle als auch alle folgenden Seiten werden auf die angegebene Hintergrundfarbe gesetzt, wobei diese Definition grundsätzlich global wirkt und somit innerhalb einer `minipage` oder Gruppe keinen Sinn macht.

2.1.3 Boxhintergrundfarbe

Analog zum bekannten Makro `\fbox` existieren zwei Makros zum Setzen der Hintergrundfarbe von Boxen, wobei `\fcolorbox` zusätzlich das Setzen der Rahmenfarbe ermöglicht.

`\colorbox{`*Hintergrundfarbe*`}{`*Text*`}`
`\fcolorbox{`*Rahmenfarbe*`}{`*Hintergrundfarbe*`}{`*Text*`}`

```
\usepackage{xcolor}
\definecolor{Light}{gray}{.80}  \definecolor{Dark}{gray}{.20}

\colorbox{red}{Schwarz auf Rot}\hfill%
\fcolorbox{red}{cyan}{Black -- Text, cyan -- Hintergrund, red -- Rahmen}\\
\colorbox{Light}{\textcolor{Dark}{Heller Hintergrund}}\hfill%
\fcolorbox{red}{cyan}{\color{white}White -- Text, cyan -- Hintergrund,
    red -- Rahmen}\\
\colorbox{Dark}{\textcolor{white}{Dunkler Hintergrund}}
```

02-01-12

Schwarz auf Rot

Black – Text, cyan – Hintergrund, red – Rahmen

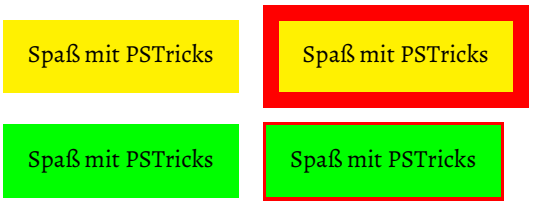
Heller Hintergrund

White – Text, cyan – Hintergrund, red – Rahmen

Dunkler Hintergrund

Die weiteren Beispiele zeigen die Handhabung der `\fbox`-Parameter `\fboxrule` und `\fboxsep`, welche zum einen die Liniendicke und zum anderen den Abstand zwischen innerem Box-Text und Rahmen festlegen.

```
\usepackage{xcolor}

\setlength{\fboxsep}{10pt}\setlength{\fboxrule}{6pt}
\colorbox{yellow}{Spaß mit PSTricks}\quad
\fcolorbox{red}{yellow}{Spaß mit PSTricks}\\[5pt]
\setlength{\fboxrule}{1pt}\colorbox{green}{Spaß mit PSTricks}\quad
\fcolorbox{red}{green}{Spaß mit PSTricks}
```



02-01-13

2.1.4 Farbwerte bestimmen

Mithilfe des `xcolor`-Pakets lassen sich auf einfache Art und Weise Farbwerte oder auch Farbserien bestimmen (Abschnitt 2.1.1). Möchte man beispielsweise eine Farbe analog zu einer gegebenen HTML-Seite definieren, so kann dies mit dem gleichnamigen Modell geschehen, wobei man sich die entsprechenden Werte in einem anderen Modell mithilfe von `\convertcolorspec` transformieren und ausgeben lassen kann.

```
HTML-Farbe 006666
rgb :  0,0.4,0.4
cmyk:  0.4,0,0,0.6
hsb :  0.5,1,0.4
```

```
\usepackage{xcolor}

\definecolor{HTMLFarbe}{HTML}{006666}% #006666
\ttfamily{\color{HTMLFarbe}HTML-Farbe 006666}\\
rgb : \convertcolorspec{HTML}{006666}{rgb}\RGBFarbe \RGBFarbe\\
cmyk: \convertcolorspec{HTML}{006666}{cmyk}\RGBFarbe\RGBFarbe\\
hsb : \convertcolorspec{HTML}{006666}{hsb}\RGBFarbe \RGBFarbe
```

02-01-14

2.1.5 `xcolor` und `PSTricks`

Für die Anwendung von `xcolor` innerhalb von `PSTricks` gibt es einige Besonderheiten. Die Farbfestlegung über das optionale Argument eines Makros erfordert ein Einschließen der Farbwerte in Klammern, wenn das Farbmodell zusätzlich festgelegt wird, was im dritten Beispiel gezeigt wird. Bei fehlenden geschweiften Klammern wird das Argument nicht korrekt eingelesen und es kommt zu einer Fehlermeldung.

```
\psset{linecolor=green!50}
\psset{linecolor={[rgb]{0.5,1,0.5}}}   % Klammerung beachten!
\psframebox[linecolor={[rgb]{0.5,1,0.5}}]{foo}
```

Zusätzlich zu den in Tabelle 2.5 auf Seite 12 festgelegten Farbmodellen, existiert noch das ausschließlich `PSTricks`-spezifische `ps`, welches im eigentlichen Sinne kein Farbmodell ist, sondern die Farbwerte unbearbeitet lässt und einfach nach PS durchreicht. Dadurch lassen sich die Farbwerte auf der PS-Seite manipulieren, bevor sie angewendet werden.

\definecolor [*ps*] {*Name*}{*Basismodell*}{*PS Code*}

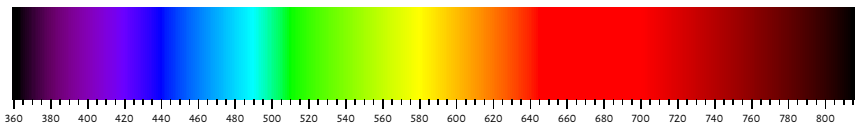
Die Angabe des Basismodells nimmt direkt Bezug auf den unter PS definierten Befehl wie `setrgbcolor`. Wird beispielsweise eine »Farbe« *foo* definiert als

`\definecolor[`*ps*`]{`*foo*`}{`*cmyk*`}{`*bar*`}`

so wird bei einem `\psline[`*linecolor=foo*`](`x_3,y_3`)` von `xcolor` nur das Argument mit dem entsprechenden PS-Befehl durchgereicht: `bar setcmykcolor`. Es obliegt dem Anwender, dass er vorher `bar` auf der PS-Seite definiert hat.

```
\usepackage{pstricks}
\newcount\WL \unitlength.75pt
\def\WaveToPS#1{%
  \definecolor{tmp}{wave}{#1}\extractcolorspec{tmp}\tmp
  \expandafter\convertcolorspec\tmp{rgb}\tmp \expandafter\WaveToPSi\tmp,}
\def\WaveToPSi#1,#2,#3,{\pstVerb{/Red {#1} def /Green {#2} def /Blue {#3} def}}
\pstVerb{/Corr {dup 0 gt {Gamma exp} if } def }
\definecolor[ps]{lambda}{rgb}{Red Corr Green Corr Blue Corr}

\begin{picture}(510,70)(310,-10)\sffamily\tiny
  \linethickness{1.25\unitlength}\WL=360
  \pstVerb{/Gamma .8 def}\multiput(320,0)(1,0){456}{%
    \WaveToPS{\the\WL}{\color{lambda}\line(0,1){50}}\global\advance\WL1}%
  \linethickness{0.25\unitlength}\WL=360
  \multiput(320,0)(20,0){23}{%
    \picture(0,0)
      \line(0,-1){5}\multiput(5,0)(5,0){3}{\line(0,-1){2.5}}%
      \put(0,-10){\makebox(0,0){\the\WL}}\global\advance\WL20
    \endpicture}%
\end{picture}
```

02-01-15



2.2 Parameter mit \psset setzen

`PSTricks` macht zur Festlegung der Parameter bzw. Optionen intensiven Gebrauch vom »key-value«-Interface (siehe Abschnitt 13.6 auf Seite 179), was für den Anwender angenehme Erleichterungen bringt, denn er kann alle Einstellungen auch global mit dem `\psset`-Makro ändern. Die Syntax ist relativ einfach:

\psset [*Paket*] {*Name1=Wert1,Name2=Wert2,...*}

Die optionale Angabe des Paketes ist nur dann zwingend, wenn verschiedene Pakete eine Option mit demselben Namen aber unterschiedlicher Bedeutung definieren. Im

Allgemeinen sind die Paketautoren bemüht, Parameternamen so zu wählen, dass sie nicht gleichzeitig in anderen Paketen mit einer anderen Bedeutung verwendet werden. Dennoch muss sich der Anwender darum kümmern, ob dies eventuell der Fall sein könnte, wenn Fehler in einem Dokument auftreten, die nicht lösbar erscheinen. Alternativ können diese Parameter auch über das optionale Argument in der für LaTeX üblichen Weise übergeben werden, hier exemplarisch für `\psline` gezeigt:

`\psline` [*Optionen*] `(...)(...)`

- Parameter, die über das optionale Argument eines Makros übergeben werden, bleiben *lokal*, sind aber für tiefer liegende Ebenen global.
- Parameter, die mit `\psset` gesetzt werden, sind *global* innerhalb der Gruppe und tiefer liegenden Ebenen.
- Durch eine Gruppenbildung kann man auch mit `\psset` gesetzte Variablen lokal halten, womit die folgenden beiden Anweisungen identisch sind:
  ```
  \psline[linewidth=5pt](3,3)
  {\psset{linewidth=5pt}\psline(3,3)}
  ```
- Historisch bedingt kennt `PSTricks` eine weitere Möglichkeit, Optionen anzugeben, nämlich mit den geschweiften Klammern:

  `\psline` [*Optionen*] {*Pfeilart*} `(...)(...)`

  Dies trägt nicht immer zur Übersichtlichkeit bei, zumal die Angabe der Pfeilart auch über Angaben in den eckigen Klammern oder `\psset` erfolgen kann.

Bis auf die Makros, deren Name mit `\q` beginnt, verfügen fast alle über eine Sternvariante, die prinzipiell einer inversen Darstellung des jeweiligen Objekts entspricht. Das Objekt wird mit der aktuellen Linienfarbe (`linecolor`) gefüllt, wobei folgende in Tabelle 2.9 zusammengefasste Optionen gesetzt werden:

Tabelle 2.9: Auswirkungen der Sternoption bei Makros auf andere Optionen, beispielsweise `\psframe*`.

`linewidth`	`0pt`
`fillcolor`	`\pslinecolor`
`fillstyle`	`solid`
`linestyle`	`none`

Die Existenz einer Sternversion beruht ausschließlich auf softwaretechnischen Kriterien, die völlig unabhängig von der Sinnhaftigkeit sind. So existieren von mehreren Makros Sternversionen, die faktisch bedeutungslos sind, beispielsweise `\psline*`.

2.3 Maßstäbe und Längen

2.3.1 Längen

Der voreingestellte Maßstab beträgt 1cm und kann getrennt für die x- und y-Richtung mit `\psset` oder über die jeweilige Makrooption auch lokal verändert werden. Eine Zusammenstellung zeigt Tabelle 2.10. Der Vorteil von `unit` ist, dass damit auf einfache Weise eine Skalierung der gesamten `pspicture`-Umgebung erfolgen kann.

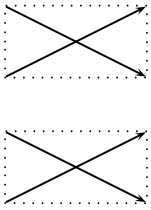
Tabelle 2.10: Längeneinheiten und ihre Registernamen in PSTricks.

Optionsname	*Bedeutung*	*Voreinstellung*	*Längenregister*
unit	alle zusammen	1 cm	\psunit
xunit	x-Achse	1 cm	\psxunit
yunit	y-Achse	1 cm	\psyunit
runit	Radius (Bogenmaß)	1 cm	\psrunit

Die Anweisung \psset{*xunit=1cm, yunit=1cm, runit=1cm*} ist daher per Definition identisch zu \psset{*unit=1cm*} und umgekehrt entsprechend.[3] Wird eine Längeneinheit *vor* Beginn der pspicture-Umgebung verändert, so beziehen sich auch die Koordinaten von pspicture auf diesen geänderten Maßstab! Im folgenden Beispiel ergibt sich dasselbe Ergebnis, obwohl völlig unterschiedliche Werte für die pspicture-Umgebung genommen wurden. Bei dimensionslosen Angaben ist darauf zu achten, dass sich die Werte nur dann auf die Vorgabe cm beziehen, wenn nicht zwischenzeitlich etwas anderes vereinbart wurde. Die zweite Variante hat den Vorteil, dass die Angaben für die pspicture-Umgebung und die folgenden Makros sich auf dieselben Einheiten beziehen.

Längeneinheit

02-03-1



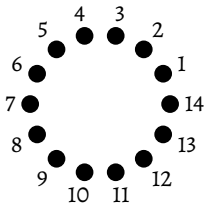
```
\usepackage{pstricks}

\psframebox[framesep=0pt,linestyle=dotted]{%
\begin{pspicture}(2,1)% cm ist Vorgabe
  \psset{xunit=0.5mm,yunit=2mm}
  \psline{->}(40,5)\psline{->}(0,1cm)(40,0)
\end{pspicture}}\\[20pt]
\psset{xunit=0.5mm,yunit=2mm}% ändert die Vorgabe!
\psframebox[framesep=0pt,linestyle=dotted]{%
\begin{pspicture}(40,5)% Vorgabe für x und y unterschiedlich!
  \psline{->}(40,5)\psline{->}(0,1cm)(40,0)
\end{pspicture}}
```

2.3.2 Winkel

```
\degrees                          % identisch mit \degrees[360]
\degrees[Wert für Vollkreis]
\radian                           % identisch mit \degrees[6.28319]
```

Die standardmäßige Vorgabe von 360 Grad für einen Vollkreis kann ungünstig sein, wenn man ein Tortendiagramm erstellen will, bei dem die Angaben in Prozent vorliegen. Für diesen Fall kann man die Winkeleinheit von 360 Grad auf 100 Grad ändern und ohne jegliche Umrechnung der prozentualen Werte die entsprechenden Kreisausschnitte zeichnen. Dies wird unter anderem auch für das Makro \psChart aus dem pstricks-add-Paket ausgenutzt (vergleiche Abschnitt 29.1.2 auf Seite 625). Für die Einstellung des Bogenmaßes existiert die Abkürzung \radian.

[3] Die Einheit cm wurde hier willkürlich gewählt, es hätte auch jede andere zulässige sein können.



02-03-2

```
\usepackage{pstricks}

\begin{pspicture}(-1.2,-1.2)(1.2,1.2)
\degrees[14]
\psforeach{\iA}{1,2,..,14}{%
  \psdot[dotscale=2](1;\iA)%
  \uput{6pt}[\iA](1;\iA){\iA}}
\end{pspicture}
```

2.3.3 Erweiterungen

PSTricks stellt die folgenden Makros bereit, mit denen Längenregister geändert werden können. Der Vorteil gegenüber (LA)TEX ist, dass man sowohl dimensionslose als auch dimensionsbehaftete Werte eingeben kann, was grundsätzlich immer für PSTricks möglich ist. Fehlt die explizite Angabe einer Dimension, so nimmt PSTricks den aktuellen Maßstab, wobei die Einheit cm vordefiniert ist.

```
\pssetlength{Längenregister}{Wert Einheit }
\psaddtolength{Längenregister}{Wert Einheit }
```

2.4 Koordinaten

Koordinaten treten fast ausnahmslos in Paaren (x,y) auf und werden lediglich bei Verwendung der Pakete, die drei Dimensionen unterstützen, zu einem Tripel (x,y,z) erweitert. Ein Koordinatenpaar besteht in der Regel aus zwei Zahlenwerten, deren Einheit sich aus dem aktuellen Maßstab ergibt. Darüber hinaus kann jedoch auch eine explizite Angabe von zulässigen Einheiten erfolgen:

```
\psline(0.1in,1)(3mm,300pt)
```

In diesem Beispiel wird im ersten Koordinatenpaar der x-Wert auf Inch und der y-Wert auf cm bezogen (soweit nichts anderes vereinbart wurde) und das zweite auf die dort angegebenen Einheiten. Diese »normale« Angabe von Koordinaten wird standardmäßig durch den Schalter \SpecialCoor erweitert und durch \NormalCoor zurückgesetzt. Dieser Schalter hat nur noch mehr oder weniger historische Bedeutung, denn durch die Leistungsfähigkeit der heutigen Hardware erscheint der Zeitaufwand beim Prüfen von speziellen Koordinaten mittlerweile als unerheblich. Beispiele zu den hier angegebenen Koordinatenformen findet man im Kapitel 12 auf Seite 155.

```
\SpecialCoor % aktiviert spezielle Koordinatenformen (Vorgabe)
\NormalCoor  % nur (x,y) Zahlenpaare sind zulässig
```

2.5 pspicture–Umgebung

```
\usepackage{pstricks}

In der Regel wird man seine PSTricks-Grafik in einer eigenen Box darstellen und nicht
unbedingt über den laufenden Text schreiben wollen, was grundsätzlich möglich ist und
```

```
jetzt hier rein exemplarisch anhand der gestrichelten Linie gezeigt wird, die einfach
mitten im Text erscheint. \psline[linestyle=dashed,linecolor=red]{|-|}(0,1.5)
```

02-05-1

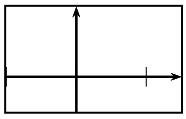
In der Regel wird man seine PSTricks-Grafik in einer eigenen Box darstellen und nicht unbedingt über den laufenden Text schreiben wollen, was grundsätzlich möglich ist und jetzt hier rein exemplarisch anhand der gestrichelten Linie gezeigt wird, die einfach mitten im Text erscheint.

Soll das Überschreiben vermieden werden, so muss der notwendige Platz reserviert werden, wozu PSTricks die pspicture-Umgebung mit folgender Syntax bereitstellt:

```
\pspicture* [Optionen] (xMin,yMin)(xMax,yMax) ... \endpspicture
\pspicture* [Optionen] (xMax,yMax) ... \endpspicture
\begin{pspicture*} [Optionen] (xMin,yMin)(xMax,yMax) ... \end{pspicture*}
\begin{pspicture*} [Optionen] (xMax,yMax) ... \end{pspicture*}
```

```
\usepackage{pstricks,pst-plot}

\raggedright Die reservierte Box ist per Definition mit ihrer unteren Seite auf der
Grundlinie der Zeile angeordnet, was man hier leicht an dieser
\psframebox[framesep=0]{%
  \begin{pspicture}(-1,-0.5)(1.5,1)
    \psaxes[labels=none]{->}(0,0)(-1,-0.5)(1.5,1)
  \end{pspicture}}
erkennen kann, deren \emph{interner} Ursprung jedoch ganz woanders liegen kann,
auch außerhalb der Box. In diesem Beispiel bei \Largrfix{1,0.5}, gemessenen
von der linken unteren Ecke der Box.
```

02-05-2

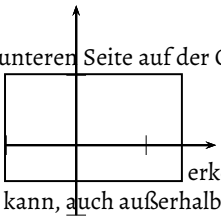
Die reservierte Box ist per Definition mit ihrer unteren Seite auf der Grundlinie der Zeile angeordnet, was man hier leicht an dieser  erkennen kann, deren *interner* Ursprung jedoch ganz woanders liegen kann, auch außerhalb der Box. In diesem Beispiel bei `(1,0.5)`, gemessenen von der linken unteren Ecke der Box.

Der Rahmen und die Achsen sind hier zusätzlich gezeichnet worden, sie haben prinzipiell nichts mit der pspicture-Umgebung zu tun (\psframebox⇒ Abschnitt 10.2.1 auf Seite 127, \psaxes⇒ Abschnitt 14.1 auf Seite 184). Es ist offensichtlich, dass der Einsatz von pspicture in der Regel als eigenständiger Absatz erfolgen wird, denn innerhalb einer Zeile macht sich dies durch einen ungünstigen Zeilenabstand bemerkbar. Eine durch die pspicture-Umgebung festgelegte Box hat per Definition keine Tiefe und ist prinzipiell nichts anderes als ein Platzhalter. Es obliegt einzig und allein dem Anwender, dafür Sorge zu tragen, dass der durch die Koordinaten angeforderte Platz auch die Grafik aufnehmen kann. Wendet man obiges Beispiel auf falsche Koordinatenwerte an, so ist der reservierte Platz entweder zu klein (Grafik liegt partiell über dem Text) oder zu groß (es ensteht

unnötiger weißer Rand). Ersteres ist im Folgenden der Fall, denn der angeforderte Platz der Größe `(-1,-0.5)(1.5,1)` ist zu viel klein für das Koordinatenkreuz `(-1,-1)(2,2)`.

```
\usepackage{pstricks,pst-plot}

\raggedright Die reservierte Box ist per Definition mit ihrer unteren Seite auf der
  Grundlinie der Zeile angeordnet, was man hier leicht an dieser
\psframebox[framesep=0]{%
  \begin{pspicture}(-1,-0.5)(1.5,1)
    \psaxes[labels=none]{->}(0,0)(-1,-1)(2,2)
  \end{pspicture}}
erkennen kann, deren \emph{interner} Ursprung jedoch ganz woanders liegen kann,
auch außerhalb der Box. In diesem Beispiel bei \Largrfix{1,0.5}, gemessenen von der
linken unteren Ecke der Box.
```

02-05-3

Die reservierte Box ist per Definition mit ihrer unteren Seite auf der Grundlinie der Zeile angeordnet, was man hier leicht an dieser  erkennen kann, deren *interner* Ursprung jedoch ganz woanders liegen kann, auch außerhalb der Box. In diesem Beispiel bei `(1,0.5)`, gemessenen von der linken unteren Ecke der Box.

- Wird nur ein Zahlenpaar angegeben, erfolgt automatisch eine Ergänzung zu `(0,0)(xMax,yMax)`. `PSTricks` überprüft nicht, ob die Kombination der einzelnen Werte sinnvoll ist oder nicht.
- Wird keine Längeneinheit angegeben, so bezieht sich die Angabe entweder auf cm oder den letzten mit `\psset{`*[x|y]unit=Wert*`}` gesetzten Wert, wobei für *[x|y]unit* unterschiedliche Werte möglich sind.
- Die Koordinaten geben nur den für TEX sichtbaren reservierten Platz an; sie haben sonst keine Auswirkung auf die Ausgabe, die auch hier außerhalb dieses Platzes liegen kann.
- Die Sternversion löscht alles außerhalb der durch die Koordinaten vorgegebenen Grenzen (Clipping) und benutzt dazu `\pstVerb` und `\pstverbscale` (siehe auch Abschnitt 13.2.5 auf Seite 166)

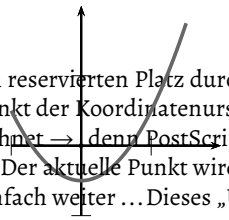
Insbesondere der letzte Fall läßt sich vorteilhaft bei der Berechnung und Darstellung mathematischer Funktionen anwenden, da man in diesen Fällen dann nicht genaue Kenntnis über die maximal zu erwartenden Werte haben muss.

Die Koordinaten der `pspicture`-Umgebung haben für TEX und PS eine unterschiedliche Bedeutung; für TEX legen sie nur die Größe der Box fest, wobei die linke, untere Ecke der Box auf der Basislinie liegt. Für PS kennzeichnen die Koordinaten, wo der Koordinatenursprung liegt, ausgehend vom linken, unteren Eckpunkt. Dabei gilt es zu beachten, dass es keine »Kommunikation« zwischen TEX und PS gibt. Der Anwender muss dafür Sorge tragen, dass PS sämtliche Informationen bekommt, um die Grafik im Sinne des Anwenders zu erstellen.

PS orientiert sich ausnahmslos am aktuellen Punkt, der gleichzeitig dem Koordinatenursprung entspricht, wenn man auf die pspicture-Umgebung verzichtet. Daher bleibt im folgenden Beispiel auch das Makro \psframebox ohne Wirkung, denn es bezieht sich auf die TEX-Box, die ohne eine Platzreservierung sowohl in der Breite als auch der Höhe gleich Null ist.

```
\usepackage{pstricks,pst-plot}

\raggedright Ohne einen reservierten Platz durch eine entsprechende \TeX-Box ist
per Definition der aktuelle Punkt der Koordinatenursprung für PostScript und
es wird komplett über den Text gezeichnet $\rightarrow$
\psframebox[framesep=0]{\psaxes[labels=none]{->}(0,0)(-1,-1)(2,2)%
\psplot[linewidth=1.5pt,linecolor=black!60]{-1}{1.5}{x dup mul 0.5 sub}}
denn PostScript weiß nichts von \TeX, ob dort Platz reserviert wurde
oder nicht. Der aktuelle Punkt wird daher auch von PostScript nicht
verändert; \TeX\ schreibt einfach weiter \ldots Dieses "`Übermalen"'
kann man aber für sogenannte Overlays sinnvoll nutzen.
```

02-05-4



Ohne einen reservierten Platz durch eine entsprechende TEX-Box ist per Definition der aktuelle Punkt der Koordinatenursprung für PostScript und es wird komplett über den Text gezeichnet → denn PostScript weiß nichts von TEX, ob dort Platz reserviert wurde oder nicht. Der aktuelle Punkt wird daher auch von PostScript nicht verändert; TEX schreibt einfach weiter … Dieses „Übermalen“ kann man aber für sogenannte Overlays sinnvoll nutzen.

2.5.1 Optionen

Tabelle 2.11 zeigt die beiden Optionen, die für die pspicture-Umgebung zur Verfügung stehen.

Name	*Bedeutung*	*Voreinstellung*
showgrid	zeichne Koordinatengitter	false
shift	absolute vertikale Verschiebung	0pt

Tabelle 2.11: Spezielle Optionen der Umgebung pspicture.

Beide Optionen stehen nur für PSTricks-Versionen ab 1.12 (13.10.2005) zur Verfügung. Ein optionales Argument ausschließlich für die vertikale Verschiebung war dagegen schon immer möglich, allerdings mit anderer Syntax und auch anderer Auswirkung, was häufig zu Missverständnissen führte. Mit der Änderung wird nun die allgemein übliche Syntax der optionalen Parameter angewendet.

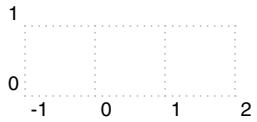
Syntax

showgrid

Bei grafischen Abbildungen wird sehr häufig ein Koordinatengitter gezeichnet, was über das Makro \psgrid möglich ist (Abschnitt 3.1). Intern ist dafür in pstricks bereits ein Stil gridstyle definiert:

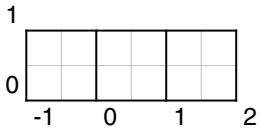
`\newpsstyle{`*gridstyle*`}{`*subgriddiv=0,gridcolor=lightgray,griddots=10*`}`

Eine Beschreibung des Makros `\newpsstyle` findet man im Abschnitt 11.1 und für die einzelnen Optionen in Abschnitt 3.1. Diese Vorgabe lässt sich jederzeit überschreiben, sodass man das Aussehen des Koordinatengitters an eigene Wünsche anpassen kann.



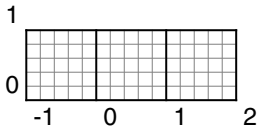
02-05-5

```
\usepackage{pstricks}

\begin{pspicture}[showgrid=true](-1,0)(2,1)
\end{pspicture}
```



02-05-6

```
\usepackage{pstricks}

\newpsstyle{gridstyle}{%
  subgriddiv=2,subgridcolor=lightgray}
\begin{pspicture}[showgrid=true](-1,0)(2,1)
\end{pspicture}
```



02-05-7

```
\usepackage{pstricks}

\newpsstyle{gridstyle}{}
\begin{pspicture}[showgrid=true](-1,0)(2,1)
\end{pspicture}
```

02-05-8

```
\usepackage{pstricks}

\begin{pspicture}(-1,0)(2,1)
\end{pspicture}% EPS ohne Bounding Box
```

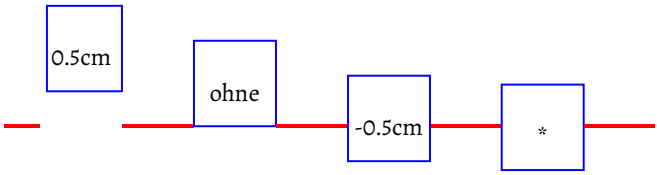
Obiges Beispiel mit der leeren Ausgabe führt in der Regel zu einer falschen Bounding-Box, da bei der Anwendung von DVIPS oder anderen Programmen grundsätzlich falsche Werte ermittelt werden; die Programme haben aufgrund der »nicht vorhandenen« Abbildungsgröße keine Anhaltspunkte für die Bestimmung.

`shift`

Die Option `shift` bezeichnet den vertikalen Offset bezogen auf die Text-Basislinie (baseline), die auch der unteren Linie der gesamten `pspicture`-Umgebung entspricht. Der voreingestellte Wert ist 0 pt und −0.5 cm würde die Box vertikal um 0.5 cm nach unten verschieben, was sich leicht am angegebenen Beispiel nachvollziehen lässt. Absolut gesehen bleibt die Basislinie natürlich auch nicht auf derselben Höhe, denn die Zeilenhöhe muss innerhalb der Seite ebenfalls angepasst werden. Die Option entspricht prinzipiell dem LaTeX-Makro `\raisebox`, was auch in dem letzten Beispiel dieses Abschnitts zu sehen ist. `shift` ist nur für die äußerste `pspicture`-Umgebung relevant, da geschachtelte Umgebungen bereits durch den `\rput`-Befehl beliebig verschoben werden können.

```
\usepackage{pstricks}

\textcolor{red}{\rule{5mm}{1pt}}%
\begin{pspicture}[shift=0.5cm](-0.6,-0.5)(0.6,0.75)
  \psframe[linecolor=blue](-0.5,-0.5)(0.6,0.75)\rput(0,0){0.5cm}
\end{pspicture}\textcolor{red}{\rule{10mm}{1pt}}%
\begin{pspicture}(-0.6,-0.5)(0.6,0.75)
  \psframe[linecolor=blue](-0.6,-0.5)(0.6,0.75)\rput(0,0){ohne}
\end{pspicture}\textcolor{red}{\rule{10mm}{1pt}}%
\begin{pspicture}[shift=-0.5cm](-0.6,-0.5)(0.6,0.75)
  \psframe[linecolor=blue](-0.6,-0.5)(0.6,0.75)\rput(0,0){-0.5cm}
\end{pspicture}\textcolor{red}{\rule{10mm}{1pt}}%
\begin{pspicture}[shift=*](-0.6,-0.5)(0.6,0.75)
  \psframe[linecolor=blue](-0.6,-0.5)(0.6,0.75)\rput(0,0){*}
\end{pspicture}\textcolor{red}{\rule{10mm}{1pt}}
```

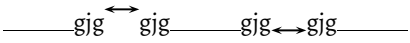
02-05-9



Mit der Angabe von `shift=*` erfolgt eine vertikale Zentrierung der `pspicture`-Umgebung, was einer vertikalen Verschiebung nach oben oder unten um die Hälfte der Boxhöhe entspricht. Mit dieser Option kann man das alte Verhalten von `PSTricks` erreichen. Um eine `PSTricks`-Box mit ihrer internen Nulllinie auf die gleiche Höhe wie die Baseline zu bekommen, wird einfach eine Verschiebung nach unten um den ersten y-Wert der Boxgröße vorgenommen. Dabei kann das Makro `\psyunit` gute Dienste leisten, da man sich dann nicht um die aktuell gültige Einheit kümmern muss. Der erste Pfeil liegt 0,3 Einheiten über der Basislinie, der zweite aufgrund der Option `[shift=-0.3\psyunit]` auf dieser. Alternativ hätte man auch `shift=*` benutzen können.

```
\usepackage{pstricks}

\rule{1cm}{.1pt}gjg\pspicture(-.25,-.3)(.25,.25) \psline{<->}(-.25,0)(.25,0)
\endpspicture gjg\rule{1cm}{.1pt}gjg%
\pspicture[shift=-0.3\psyunit](-.25,-.3)(.25,.25)\psline{<->}(-.25,0)(.25,0)
\endpspicture gjg\rule{1cm}{.1pt}
```

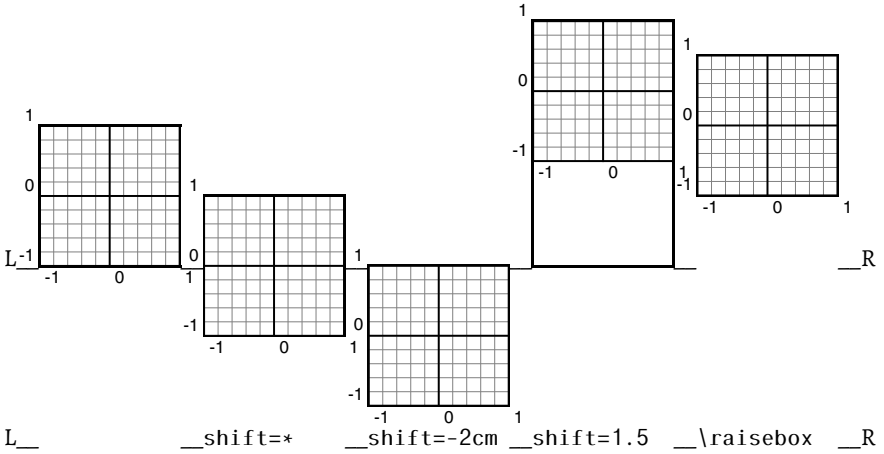
02-05-10



Zu beachten ist, dass bei Verwendung von `\fbox` die Basislinie immer erhalten bleibt, sodass der Rahmen im Gegensatz zu einer Anwendung mit `\raisebox` unerwünschte Maße annimmt (vergleiche Beispiel 02-05-11 auf der nächsten Seite). Wird für die `shift`-Option keine Einheit angegeben, so wird die aktuell gültige genommen, was in der Regel cm sind. Im folgenden Beispiel ist diese Tatsache deutlich bei den rechten beiden

Rechtecken zu sehen. Soll eine pspicture durch Anwendung von \fbox mit einem Rahmen versehen werden, so sollte man bei einer notwendigen Verschiebung anstelle der Option shift besser \raisebox anwenden. Dann bekommt die Box eine »positive« Tiefe, wodurch der Rahmen dann ebenfalls mit verschoben wird, was ganz rechts im folgenden Beispiel deutlich zu sehen ist.

```
\usepackage{pstricks,array}

\psset{gridlabels=7pt}% Standardwert
L\_\_\fbox{\begin{pspicture}(-1,-1)(1,1) \psgrid \end{pspicture}}\_\_%
\fbox{\begin{pspicture}[shift=*](-1,-1)(1,1) \psgrid \end{pspicture}}\_\_%
\fbox{\begin{pspicture}[shift=-2cm](-1,-1)(1,1) \psgrid \end{pspicture}}\_\_%
\fbox{\begin{pspicture}[shift=1.5](-1,-1)(1,1) \psgrid \end{pspicture}}\_\_%
\raisebox{0.5\height}{\fbox{\begin{pspicture}(-1,-1)(1,1) \psgrid
\end{pspicture}}}\_\_R  \par\medskip
\begin{tabular}{@{L\_\_}
  *5{>{\ttfamily}p{\dimexpr2cm+2\fboxsep+2\fboxrule}@{\_\_}}@{R}}
 &  shift=* & shift=-2cm & shift=1.5 & \textbackslash raisebox
\end{tabular}
```



02-05-11

2.6 Leerraum (»Whitespace«)

Innerhalb einer pspicture-Umgebung wird jeder Leerraum zwischen PSTricks-Objekten entfernt. Außerhalb dieser Umgebung wird jedes Objekt wie ein einzelnes Zeichen behandelt, sodass zusätzlicher Leerraum nicht entfernt wird. Dies kann manchmal unerwünscht sein, beispielsweise innerhalb einer LaTeX-Umgebung picture. In solchen Fällen kann mit \KillGlue und \DontKillGlue Leerraum ignoriert oder berücksichtigt werden (siehe dazu Abschnitt 13.2.2 auf Seite 165).

Kapitel 3

Koordinatensystem

Grundsätzlich liegt durch PS bedingt ein kartesisches Koordinatensystem zugrunde. Dabei kann man den Ursprung, der durch die Wahl der `pspicture`-Umgebung festgelegt wurde, für jede Anwendung beliebig lokal verschieben. Weiterhin lassen sich auch beide Achsen vertauschen, was beim Plotten von Funktionen von Interesse ist, wenn das Berechnen der Umkehrfunktion leichter möglich ist (siehe Beispiel auf der nächsten Seite). Tabelle 3.1 zeigt die Syntax dieser Optionen, welche grundsätzlich bei allen Makros anwendbar sind, sich jedoch ausschließlich auf das zugrundeliegende Koordinatensystem beziehen.

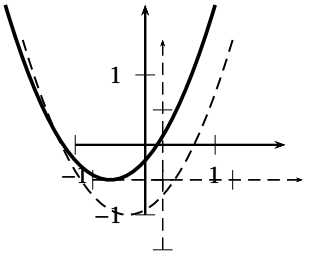
Name	*Werte*	*Vorgabe*
`origin`	{*xWert Einheit* , *yWert Einheit* }	{*0pt,0pt*}
`swapaxes`	*Boolean*	`false`

Tabelle 3.1: Parameter für das Koordinatensystem.

Erst ab einer `PSTricks`-Version > 1.12 wird die `origin`-Option in der hier angegebenen Weise angewendet. Bei vorhergehenden Versionen erfolgte keine korrekte Zuordnung der Parameterwerte. Weiterhin wirkt diese Option nur bei Makros, die als `PSTricks`-Objekte[1] definiert wurden, beispielsweise `\psdot`. Im folgenden Beispiel ist dies auch daran zu erkennen, dass die Koordinatenbeschriftung nicht mit verschoben wird; diese erfolgt auf TEX-Ebene und ist schon deswegen von der Verschiebung ausgenommen!

[1] Unter einem `PSTricks`-Objekt wird jedes Makro verstanden, welches intern über das Makro `\pst@object` definiert wird.

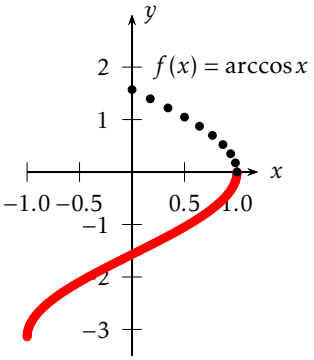
Das folgende Beispiel zeichnet zum einen die durch die Koordinaten der pspicture-Umgebung festgelegten Koordinatenachsen und eine Parabel $y = (x+0.5)^2 - 0.5$ (jeweils durchgezogene Linien) und zum anderen beide Makros noch einmal mit der zusätzlichen Option `origin={0.25,-0.5}`, aber ohne Änderung der anderen Werte (jeweils gestrichelte Linien). Bezogen auf das Hauptkoordinatensystem verschiebt sich der Ursprung des gestrichelten Systems von $(0;0)$ nach $(0.25;-0.5)$.



03-00-1

```
\usepackage{pst-plot}

\begin{pspicture}(-1,-1)(2,2)
  \psaxes{->}(0,0)(-1,-1)(2,2)
  \psparabola[linewidth=1.5pt](1,2)(-0.5,-0.5)
  \psset{linestyle=dashed}
  \psparabola[origin={0.25,-0.5}](1,2)(-0.5,-0.5)
  \psaxes[origin={0.25,-0.5},
     linewidth=0.2pt]{->}(0,0)(-1,-1)(2,2)
\end{pspicture}
```

Prinzipielle Bedeutung kommt hier nur der Option `swapaxes` zu, denn PS verfügt lediglich über eine trigonometrische Umkehrfunktion, den Arcus Tangens (atan). Alle anderen Umkehrfunktionen müssen über den Tangens definiert werden. Im folgenden Beispiel wird dagegen die `swapaxes`-Option benutzt, um auf einfache Weise die Funktion $y = \arccos x$ darzustellen, indem einfach die Kosinus-Funktion mit vertauschten Achsen gezeichnet wird. Dadurch braucht man keine mathematischen Umrechnungen vorzunehmen; es ist lediglich auf das vorzugebende Intervall zu achten. Das relativ neue `PSTricks`-Paket `pst-math`[2] bietet Unterstützung für mathematische Funktionen, für die in PS keine direkte Entsprechung existiert. [38]



03-00-2

```
\usepackage{pst-plot}

\psset{xunit=1.5,yunit=0.75,plotpoints=200,
  plotstyle=dots}
\begin{pspicture}(-1.1,-3.5)(1.2,3.1)
  \psaxes[Dx=0.5]{->}(0,0)(-1,-3.5)(1.2,3)%
    [$x$,0][$y$,0]
  \rput[l](0.2,2){$f(x)=\arccos x$}
  \psset{yunit=1.5cm,xunit=.75cm,swapaxes=true}
  \psplot[linecolor=red]{-3.1415}{0}%
    {x RadtoDeg cos}
  \psplot[plotstyle=dots,plotpoints=10]%
      {0}{1.5707}{x RadtoDeg cos}
\end{pspicture}
```

3.1 Grids

`PSTricks` bietet eine Vielzahl an Möglichkeiten, um verschiedenste kartesische Koordinatenraster zu erstellen. Daraus folgt dann aber gleich eine relativ große Zahl an

[2] Siehe auch Abschnitt 27.1 auf Seite 559.

Parametern, die man entsprechend einstellen kann. Weitere Möglichkeiten enthält das Paket `pst-plot`, insbesondere was logarithmische Achseneinteilungen und dezimale Beschriftungen betrifft. Das Erstellen der Gitter kann automatisiert werden, falls alle Abbildungen bezüglich des Gitters identisch sind. Die Funktionsweise der Option `showgrid` wurde bereits in Abschnitt 2.5.1 auf Seite 29 erläutert. Man erhält damit dasselbe Verhalten, wenn die im Folgenden gezeigten Optionen jeweils als neuer Gitterstil definiert werden; `\psset` wird faktisch nur durch `\newpsstyle{`*gridstyle*`}` ersetzt.

3.2 Parameter

Tabelle 3.2 zeigt sämtliche Parameter, die in direktem Zusammenhang mit `\psgrid` stehen. In den folgenden Beispielen wird immer von der Grundeinstellung aller Parameter

Name	*Werte*	*Vorgabe*
`gridwidth`	*Wert Einheit*	`0.8pt`
`gridcolor`	*Farbe*	`black`
`griddots`	*Wert*	`0`
`gridlabels`	*Wert Einheit*	`10pt`
`gridlabelcolor`	*Farbe*	`black`
`subgriddiv`	*Wert*	`5`
`subgridwidth`	*Wert Einheit*	`0.4pt`
`subgridcolor`	*Farbe*	`gray`
`subgriddots`	*Wert*	`0`

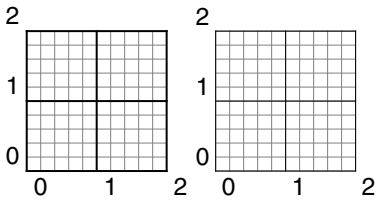
Tabelle 3.2: Zusammenfassung aller Parameter für `\psgrid`.

ausgegangen, weshalb auch die Label in der Regel optisch zu groß erscheinen, da aus Platzgründen hier nur kleine Gitter verwendet werden.

3.2.1 gridwidth

`gridwidth` bestimmt die Dicke der Haupt-Gitterlinien und sollte eher zu klein als zu groß gewählt werden, dabei jedoch dicker als die Unterlinien (`subgridwidth`).

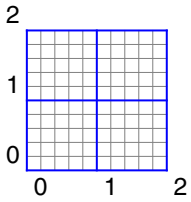
03-02-1




```
\usepackage{pstricks}

\begin{pspicture}(2,2)
  \psgrid% Standarddicke ist 0.8pt
\end{pspicture}\qquad
\begin{pspicture}(2,2)
  \psgrid[gridwidth=0.1pt]
\end{pspicture}
```

3.2.2 gridcolor

`gridcolor` bestimmt die Farbe der Haupt-Gitterlinien und kann zur Hervorhebung benutzt werden.

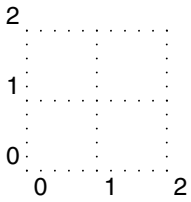



```
\usepackage{pstricks}

\begin{pspicture}(2,2)
   \psgrid[gridcolor=blue]
\end{pspicture}
```

03-02-2

3.2.3 griddots

griddots bestimmt die Zahl der Punkte (dots) pro Einheit, wenn statt der durchgehenden Linie eine gepunktete Haupt-Gitterlinie gezeichnet werden soll. Dies ist insbesondere dann von Interesse, wenn das Gitter an sich etwas mehr in den Hintergrund treten soll. Zu beachten ist, dass die Punkte nur zu sehen sind, wenn subgriddiv auf Null oder Eins gesetzt wird, ansonsten würden die Punkte durch die Linien des subgrid-Netzes überdeckt werden.

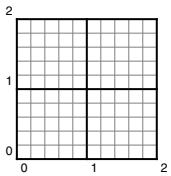



```
\usepackage{pstricks}

\begin{pspicture}(2,2)
   \psgrid[griddots=5,subgriddiv=0]
\end{pspicture}
```

03-02-3

3.2.4 gridlabels

gridlabels bestimmt die Schriftgröße der Label.




```
\usepackage{pstricks}

\begin{pspicture}(2,2)
   \psgrid[gridlabels=5pt]
\end{pspicture}
```

03-02-4

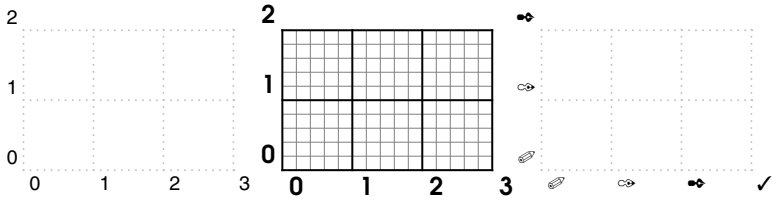
3.2.5 gridfont

Im Gegensatz zu den Labels von \psaxes, die auf TEX-Ebene geschrieben werden, erfolgt dies für das Koordinatengitter auf PS-Ebene. Dies hat zur Folge, dass nur die PS-Fonts für die Label benutzt werden können. Standardmäßig ist dies die Schrift Helvetica, welche über die Option gridfont geändert werden kann, wobei normalerweise nur die folgenden PS-Fonts möglich sind:

Helvetica (Vorgabe) – Helvetica-Narrow – Times-Roman – Courier – AvantGard-Demi – NewCenturySchlbk – Palatino-Roman – Bookman-Demi – Dingbats – Symbol

```
\usepackage{pstricks}

\begin{pspicture}[showgrid=true](3,2)\end{pspicture}\qquad
\begin{pspicture}(3,2)\psgrid[gridfont=AvantGard-Demi]\end{pspicture}\qquad
\begin{pspicture}(3,2)\psgrid[style=gridstyle,gridfont=Dingbats]\end{pspicture}
```

03-02-5

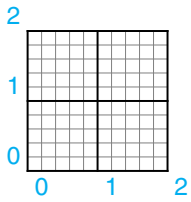


Zu beachten ist, dass die Namen der PostScript-Fonts je nach TEX-Installation abweichen können, beispielsweise `ZapfDingbats` statt `Dingbats`.

3.2.6 gridlabelcolor

`gridlabelcolor` bestimmt die Schriftfarbe der Label.

03-02-6

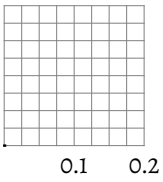



```
\usepackage{pstricks,pst-plot}

\begin{pspicture}(2,2)
    \psgrid[gridlabelcolor=cyan]
\end{pspicture}
```

3.2.7 subgriddiv

`subgriddiv` bestimmt die Zahl der Unterteilungen zwischen zwei ganzen Zahlen. Für große Maßstäbe kann es angebracht sein, weitaus mehr Unterteilungen zuzulassen. Die Angaben `subgriddiv=0` und `subgriddiv=1` sind identisch und führen beide dazu, dass keine weiteren Linien gezeichnet werden.

03-02-7




```
\usepackage{pstricks}

\psset{unit=10}
\begin{pspicture}(0.2,0.2)
  \psgrid[gridlabels=0pt,subgriddiv=40]
  \uput[-90](0.1,0){0.1}
  \uput[-90](0.2,0){0.2}
\end{pspicture}
```

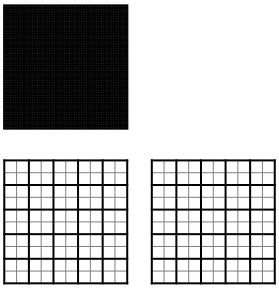
Für die Berechnung der Unterteilungen wird auf die Werte von `xunit` und `yunit` zurückgegriffen. Hierbei kann es zu massiven Problemen kommen, wenn man eine sehr kleine Einheit definiert hat und dann `\psgrid` mit absoluten Koordinaten aufruft, beispielsweise `\psgrid(`*10cm,10cm*`)`. Bei einem Maßstab von `1pt` kommt es hier zu Schwierigkeiten,

denn 1 pt bestimmt den Abstand einer Hauptteilung, sodass es davon in diesem Beispiel mit `\psgrid(`*10cm,10cm*`)` ungefähr 280 gibt und zusätzlich ungefähr 1400 Unterteilungen. Je nach Version von `PSTricks` ist die maximale Zahl an Unterteilungen begrenzt, in der derzeitigen Version auf 500. Das Problem kann behoben werden, indem man lokal auf einen anderen Maßstab umschaltet: `\psgrid[`*unit=1cm*`](10cm,10cm)`.

Maßstab beachten

`unit` lokal ändern

Im folgenden Beispiel 03-02-8 werden beide Fälle gegenübergestellt. Die Festlegung der Grundeinheit durch `unit=1pt` ist ein durchaus typischer Fall, wenn es darum geht, gegebene Grafiken zu beschriften, bei denen Angaben bereits in pt vorliegen. Durch `\begin{pspicture}(50,50)` wird daher eine Box der Größe 50 pt×50 pt reserviert. Erstellt man jetzt für diese ein Koordinatengitter mit einer zusätzlichen Unterteilung von `subgriddiv=2`, so ergibt sich faktisch ein schwarzes Feld. Pro Haupteinheit von 1 pt≈0,35 mm werden insgesamt drei Linien gezeichnet, was zu dem beschriebenen Effekt führt. Schaltet man jetzt *lokal* auf eine andere Einheit um, indem `\psgrid` über das optionale Argument `unit=10pt` vorgegeben wird, so beträgt die Haupteinheit jetzt 10 pt≈3,5 mm. Die Koordinaten müssen dann ebenfalls entsprechend korrigiert werden, indem sie nur noch $\frac{1}{10}$ der ursprünglichen Werte betragen oder alternativ mit der Einheit pt angegeben werden.

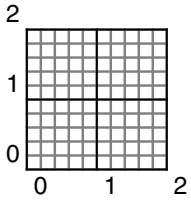


```
\usepackage{pstricks}

\psset{unit=1pt}%     Sehr kleine Grundeinheit
\begin{pspicture}(50,50)% 50 Haupteinteilungen
  \psgrid[gridlabels=0pt,subgriddiv=2]
\end{pspicture}\par\bigskip
\begin{pspicture}(50,50)
  \psgrid[unit=10pt,% andere Einheit für psgrid
    gridlabels=0pt,subgriddiv=2](5,5)
\end{pspicture}\quad
\begin{pspicture}(50,50)
  \psgrid[unit=10pt,% andere Einheit für psgrid
    gridlabels=0pt,subgriddiv=2](50pt,50pt)
\end{pspicture}
```

03-02-8

3.2.8 `subgridwidth`

`subgridwidth` bestimmt die Dicke der Unter-Gitterlinien und sollte eher zu klein als zu groß gewählt werden und vor allem kleiner als die übergeordneten Hauptgitterlinien.



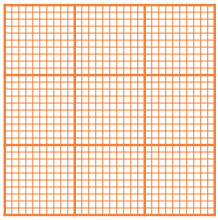

```
\usepackage{pstricks}

\begin{pspicture}(2,2)
   \psgrid[subgridwidth=1pt]
\end{pspicture}
```

03-02-9

3.2.9 subgridcolor

subgridcolor bestimmt die Farbe der Unter-Gitterlinien und kann zur Hervorhebung benutzt werden. Diese sind in dem folgenden Beispiel lediglich als Graustufe wahrnehmbar.

03-02-10

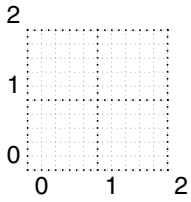


```
\usepackage{pstricks}

\definecolor{orange}{cmyk}{0,0.61,0.87,0}
\begin{pspicture}(3,3)
  \psgrid[subgriddiv=10,gridlabels=0,
     gridwidth=1pt,gridcolor=orange,
     subgridwidth=0.1pt,subgridcolor=orange]
\end{pspicture}
```

3.2.10 subgriddots

subgriddots bestimmt die Zahl der Punkte (dots) pro Zwischeneinheit, wenn statt der durchgehenden Linie eine gepunktete Sub-Gitterlinie gezeichnet werden soll. Dies ist insbesondere dann von Interesse, wenn das Gitter an sich etwas mehr in den Hintergrund treten soll.

03-02-11




```
\usepackage{pstricks}

\begin{pspicture}(2,2)
   \psgrid[griddots=10,subgriddots=3]
\end{pspicture}
```

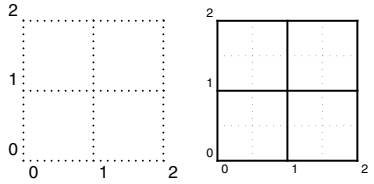
3.3 Makros

Es existiert nur ein einziges Makro zum Zeichnen eines Gitternetzes. Aufgrund der zahlreichen Optionen ist eine Definition eines eigenen Makros sinnvoll. Dies kann mit dem standardmäßigen \newcommand geschehen

```
\newcommand*\myGrid{\psgrid[subgriddiv=0,griddots=10,gridlabels=7pt]}
```

oder alternativ über das PSTricks-Makro \newpsobject (siehe auch Abschnitt 11.2 auf Seite 138). Beides wird im folgenden Beispiel gezeigt.

03-03-1




```
\usepackage{pstricks}

\newcommand*\Grid{\psgrid[subgriddiv=0,
    griddots=10,gridlabels=7pt]}
\newpsobject{psGrid}{psgrid}{subgriddots=5,
    subgriddiv=2,gridlabels=5pt}
\begin{pspicture}(0,-0.2)(2,2)\Grid  \end{pspicture}
\qquad
\begin{pspicture}(0,-0.2)(2,2)\psGrid\end{pspicture}
```

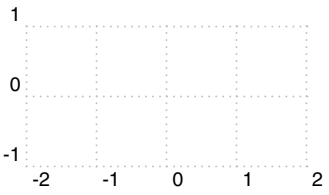
Alternativ kann auch der intern vorgegebene Stil `gridstyle` überschrieben werden und dann über die `showgrid`-Option der `pspicture`-Umgebung automatisch für das Zeichnen eines Gitternetzes verwendet werden (siehe Abschnitt 2.5.1 auf Seite 29).

Das `\psgrid`-Makro ist ein mächtiges Werkzeug zum Zeichnen von Koordinatengittern. Die Syntax ist dagegen sehr einfach:

```
\psgrid [Optionen]
\psgrid [Optionen] (x,y)
\psgrid [Optionen] (x1,y1)(x2,y2)
\psgrid [Optionen] (x0,y0)(x1,y1)(x2,y2)
```

Per Definition wird von einem kartesischen Koordinatensystem ausgegangen.

- Ohne jegliche Angabe eines Punktes nimmt `\psgrid` die durch die `pspicture` Umgebung festgelegten Koordinaten. Existieren diese nicht, weil `\psgrid` außerhalb einer `pspicture`-Umgebung benutzt wird, dann wird ein 10×10 Koordinatengitter in dem aktuell gültigen Maßstab angenommen. Wird nur ein Koordinatenpaar angegeben, wird automatisch `(0,0)` als Ursprung festgelegt. Die Angabe von zwei Koordinatenpaaren legt die linke untere und die rechte obere Ecke des Koordinatengitters fest. Werden drei Koordinatenpaare angegeben, so bezeichnet das erste Paar den Schnittpunkt der Koordinatenachsen und die anderen beiden wieder die linke untere und die rechte obere Ecke des Koordinatengitters.
- Die Label werden in der Regel für die horizontale Achse rechts unter und für die vertikale Achse links über den jeweiligen Punkt gesetzt. Wird die Reihenfolge der Koordinaten vertauscht, so wechselt die Beschriftung für die horizontalen Werte auf links darüber und für die vertikalen auf rechts darunter.
- Das Makro `\psgrid` arbeitet nur mit ganzzahligen Werten für die Koordinaten, sodass ohne weitere Manipulation über die Maßstabsfaktoren das Ergebnis nicht immer den Erwartungen entspricht. Im folgenden Beispiel wird der Platz für eine Box von der Breite 3 cm und der Höhe 2,5 cm reserviert. `\psgrid` benötigt jedoch aufgrund der Rundungen eine Breite von 4 cm und eine Höhe von 2 cm. Dies ist unbedingt zu beachten, um Überschneidungen mit dem Textbereich zu vermeiden.




```
\usepackage{pstricks}

\begin{pspicture}[showgrid=true]%
    (-1.5,-1.25)(1.5,1.25)
\end{pspicture}
```

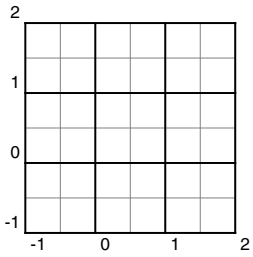
03-03-2

Die folgende Serie von Beispielen zeigt die verschiedenen Möglichkeiten von `\psgrid`. Die ständige Wiederholung von

```
\psset{gridlabels=7pt,subgriddiv=2}
```

hat hier nur erklärenden Charakter, kann in einem realen Dokument natürlich global erfolgen und damit für alle Koordinatensysteme gültig sein.

03-03-3

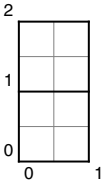



```
\usepackage{pstricks}

\psset{gridlabels=7pt,subgriddiv=2}
\begin{pspicture}(-1,-1)(2,2)
   \psgrid
\end{pspicture}
```

03-03-4

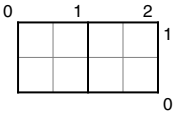

```
\usepackage{pstricks}

\psset{gridlabels=7pt,subgriddiv=2}
\begin{pspicture}(-1,-1)(2,2)
   \psgrid(2,1)
\end{pspicture}
```

03-03-5



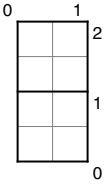

```
\usepackage{pstricks}

\psset{gridlabels=7pt,subgriddiv=2}
\begin{pspicture}(-1,-.25)(2,2)
   \psgrid(1,2)
\end{pspicture}
```

03-03-6



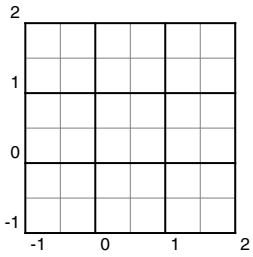

```
\usepackage{pstricks}

\psset{gridlabels=7pt,subgriddiv=2}
\begin{pspicture}(-1,-.25)(2,1.25)
   \psgrid(2,1)(0,0)
\end{pspicture}
```

03-03-7

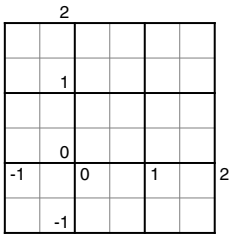



```
\usepackage{pstricks}

\psset{gridlabels=7pt,subgriddiv=2}
\begin{pspicture}(-1,-.25)(2,2)
   \psgrid(1,2)(0,0)
\end{pspicture}
```

03-03-8

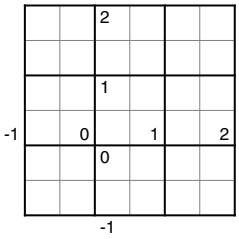



```
\usepackage{pstricks}

\psset{gridlabels=7pt,subgriddiv=2}
\begin{pspicture}(-1,-1)(2,2)
   \psgrid(-1,-1)(2,2)
\end{pspicture}
```




```
\usepackage{pstricks}

\psset{gridlabels=7pt,subgriddiv=2}
\begin{pspicture}(-1,-1)(2,2)
   \psgrid(0,0)(-1,-1)(2,2)
\end{pspicture}
```

03-03-9

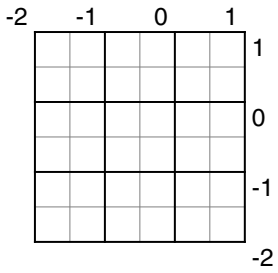



```
\usepackage{pstricks}

\psset{gridlabels=7pt,subgriddiv=2}
\begin{pspicture}(-1,-1)(2,2)
   \psgrid(0,0)(2,2)(-1,-1)
\end{pspicture}
```

03-03-10

3.4 Spezialfälle

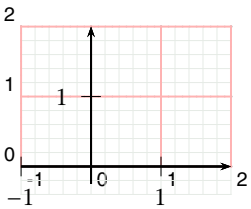
PSTricks erlaubt neben positiven Einheiten auch negative, was dann zu einem diagonal verschobenen Koordinatenursprung mit negativen Achsen führt:




```
\usepackage{pstricks}

\psset{subgriddiv=2,unit=-1cm}
\begin{pspicture}(-1,-1)(2,2)
   \psgrid
\end{pspicture}
```

03-04-1

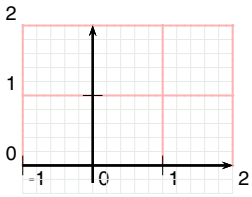
Eine Anwendung von \psgrid zusammen mit \psaxes (siehe Abschnitt 14.1 auf Seite 184) ergibt eine doppelte Bezeichnung der Koordinaten, die in der Regel unerwünscht ist. Bei einem der beiden Makros ist dann über eine entsprechende Option die Ausgabe zu unterbinden.




```
\usepackage{pstricks,pst-plot}

\begin{pspicture}[showgrid=true](-1,-0.5)(2,2)
  \psaxes{->}(0,0)(-1,-0.25)(2,2)
\end{pspicture}
```

03-04-2

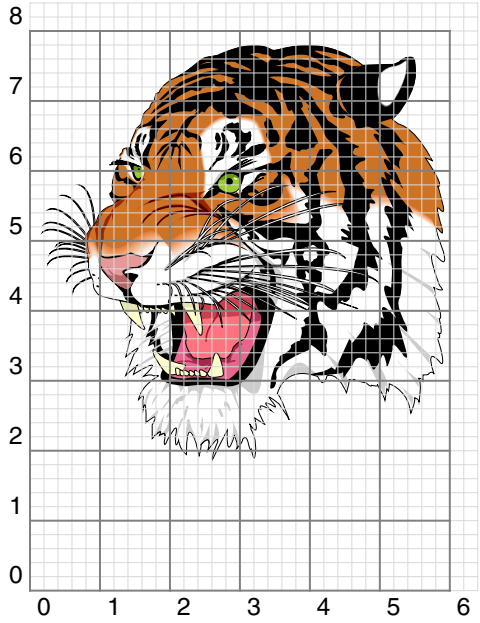
03-04-3




```
\usepackage{pstricks,pst-plot}

\begin{pspicture}[showgrid=true](-1,-0.5)(2,2)
  \psaxes[labels=none]{->}(0,0)(-1,-0.25)(2,2)
\end{pspicture}
```

Besonders zum Überschreiben von existierenden Grafiken[3] beziehungsweise Erzeugen von sogenannten Overlays, kann ein Koordinatengitter hilfreich sein, um gezielter bestimmte Positionen in der Abbildung durch Koordinaten festlegen zu können. Hier wird nur die Erzeugung der Gitterüberlagerung sowie die Anwendung unterschiedlicher Maßstäbe für `pspicture` und `\psgrid` gezeigt. Ohne die genauen Maße der skalierten Abbildung zu kennen, kann ein Koordinatengitter im cm-Maßstab über die Abbildung gelegt werden.

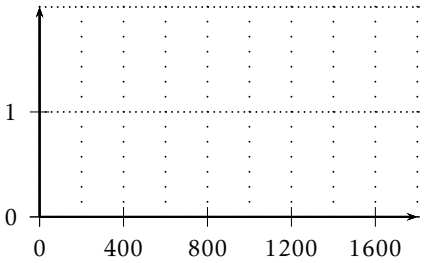
03-04-4




```
\usepackage{pstricks,graphicx}
\newsavebox\IBox

\savebox\IBox{%
  \includegraphics[scale=0.3]{%
    figures/tiger}}
\psset{unit=1pt}
\begin{pspicture}(\wd\IBox,\ht\IBox)
  \rput[lb](0,0){\usebox\IBox}
  \psgrid[unit=1cm,subgriddiv=5,
    griddots=0,
    subgridwidth=0.1pt,
    subgridcolor=black!15,
    gridcolor=black!50]%
     (\wd\IBox,\ht\IBox)
\end{pspicture}
```

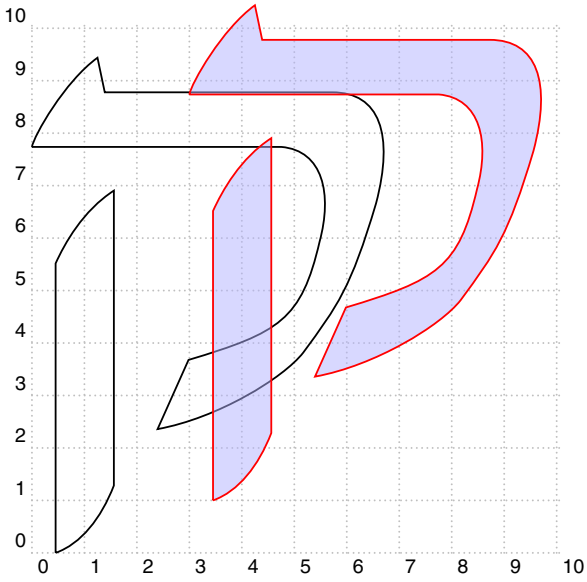
Unterschiedliche Einheiten für die Achsen sind möglich:

03-04-5




```
\usepackage{pst-plot}

\psset{xunit=0.03mm,yunit=15mm}
\begin{pspicture}(-0.25,-0.25)(1950,2)
  \psgrid[subgriddiv=0,
        griddots=7,
        gridlabels=0pt,
        xunit=200](9,2)
  \psaxes[Dx=400]{->}(1800,2)
\end{pspicture}
```

[3] Die Grafik `tiger.eps` steht allgemein auf CTAN zur Verfügung.



03-04-6

Kapitel 4

Linien und Polygone

Linien stellen einen Schwerpunkt einer jeden grafischen Software dar und haben auch in PSTricks eine große Bedeutung. Entsprechend umfangreich ist auch die Zahl der möglichen Parameter, die alle in Tabelle 4.1 zusammengefasst sind und im folgenden erläutert werden.

4.1 Parameter

Tabelle 4.1 enthält sämtliche Parameter, die im Zusammenhang mit Linien von Interesse sind. Ein Großteil von ihnen kann auch für nicht-linientypische Makros eingesetzt werden, beispielsweise für \pscircle.

Tabelle 4.1: Zusammenfassung aller Parameter für Linien und Polygonzüge

Name	*Werte*	*Vorgabe*
linewidth	*Wert Einheit*	0.8pt
linecolor	*Farbe*	black
linestyle	none|solid|dotted|dashed	solid|symbol
symbolStep	*Wert Einheit*	20pt
symbolWidth	*Wert Einheit*	10pt
symbolFont	*PS-Font*	Dingbats

Fortsetzung …

… Fortsetzung

Name	*Werte*	*Vorgabe*
rotateSymbol	*Boolean*	false
startAngle	*Winkel*	0
linejoin	0|1|2	0
linecap	0|1|2	0
dash	*Wert Einheit Wert Einheit* …	5pt 3pt
dotsep	*Wert Einheit*	3pt
doubleline	*Boolean*	false
doublesep	*Wert Einheit*	1.25\pslinewidth
doublecolor	*Farbe*	white
dimen	outer|inner|middle	outer
arrows	*Pfeiltyp*	-
showpoints	*Boolean*	false
linearc	*Wert Einheit*	0pt
framearc	*Wert*	0
cornersize	relative|absolute	relative
gangle	*Winkel*	0
border	*Wert Einheit*	0pt
bordercolor	*Farbe*	white
shadow	*Boolean*	false
shadowsize	*Wert Einheit*	3pt
shadowangle	*Winkel*	-45
shadowcolor	*Farbe*	darkgray
linetype	*Wert*	0
liftpen	0|1|2	0

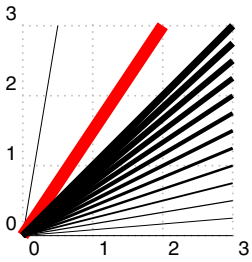
Im Folgenden wird zu jedem der angegebenen Parameter ein Beispiel angegeben, wobei sich die Reihenfolge an Tabelle 4.1 orientiert. Eine Beschreibung der Parameter zu den Fülloptionen finden sich in Kapitel 7 auf Seite 93.

4.1.1 linewidth

Grundsätzlich kann jede beliebige Liniendicke gewählt werden. Sowohl die größte als auch die kleinste Dicke orientieren sich an dem zugrundeliegenden PS-Treiber, über den TEX bzw. PS keinerlei Informationen vorliegen, sodass an dieser Stelle keine Entscheidung über Sinn oder Unsinn der Liniendicke getroffen werden kann. Variable Liniendicken sind nur für Kurvenzüge möglich (⇒ Abschnitt 5.1.4 auf Seite 67).

Zu beachten ist in jedem Fall, dass es bei der PDF-Ausgabe auf dem Bildschirm Probleme mit zu dünnen Linien geben kann, denn die Bildschirmauflösung setzt hier Grenzen. Erst der Ausdruck zeigt in der Regel die korrekte Liniendicke.

04-01-1

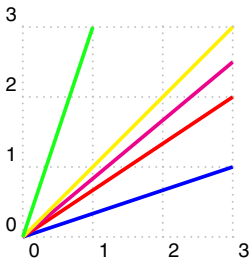



```
\usepackage{pstricks,multido}

\begin{pspicture}[showgrid](3,3)
  \psline[linewidth=0.01pt](0.5,3)
  \psline[linewidth=5pt,linecolor=red](2,3)
  \multido{\rA=0.0+0.25}{13}{%
    \psline[linewidth=\rA pt](3,\rA)}
\end{pspicture}
```

4.1.2 linecolor

Wie im Abschnitt 2.1 auf Seite 10 erwähnt, kennt PSTricks ohne externe Pakete bereits insgesamt 11 vordefinierte Farben, deren Zahl vom Anwender beliebig erweitert werden kann.

04-01-2

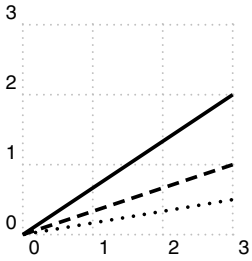



```
\usepackage{pstricks}

\begin{pspicture}[showgrid,
    linewidth=1.5pt](3,3)
  \psline[linecolor=blue](3,1)
  \psline[linecolor=red](3,2)
  \psline[linecolor=magenta](3,2.5)
  \psline[linecolor=yellow](3,3)
  \definecolor{LColor}{rgb}{0.1,1,0.1}
  \psline[linecolor=LColor](1,3)
\end{pspicture}
```

4.1.3 linestyle

Dem Beispiel kann entnommen werden, dass die erste Linie mit dem Linienstil none nicht gezeichnet wird. Ein derartiges Verhalten ist insbesondere dann interessant, wenn man beispielsweise Flächen ohne eine Randlinie füllen oder Endpunkte (Knoten) einer Linie setzen will, ohne dass diese gezeichnet wird.

04-01-3




```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,3)
 \psset{linewidth=1.5pt}
 \psline[linestyle=none](3,3)%<-- keine Linie!
 \psline[linestyle=solid](3,2)
 \psline[linestyle=dashed](3,1)
 \psline[linestyle=dotted](3,0.5)
\end{pspicture}
```

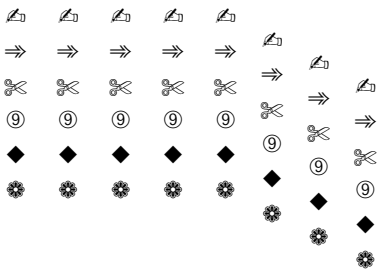
DerLinienstil symbol kann über weitere Parameter gesteuert werden, die in Tabelle 4.1 auf Seite 45 zusammengefasst sind. Standardmäßig wird das Symbol der Oktalnummer 141 (dezimal 97), welches dem Kleinbuchstaben »a« entspricht. Die Symbole können über das entsprechende Zeichen aus einem Roman-Zeichensatz oder alternativ über die

Angabe der Oktalzahl ausgewählt werden. Der sinnvolle Bereich ist dabei 041 … 176 und 241 … 376, der sich aus Tabelle 4.2 ersehen lässt, wenn die Werte links und oberhalb der Auflistung zugrunde gelegt werden.

Tabelle 4.2: Die Zuordnung zwischen Oktal- bzw. Hexadezimalzahl und Symbol beim PostScript-Zeichensatz `Zapf-Dingbats`.

	´0	´1	´2	´3	´4	´5	´6	´7	
´04x		✁	✂	✃	✄	☎	✆	✇	″2x
´05x	✈	✉	☛	☞	✌	✍	✎	✏	
´06x	✐	✑	✒	✓	✔	✕	✖	✗	″3x
´07x	✘	✙	✚	✛	<	✜	>	✝	
´10x	✞	✡	✢	✣	✤	✥	✦	✧	″4x
´11x	★	☆	✪	✫	✬	✭	✮	✯	
´12x	✰	✱	✲	✳	✴	✵	✶	✷	″5x
´13x	✸	✹	✺	✻	✼	✽	✾	✿	
´14x	❀	❁	❂	❃	❄	❅	❆	❇	″6x
´15x	❈	❉	❊	❋	●	❍	■	❑	
´16x	❒	❑	❒	▲	▼	◆	❖	◗	″7x
´17x	❙	❚	❚	❛	❜	❝	❞		
´24x		❡	❢	❢	❤	❥	❦	❧	″Ax
´25x	♣	♦	♥	♠	①	②	③	④	
´26x	⑤	⑥	⑦	⑧	⑨	⑩	❶	❷	″Bx
´27x	❸	❹	❺	❻	❼	❽	❾	❿	
´30x	①	②	③	④	⑤	⑥	⑦	⑧	″Cx
´31x	⑨	⑩	❶	❷	❸	❹	❺	❻	
´32x	❼	❽	❾	❿	➔	→	↔	↕	″Dx
´33x	➘	➙	➚	➛	➜	→	➙	➟	
´34x	➟	➡	➢	➢	➤	➥	➦	➧	″Ex
´35x	➨	➩	➪	➫	➬	➭	➭	➯	
´36x		➱	➲	➳	➴	➵	➶	➷	″Fx
´37x	➸	➹	➺	➻	➼	➽	⇛		
	″8	″9	″A	″B	″C	″D	″E	″F	

Grundsätzlich kann für die Symbole jeder der standardmäßig vorhandenen PS-Zeichensätze benutzt werden. Eine entsprechende Liste wurde bereits in Abschnitt 3.2.5 auf Seite 36 angegeben.



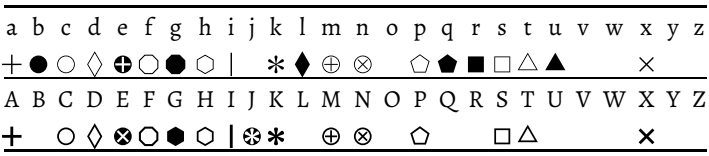
```
\usepackage{pstricks}

\pspicture(0,-1)(5,3) \psset{linestyle=symbol}
\psline(0,0)(3,0)(5,-1)
\psline[symbol=u](0,0.5)(3,0.5)(5,-.5)
\psline[symbol=310](0,1)(3,1)(5,0)
\psline[symbol=044](0,1.5)(3,1.5)(5,.5)
\psline[symbol=376](0,2)(3,2)(5,1)
\psline[symbol=055](0,2.5)(3,2.5)(5,1.5)
\endpspicture
```

04-01-4

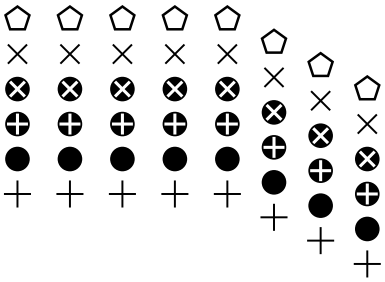
Alternativ zu den Symbolen aus den PS-Zeichensätzen können die intern definierten Symbole des Zeichensatzes `PSTricksDotFont` benutzt werden (siehe Tabelle 6.2 auf Seite 84). Es handelt sich dabei um Type-3-Vektorzeichen, auf die nur über einen zugeordneten Buchstaben zugegriffen werden kann. Eine Zusammenstellung zeigt Tabelle 4.3.

Tabelle 4.3: Die Zuordnung zwischen Zeichen und Symbol beim PSTricks-Zeichensatz `PSTricksDotFont`. Leere Felder sind dabei nicht belegt.

04-01-5



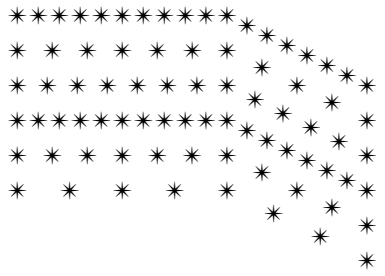
04-01-6



```
\usepackage{pstricks}

\pspicture(0,-1)(5,3)
\psset{linestyle=symbol,
  symbolFont=PSTricksDotFont}
\psline(0,0)(3,0)(5,-1)
\psline[symbol=b](0,0.5)(3,0.5)(5,-.5)
\psline[symbol=e](0,1)(3,1)(5,0)
\psline[symbol=E](0,1.5)(3,1.5)(5,.5)
\psline[symbol=x](0,2)(3,2)(5,1)
\psline[symbol=P](0,2.5)(3,2.5)(5,1.5)
\endpspicture
```

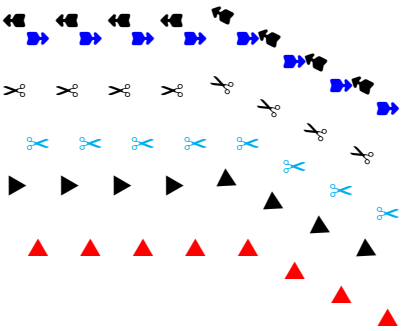
Der Abstand der Symbole kann über das optionale Argument `symbolStep` beeinflusst werden, der standardmäßig auf 20 pt festgelegt. Längenangaben ohne Einheit werden auf die aktuell gültige Maßeinheit bezogen, im Standardfall auf die vorgegebenen cm.

04-01-7



```
\usepackage{pstricks}

\pspicture(0,-1)(5,3)
\psset{linestyle=symbol,symbol=T}
\psline(0,0)(3,0)(5,-1)
\psline[symbolStep=0.5](0,0.5)(3,0.5)(5,-.5)
\psline[symbolStep=8pt](0,1)(3,1)(5,0)
\psline[symbolStep=11pt](0,1.5)(3,1.5)(5,.5)
\psline[symbolStep=14pt](0,2)(3,2)(5,1)
\psline[symbolStep=3mm](0,2.5)(3,2.5)(5,1.5)
\endpspicture
```

Die Ausrichtung der Symbole entspricht dem vorgegebenen Layout des Zeichens, unabhängig von der Steigung der jeweiligen Linie. Insbesondere bei Dreiecksymbolen kann eine Rotation angebracht sein, wenn eine Spitze des Dreiecks in Richtung der Linie zeigt. Für das Symbol der Schere kann dies ebenfalls angebracht sein. Den Winkel der Rotation kann man mit `startAngle` beeinflussen. Dies ist immer dann notwendig, wenn intern die Richtung der Linie nicht berechnet werden konnte.



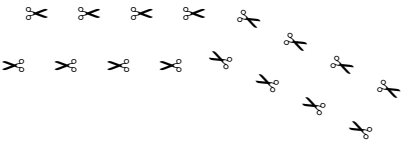
```
\usepackage{pstricks}
```

04-01-8

```
\pspicture(0,-1)(5,3)
\psset{linestyle=symbol}
\psline[symbol=163,linecolor=red](0,0)(3,0)(5,-1)
\psline[symbol=163,rotateSymbol,rot=30](0,1)(3,1)(5,0)
\psline[symbol=042,linecolor=cyan](0,1.5)(3,1.5)(5,0.5)
\psline[symbol=042,rotateSymbol](0,2.5)(3,2.5)(5,1.5)
\psline[symbol=375,linecolor=blue](0,3)(3,3)(5,2)
\psline[symbol=375,rotateSymbol](0,3.5)(3,3.5)(5,2.5)
\endpspicture
```

Linien-richtung

Zu beachten ist, dass Linienzüge intern in umgekehrter Reihenfolge gezeichnet werden und daher die Richtung der Symbole um genau 180° gedreht erscheint. Über das optionale Argument `rot` können diese bei Bedarf beliebig gedreht werden.

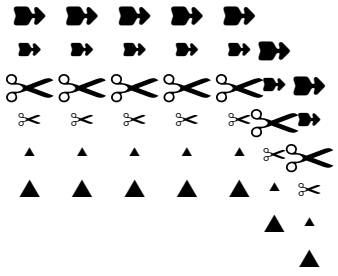


```
\usepackage{pstricks}
```

04-01-9

```
\pspicture(0,-1)(5,0.5) \psset{linestyle=symbol}
\psline[symbol=042,rotateSymbol](0,0)(3,0)(5,-1)
\psline[symbol=042,rotateSymbol,
  rot=180](0,.5)(3,.5)(5,-0.5)
\endpspicture
```

Die Symbolgröße kann über den Faktor `symbolWidth` beeinflusst werden, wobei die Skalierung symmetrisch zur Breite/Höhe erfolgt. Eine getrennte Einstellung ist hier nicht möglich. Bei eienr Größenangabe ohne Einheit wird diese auf die aktuell gültige Maßeinheit bezogen, im Standardfall auf die vorgegebenen cm.



```
\usepackage{pstricks}
```

04-01-10

```
\pspicture(0,-1)(4,3)  \psset{linestyle=symbol}
\psline[symbol=163](0,0)(3,0)(4,-1)
\psline[symbol=163,symbolWidth=5pt](0,.5)(3,.5)(4,-.5)
\psline[symbol=042](0,1)(3,1)(4,0)
\psline[symbol=042,
  symbolWidth=0.3in](0,1.5)(3,1.5)(4,.5)
\psline[symbol=375](0,2)(3,2)(4,1)
\psline[symbol=375,
  symbolWidth=0.5](0,2.5)(3,2.5)(4,1.5)
\endpspicture
```

4.1.4 `linejoin`

PS kann beim Aufeinandertreffen zweier Linienenden nicht wissen, wie diese verbunden werden sollen, denn dies kann auf unterschiedliche Weise erfolgen und wird auf PS-Ebene durch den Befehl `setlinejoin` kontrolliert. Ein entsprechender Parameter kann von TEX aus durch die Option `linejoin` an PS übergeben werden, wobei nur die Werte 0, 1 oder 2 eine Wirkung in der Ausgabe zeigen. Dieser Parameter hat insbesondere bei dicken Linien und/oder kleinen Verbindungswinkeln eine große Bedeutung.

0 Die Ränder zweier Linien werden bis zu einem Schnittpunkt verlängert.
1 Die äußere Kontur wird durch einen Kreisbogen dargestellt.
2 Die äußere Kontur wird durch eine horizontale Linie dargestellt.

```
\usepackage{pstricks}

\psset{linewidth=3mm,unit=0.8}
\begin{pspicture}(4,2) \psline(0,0)(1,2)(2,0)(3,2)(4,0)            \end{pspicture}
\begin{pspicture}(4,2) \psline[linejoin=1](0,2)(1,0)(2,2)(3,0)(4,2)\end{pspicture}
\begin{pspicture}(4,2) \psline[linejoin=2](0,0)(1,2)(2,0)(3,2)(4,0)\end{pspicture}
```

04-01-11

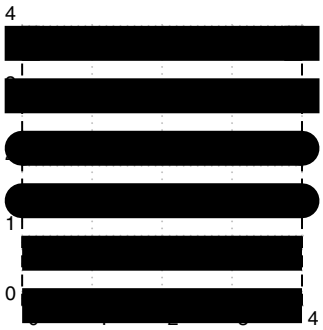
4.1.5 linecap

Das Ende von dicken Linien ist von Bedeutung und kann daher durch die Option `linecap` beeinflusst werden. Mögliche Werte sind 0, 1 oder 2.

0 Die Linien werden an den Koordinaten abgeschnitten (Standard).
1 Die Linien werden an den Koordinaten durch Halbkreise vom Radius der halben Liniendicke (0.5·`\pslinewidth`) verlängert.
2 Die Linien werden durch halbe Quadrate der Liniendicke (·`\pslinewidth`) verlängert.

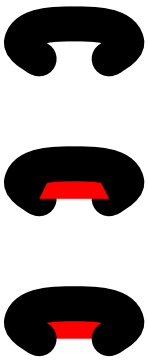
Wie dem folgenden Beispiel zu entnehmen ist, kann der Effekt auch ohne `linecap` erreicht werden, indem die entsprechenden Linienenden durch »Pfeilsymbole« festgelegt werden (siehe dazu Tabelle 8.2 auf Seite 109). Die folgenden Beispiele verwenden übertriebene Liniendicken, um den Effekt besser zeigen zu können. In der Realität wird dies so nicht immer sichtbar werden und manchmal vielleicht beim Betrachten der Grafik so gar nicht auffallen.

04-01-12




```
\usepackage{pstricks}

\begin{pspicture}[showgrid](4,4)%
  \psline[linestyle=dashed](0,4)
  \psline[linestyle=dashed](4,0)(4,4)
  \psset{linewidth=5mm}
  \psline[arrows=C-C](0,3.75)(4,3.75)
  \psline[linecap=2](0,3)(4,3)
  \psline[arrows=c-c](0,2.25)(4,2.25)
  \psline[linecap=1](0,1.5)(4,1.5)
  \psline[arrows=-](0,0.75)(4,0.75)
  \psline(4,0)
\end{pspicture}
```

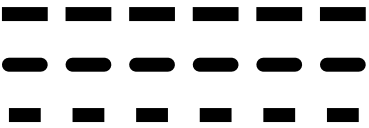
Die Anwendung der linecap-Option macht nur in ganz speziellen Fällen Sinn. Pfeile sind grundsätzlich nicht Teil eines aktuellen Pfades, sodass Fülloptionen sich nur auf den Linienteil beziehen. Dies gilt nicht für linecap, da hier das Linienende mit zum aktuellen Pfad gehört.



04-01-13

```
\usepackage{pstricks}

\def\curve{\pscurve(-.1,.1)(-.15,.15)(0,.2)(.15,.15)(.1,.1)}
\psset{unit=5cm,linewidth=5mm}
\begin{pspicture}(-0.2,-0.6)(0.2,0.5)%
\rput(0,.2){\psset{arrows=c-c}\curve}
\rput(0,-.2){%
  \psset{fillstyle=solid,fillcolor=red,arrows=c-c}%
  \curve}
\rput(0,-.6){%
  \psset{fillstyle=solid,fillcolor=red,linecap=1}%
  \curve}
\end{pspicture}
```

Für gestrichelte Linien hat linecap eine Auswirkung auf jedes einzelne Linienelement und nicht nur auf die äußeren Elemente:

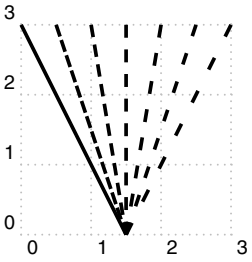


04-01-14

```
\usepackage{pstricks}

\psset{linewidth=2mm,linestyle=dashed,
  dash=5mm 5mm}
\psline[linecap=2](5,0)\\[3mm]
\psline[linecap=1](5,0)\\[3mm]
\psline[linecap=0](5,0)
```

4.1.6 dash

Voraussetzung für die Anwendung des dash-Parameters ist der Linienstil dashed.



04-01-15

```
\usepackage{pstricks,multido}

\begin{pspicture}[showgrid](3,3)
  \psset{linewidth=1.5pt,linestyle=dashed}
  \multido{\rA=0.0+1.5,\rB=0.0+0.5}{7}{%
      \psline[dash=5pt \rA pt](1.5,0)(\rB,3)}
\end{pspicture}
```

PS erlaubt eine unbegrenzte Anzahl an Intervallen für die Definition einer gestrichelten Linie.

dash=Wert1 *Einheit* Wert2 *Einheit* ...

Die Option kann auch für jedes andere Linien- oder Kurvenmakro angewendet werden.

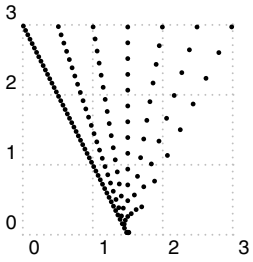
04-01-16

```
\usepackage{pstricks}

\begin{pspicture}(4,1.5)
  \psset{linestyle=dashed,linewidth=2pt}
  \psline[dash=3mm 3mm 1mm 1mm](0,1.5)(4,1.5)
  \psline[dash=5mm 2mm 0.1 0.2](0,1)(4,1)
  \psline[dash=5mm 1mm 1mm 1mm](0,0.5)(4,0.5)
  \psline[dash=5mm 1mm 1mm 1mm 1mm 1mm
    1mm 1mm 1mm 1mm](4,0)
\end{pspicture}
```

4.1.7 dotsep

Voraussetzung für die Anwendung des dotsep-Parameters ist der Linienstil . Die Größe der einzelnen Punkte orientiert sich an der Vorgabe von linewidth und ist nicht abhängig von den Parametern dotsize und dotscale, die sich auf das \psdot-Makro beziehen, welches später behandelt werden wird.

04-01-17

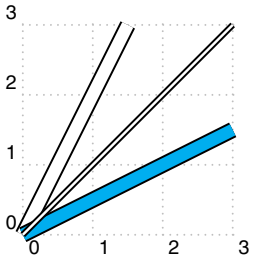



```
\usepackage{pstricks,multido}

\begin{pspicture}[showgrid](3,3)
  \psset{linewidth=2pt,linestyle=dotted}
  \multido{\rA=0.0+1.5,\rB=0.0+0.5}{7}{%
     \psline[dotsep=\rA pt](1.5,0)(\rB,3)}
\end{pspicture}
```

4.1.8 doubleline, doublesep, doublecolor

doublecolor und doublesep beziehen sich nur auf das »Innere« der Linie, die Linienfarbe und Liniendicke an sich kann mit linecolor bzw. linewidth geändert werden.

04-01-18

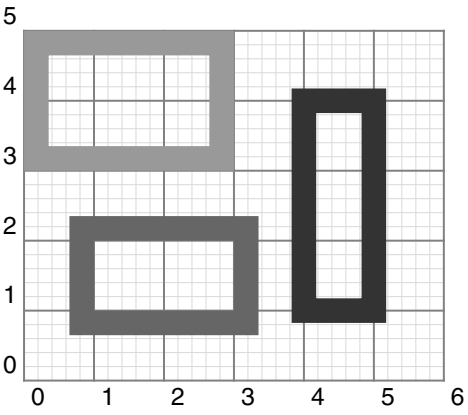



```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,3)
  \psset{doubleline=true}
  \psline[doublesep=5pt](1.5,3)
  \psline[doublesep=5pt,doublecolor=cyan](3,1.5)
  \psline(3,3)
\end{pspicture}
```

4.1.9 dimen

Diese Option bezieht sich ausschließlich auf geschlossene Linienzüge wie \psframe, \pscircle, \psellipse und \pswedge. Alle anderen sind nicht betroffen, wobei sich dimen bei \pswedge nur auf den Radius bezieht, das Zentrum liegt immer in der Mitte.

`dimen` legt fest, worauf sich die angegebenen Koordinaten beziehen, entweder auf das *Innere* (`inner`), das *Äußere* (`outer`) oder die *Mittellinie* (`middle`) des Grafik-Objekts. Die folgende Abbildung macht dies deutlicher.




```
\usepackage{pstricks}
```

04-01-19

```
\begin{pspicture}(6,5)
  \psgrid[subgriddiv=5,griddots=0,
    subgridwidth=0.1pt,
    subgridcolor=black!15,
    gridcolor=black!50]
  \psset{linewidth=10pt}
  \psframe[dimen=outer,
      linecolor=black!40](0,3)(3,5)
  \psframe[dimen=inner,
      linecolor=black!60](1,1)(3,2)
  \psframe[dimen=middle,
      linecolor=black!80](4,1)(5,4)
\end{pspicture}
```

4.1.10 `arrows`

`PSTricks` hat bereits eine große Zahl an vordefinierten Pfeilen, bzw. Linienendmarkierungen, die in Tabelle 8.2 auf Seite 109 zusammengestellt sind. Diese Pfeile können für einen Großteil der Linien- beziehungsweise Kurvenmakros alternativ über das key-value-Interface oder die spezielle Option gesetzt werden, was im Folgenden für `\psline` gezeigt wird:

```
\psline [arrows=Pfeiltyp] (x,y)...
\psline {Pfeiltyp} (x,y)...
```

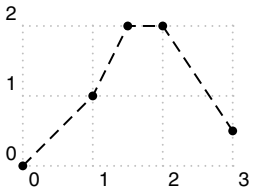
Hierin steht »*Pfeiltyp*« für einen Ausdruck der Form »*Startpfeil-Endpfeil*«, wobei sowohl *Startpfeil* als auch *Endpfeil* fakultative Angaben sind. `arrows=-` oder `{-}` ergeben somit eine Linie oder Kurve ohne Pfeile an den Enden, was der allgemeinen Vorgabe entspricht.

Aufgrund der unterschiedlichen Möglichkeiten zur Festlegung von Linienenden, sind auch widersprüchliche Angaben möglich, die jedoch keine Auswirkung haben, da immer nur die letzte Angabe wirksam ist. Ein `\psline[arrows=->]{-}(3,0)` hat daher dieselbe Wirkung wie ein `\psline(3,0)`, denn die Optionen werden auch intern in derselben Reihenfolge eingelesen und daher überschreibt »`-`« alle vorhergehenden Definitionen. Besteht die Linie aus einem Linienzug, so bezieht sich die Pfeilangabe auf den Beginn (erste Linie) und das Ende (letzte Linie) des Linienzuges. Per Definition erstellt `\pspolygon` geschlossene Linienzüge, indem eine Linie vom letzten zum ersten Punkt gezogen wird, sodass Pfeilangaben hier prinzipiell keinen Sinn machen.

4.1.11 showpoints

Dies ist primär für Bézierkurve und alle anderen Makros, die Kurven zeichnen von Interesse, um so besser zu erkennen, wo die eigentlichen Punkte liegen. Aber auch bei Linienzügen kann es in manchen Fällen sinnvoll sein, die Option `showpoints` zu setzen.

04-01-20



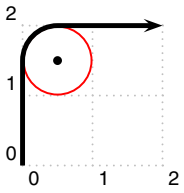

```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,2)
  \psline[showpoints,linestyle=dashed]%
      (0,0)(1,1)(1.5,2)(2,2)(3,0.5)
\end{pspicture}
```

Die Größe der Punkte kann über die Parameter `dotsize` und `dotscale` beeinflusst werden. Die Beschreibung dazu findet sich in Kapitel 6 auf Seite 83.

4.1.12 linearc

Mit dieser Option können anspruchsvolle Linienzüge erstellt werden. Prinzipiell macht die Option `linearc` nur bei Linienzügen bzw. Polygonen Sinn. Die erste Abbildung zeigt, dass der Wert für `linearc` den Radius des Kreises angibt, um den die Linie »gebogen« wird.

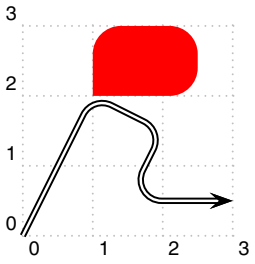
04-01-21




```
\usepackage{pstricks}

\begin{pspicture}[showgrid](2,2)
  \pscircle[linecolor=red](0.5,1.5){0.5}
  \psdot(0.5,1.5)
  \psline[linearc=0.5,linewidth=2pt]{->}(0,0)(0,2)(2,2)
\end{pspicture}
```

Wie der folgenden Abbildung zu entnehmen ist, werden Anfang und Ende eines Linienzuges, wenn sie identisch sind, normal verbunden, eine Anwendung des Parameters `linearc` erfolgt in diesem Fall nicht. Ein anderes Verhalten zeigt `\pspolygon`, welches per Definition geschlossene Kurvenzüge voraussetzt (vergleiche Abschnitt 4.4 auf Seite 59).

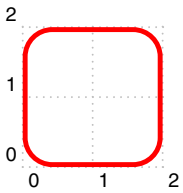
04-01-22




```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,3)
  \psline*[linecolor=red,linearc=0.4]%
      (1,2)(2.5,2)(2.5,3)(1,3)(1,2)
  \psline[linearc=0.3,doubleline=true]%
      {->}(0,0)(1,2)(2,1.5)(1.5,0.5)(3,0.5)
\end{pspicture}
```

4.1.13 framearc

Dieser Parameter ist letztlich identisch zu `linearc`, mit dem Unterschied, dass er sich auf eine geschlossene Fläche (Rahmen) bezieht, was sich primär auf `\psframe` (Abschnitt 4.5 auf Seite 59) und `\pspolygon` (Abschnitt 4.4 auf Seite 59) bezieht. Weiterhin kann `framearc` nur Werte zwischen 0 und 1 annehmen, wobei 1 sich auf die Hälfte der kürzesten Seite bezieht. Für ein Quadrat würde sich dann für den Wert 1 ein Kreis ergeben.

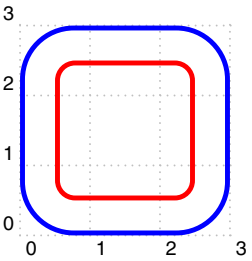


04-01-23

```
\usepackage{pstricks}

\begin{pspicture}[showgrid](2,2)
  \psframe[linewidth=2pt,framearc=0.4,
     linecolor=red](2,2)
\end{pspicture}
```

4.1.14 cornersize

Damit alle flächenförmigen Gebilde gleiches Verhalten an den Kanten zeigen, kann für `cornersize` zwischen den Werten `relative` und `absolute` gewählt werden. `relative` bezieht sich auf die Hälfte der kürzesten Seite, während `absolute` veranlasst, dass anstelle von `framearc` nun der absolute Wert von `linearc` herangezogen wird.

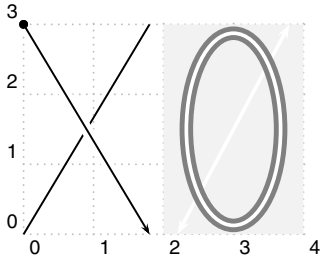


04-01-24

```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,3)
  \psframe[linewidth=2pt,linearc=0.25,
     cornersize=absolute,
     linecolor=red](0.5,0.5)(2.5,2.5)
  \psframe[linewidth=2pt,framearc=0.5,
     linecolor=blue](3,3)
\end{pspicture}
```

4.1.15 border, bordercolor

Mit dem Parameter `border` lassen sich sehr einfach Kreuzungen von Linienzügen zeigen, indem eine der Linien als oben liegend angesehen und mit einem umlaufenden Rahmen versehen wird. Mit `bordercolor` kann die Farbe festgelegt werden.



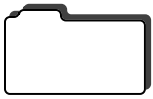
04-01-25

```
\usepackage{pstricks}

\begin{pspicture}[showgrid](4,3)
 \psline(0,0)(1.8,3)
 \psline[border=2pt]{*->}(0,3)(1.8,0)
 \psframe*[linecolor=gray!10](2,0)(4,3)
 \psset{linecolor=white}
 \psline[linewidth=1.5pt]{<->}(2.2,0)(3.8,3)
 \psellipse[linewidth=1.5pt,bordercolor=gray,
   border=2pt](3,1.5)(.7,1.4)
\end{pspicture}
```

4.1.16 shadow, shadowsize, shadowangle, shadowcolor

Schatteneffekte dienen vorrangig dazu, bestimmte Bereiche besonders hervorzuheben. Dabei sollte insbesondere die Schattengröße sorgfältig gewählt werden. Im Prinzip macht dies erst dann Sinn, wenn man geschlossene Polygonzüge hat, da nur hier die Schattenbildung auch gefüllte Flächen liefert, während sie bei Linien letztlich diese nur doppelt zeichnet, was mit der doubleline-Option ebenfalls möglich ist. Weitere Informationen dazu, wie PSTricks diese Schattenwirkung erzielt, gibt es im Kapitel 11.3.15 auf Seite 148.

04-01-26



```
\usepackage{pstricks}

\begin{pspicture}(2,1)
  \pspolygon[linearc=2pt,shadow=true,
     shadowangle=45](0,0)(0,1.1)(0.2,1.1)%
     (0.2,1.2) (0.8,1.2)(0.8,1.05)(2,1.05)(2,0)
\end{pspicture}
```

4.1.17 linetype

Die Linienstile dashed und dotted können nur dann lückenlos an bestehende Pfade (Linien oder Kurven) anschließen, wenn sie etwas über den aktuellen Pfadzustand, bzw. über die Art der Linie/Kurve wissen, die vorher gezeichnet wurde. Dies ist insbesondere für \pscustom wichtig (siehe Seite 137), wo beliebige Linien- und Kurvenarten aneinander gefügt werden können. Mit dem Parameter linetype kann man der aktuellen Kurve den eigenen Linientyp mitgeben (Tabelle 4.4).

Tabelle 4.4: Mögliche Werte für linetype

Wert	*Typ*
0	Offene Kurve ohne Pfeile
-1	Offene Kurve mit Pfeil am Anfang
-2	Offene Kurve mit Pfeil am Ende
-3	Offene Kurve mit Pfeil am Anfang und Ende
1	Geschlossene Kurve mit verschiedenen Elementen
n>1	Geschlossene Kurve mit n gleichartigen Elementen

4.1.18 liftpen

Der Parameter liftpen kontrolliert das Verhalten beim Zeichnen von offenen Kurven, was insbesondere für \pscustom von Interesse ist (→ 11.3.2 auf Seite 140). Dort finden sich auch entsprechende Beispiele.

4.1.19 labelsep

Der Parameter labelsep gibt den Abstand von den Koordinaten und von einem zu setzenden Label an, was insbesondere für das \uput-Makro von Interesse ist (siehe Kapitel 9 auf Seite 119). Dort finden sich auch entsprechende Beispiele. Der Wert für

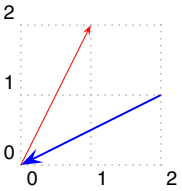
`labelsep` kann über das Längenregister `\pslabelsep` abgefragt werden und wird intern auch von anderen Makros benutzt, beispielsweise von `\psaxes` (siehe Abschnitt 14.1 auf Seite 184).

4.2 \psline

Die Syntax für Linien bzw. für polygonartige Linienzüge, die sich durch eine Folge von Koordinaten ergeben, lautet:

```
\psline* [Optionen] {Pfeilart} (x,y)
\psline* [Optionen] {Pfeilart} (x1,y1)(x2,y2)...(xn,yn)
```

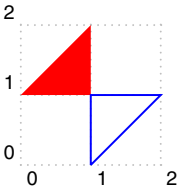
- Intern werden sämtliche angegebenen Punkte in der umgekehrten Reihenfolge abgearbeitet (LIFO-Prinzip). Dies ist wichtig zu wissen, wenn mit `\pscustom` geschlossene Linien- oder Kurvenzüge erstellt werden sollen.
- Existiert für `\psline` nur ein Koordinatenpaar, so wird grundsätzlich vom aktuellen Punkt aus zum angegeben Punkt gezeichnet, wobei der aktuelle Punkt immer auf den Koordinatenursprung $(0,0)$ gesetzt wird, wenn eine `pspicture` Umgebung existiert. Die Sternversionen führen grundsätzlich zu einem geschlossenen Polygonzug, indem vom letzten angegebenen Punkt eine Linie zum ersten gezogen wird. Danach wird der gesamte Bereich mit der Linienfarbe *gefüllt*.



04-02-1

```
\usepackage{pstricks}

\begin{pspicture}[showgrid](2,2)
  \psset{arrowscale=2}
  \psline*[linecolor=red]{->}(1,2)% sinnlos
  \psline[linecolor=blue]{<-}(2,1)
\end{pspicture}
```

Sind für Sternversionen nur ein oder zwei Punkte angegeben, so ergibt sich eine Linie mit einer komplementären Farbe, die auf dem Bildschirm sichtbar ist, aber im Ausdruck in der Regel nicht zu sehen ist. Wie in obigem Beispiel deutlich zu sehen ist, haben die Angaben zu den Pfeilen ebenfalls keine Wirkung. Dies wird hier ohnehin nur der Vollständigkeit halber gezeigt, denn obiger Fall ist praktisch sinnlos, aber eben möglich.



04-02-2

```
\usepackage{pstricks}

\begin{pspicture}[showgrid](2,2)
  \psline*[linecolor=red](0,1)(1,2)(1,1)
  \psline[linecolor=blue](1,0)(1,1)(2,1)(1,0)
\end{pspicture}
```

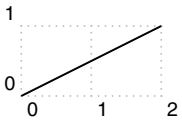
4.3 \qline

Dies stellt sozusagen die Minimalversion (**q**uick line) von `\psline` dar, denn es werden keine lokalen Optionen ausgewertet und es müssen *grundsätzlich* zwei Punkte angegeben werden:

`\qline(`x_1`,`y_1`)(`x_2`,`y_2`)`

Alle mit \psset gesetzten Parameter werden jedoch von \qline beachtet, da diese Angaben für die aktuelle und tiefer liegende Umgebungen global gelten. \psset

04-03-1




```
\usepackage{pstricks}

\begin{pspicture}[showgrid](2,1)
   \qline(0,0)(2,1)
\end{pspicture}
```

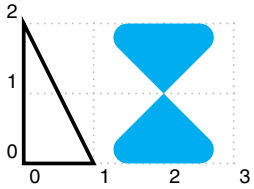
4.4 \pspolygon

Im Gegensatz zu \psline stellt \pspolygon grundsätzlich einen *geschlossenen* Kurvenzug dar, wozu eventuell die angegebenen Koordinatenpaare entsprechend ergänzt werden. Eine Angabe von Pfeilen ist daher nicht zulässig beziehungsweise sinnlos.

`\pspolygon`$*$ [*Optionen*] $(x_1,y_1)(x_2,y_2)$
`\pspolygon`$*$ [*Optionen*] $(x_1,y_1)(x_2,y_2)\ldots(x_n,y_n)$

Werden nur zwei Punkte übergeben, so wird als Start- *und* Endpunkt standardmäßig $(0,0)$ hinzugefügt. Ist der Endpunkt ungleich dem Startpunkt, so wird automatisch vom Endpunkt eine direkte Linie zum Startpunkt gezogen, um den Kurvenzug auf diese Weise zu schließen. Die Sternversion füllt das Innere des Polygonzugs mit der aktuellen Linienfarbe und dem aktuellen Füllmuster.

04-04-1




```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,2)
  \pspolygon[linewidth=1.5pt](0,2)(1,0)
  \pspolygon*[linearc=.2,linecolor=cyan,
      swapaxes=true](0,1)(0,3)(2,1)(2,3)
\end{pspicture}
```

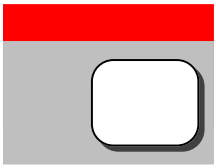
4.5 \psframe und \psTextFrame

\psframe zeichnet ein horizontal liegendes Rechteck, welches durch zwei diagonal gegenüberliegende Punkte gegeben ist.

`\psframe`$*$ [*Optionen*] (x,y)
`\psframe`$*$ [*Optionen*] $(x_1,y_1)(x_2,y_2)$
`\psTextFrame`$*$ [*Optionen*] $(x_1,y_1)(x_2,y_2)\{$*Text*$\}$

Wird für \psframe nur ein Punkt übergeben, so wird als zweiter Punkt automatisch `(0,0)` genommen. Die Sternversion füllt das Innere des Rechtecks mit der aktuellen

Linienfarbe und dem aktuellen Füllmuster. Für das Rechteck existieren die speziellen Optionen `framearc` und `cornersize`.

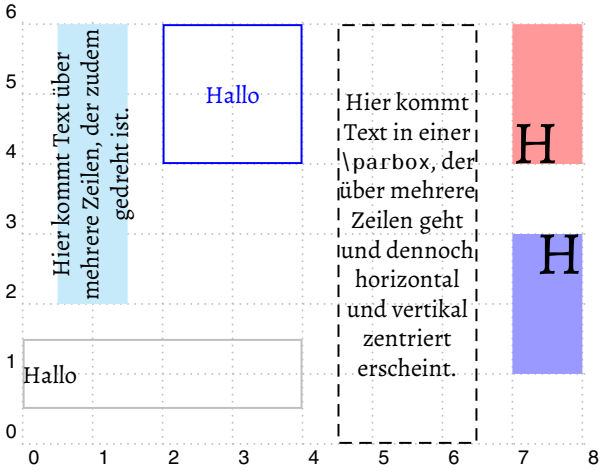


04-05-1

```
\usepackage{pstricks}

\begin{pspicture}(3,2)
 \psframe*[shadowsize=15pt,linecolor=lightgray,
  shadow=true,shadowcolor=red,shadowangle=90](3,1.75)
 \psframe[fillcolor=white,fillstyle=solid,
  framearc=0.5,shadow=true](1.25,0.25)(2.8,1.5)
\end{pspicture}
```

Das Argument *Text* bei `\psTextFrame` kann keine Zeilenumbrüche aufweisen. Werden diese benötigt, ist eine `\parbox` in der üblichen Weise anzuwenden. Die `ref`-Option erlaubt eine unterschiedliche Anordnung bezogen auf die Box und die `rot`-Option dient zum Rotieren von *Text*. Das Makro selbst benutzt `\psframe` und `\rput`.

```
\usepackage{pstricks}

\begin{pspicture}[showgrid](0,-0.5)(8,6)
\psTextFrame[linecolor=lightgray,ref=l](0,0.5)(4,1.5){Hallo}
\psTextFrame[linecolor=blue](2,4)(4,6){\color{blue}Hallo}
\psTextFrame*[linecolor=red!40,ref=lB](7,4)(8,6){\Huge H}
\psTextFrame*[linecolor=blue!40,ref=rt](7,1)(8,3){\Huge H}
\psTextFrame[linestyle=dashed](4.5,0)(6.5,6){\parbox{2cm}{\centering
  Hier kommt Text in einer \texttt{\textbackslash parbox}, der über mehrere
  Zeilen geht und dennoch horizontal und vertikal zentriert erscheint.}}
\psTextFrame*[linecolor=cyan!20,rot=90](.5,2)(1.5,6){\parbox{4cm}{\centering
  Hier kommt Text über mehrere Zeilen, der zudem gedreht ist.}}
\end{pspicture}
```



04-05-2

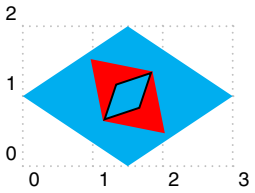
4.6 \psdiamond

\psdiamond zeichnet eine horizontal liegende Raute, welche durch ihren Mittelpunkt und ihre zwei rechtwinklig aufeinander stehenden Diagonalen gegeben ist, wobei dx und dy jeweils nur die Hälfte der Längen angeben.

```
\psdiamond* [Optionen] (dx,dy)
\psdiamond* [Optionen] (xM,yM)(dx,dy)
```

Wird nur ein Punkt übergeben, so wird als Mittelpunkt automatisch der Koordinatenursprung $(0,0)$ angenommen, unabhängig davon, ob dieser Punkt innerhalb oder außerhalb der PSTricks-Box liegt. Die Sternversion füllt das Innere der Raute mit der aktuellen Linienfarbe und dem aktuellen Füllmuster. Mit dem Parameter gangle=*Winkel* kann die Raute beliebig rotiert werden.

04-06-1




```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,2)
  \psdiamond*[linecolor=cyan](1.5,1)(1.5,1)
  \psdiamond*[linecolor=red,gangle=45]%
     (1.5,1)(0.5,0.75)
  \psdiamond[fillstyle=solid,fillcolor=cyan,
      gangle=-45](1.5,)(0.25,0.5)
\end{pspicture}
```

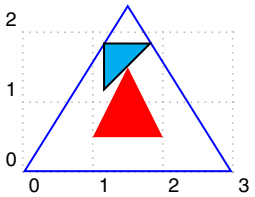
4.7 \pstriangle

\pstriangle zeichnet ein gleichschenkliges Dreieck, welches durch den Mittelpunkt der Grundlinie (Basis), die Länge dieser Grundlinie und die dazugehörige Höhe gegeben ist, wobei dx und dy die ganze Länge von Basis bzw. Höhe angeben.

```
\pstriangle* [Optionen] (dx,dy)
\pstriangle* [Optionen] (xM,yM)(dx,dy)
```

Wird nur ein Punkt übergeben, so wird als Mittelpunkt der Grundlinie automatisch der Koordinatenursprung $(0,0)$ angenommen, unabhängig davon, ob dieser Punkt innerhalb oder außerhalb der PSTricks-Box liegt. Die Sternversion füllt das Innere des Dreiecks mit der aktuellen Linienfarbe und dem aktuellen Füllmuster. Mit dem Parameter gangle=⟨*Winkel*⟩ kann das Dreieck beliebig rotiert werden.

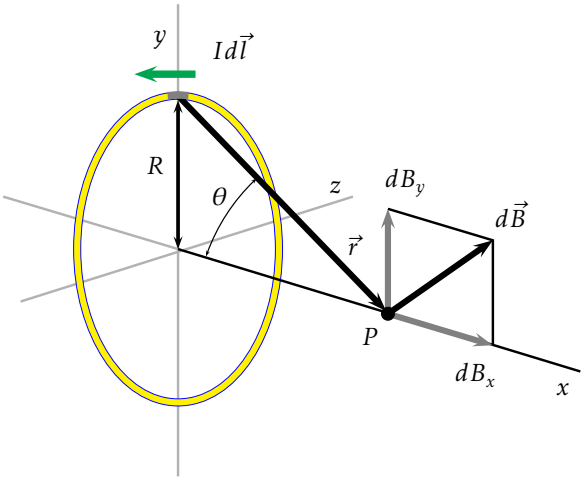
04-07-1




```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,2.4)
  \pstriangle[linecolor=blue](1.5,0)(3,2.4)
  \pstriangle*[linecolor=red](1.5,0.5)(1,1)
  \pstriangle[fillstyle=solid,fillcolor=cyan,
      gangle=45](1.5,1.5)(1,0.5)
\end{pspicture}
```

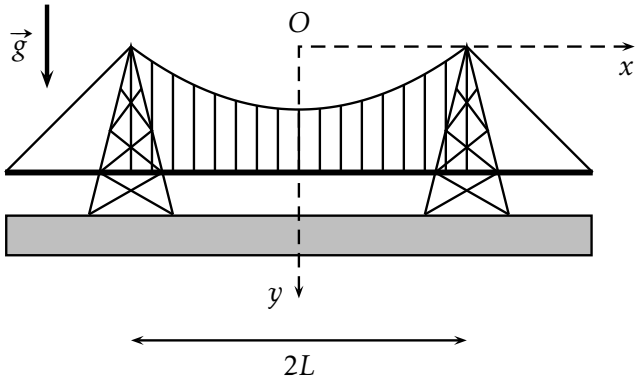
4.8 Beispiele

04-08-1



(Uwe Ziegenhagen)

04-08-2



(François Vandenbrouck)

Kapitel 5

Kreise, Ellipsen und Kurven

Alles, was keinen Polygonzug darstellt, wird formal als Kurve bezeichnet. Dazu gehören als Spezialfälle der Kreis und die Ellipse bzw. Teile davon.

5.1 Parameter

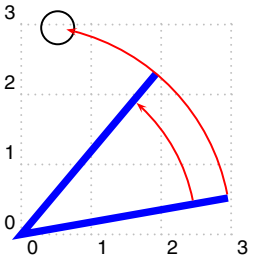
Tabelle 5.1 enthält sämtliche Parameter, die im Zusammenhang mit Kreisen, Ellipsen und Kurven von Interesse sind. Es gelten weiterhin die in Tabelle 4.1 auf Seite 45 dargestellten Parameter, soweit sie sich auf Linien allgemein und das Füllen beziehen. `arcsep` ist lediglich eine Abkürzung für das gleichzeitige Setzen von `arcsepA` (Punkt A) und `arcsepB` (Punkt B). Im Folgenden wird zu jedem der angegebenen Parameter ein Beispiel angegeben, wobei sich die Reihenfolge an Tabelle 5.1 orientiert.

5.1.1 `arcsep`, `arcsepA` und `arcsepB`

Dieser Parameter macht insbesondere dann Sinn, wenn Liniensegmente nicht im Zentrum einer anderen Linie oder einem anderem Punkt, sondern genau an der äußeren Kante enden sollen. Dies ist besonders wichtig, wenn die Linie oder Kurve mit einem Pfeil endet, denn dieser soll in der Regel mit seiner Spitze nicht in eine andere Linie oder anderen Punkt hineinragen. Wie im folgenden Beispiel zu erkennen ist, arbeitet diese Option nicht in allen Fällen einwandfrei, denn die obere Linie endet hier nicht am Kreisrand.

Name	*Werte*	*Vorgabe*
`arcsep`	*Wert Einheit*	`0pt`
`arcsepA`	*Wert Einheit*	`0pt`
`arcsepB`	*Wert Einheit*	`0pt`
`curvature`	*Wert1 Wert2 Wert3*	`1 0.1 0`
`correctAngle`	*Boolean*	`true`
`startLW`	*Wert Einheit*	`\pslinewidth`
`endLW`	*Wert Einheit*	`\pslinewidth`
`startWL`	*Wert*	`380`
`endWL`	*Wert*	`780`
`variableLW`	*Boolean*	`false`
`variableColor`	*Boolean*	`false`

Tabelle 5.1: Zusammenfassung aller Parameter für Kreise, Ellipsen und Kurven

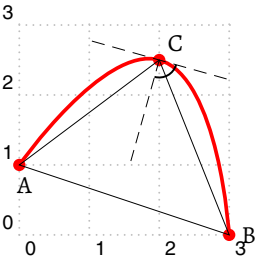


05-01-1

```
\usepackage{pstricks}

\begin{pspicture}[showgrid=true](3,3)
 \psline[linewidth=3pt,linecolor=blue](3;50)(0,0)(3;10)
 \psarc[arcsep=3pt,linecolor=red]{->}{2.5}{10}{50}
 \pscircle(3;80){0.25} \psarc[arcsepA=3pt,arcsepB=0.25cm,
     linecolor=red]{->}{3}{10}{80}
\end{pspicture}
```

5.1.2 curvature

Dieser Parameter kontrolliert das Aussehen der Kurven, die – bis auf die Bézierkurve– grundsätzlich alle gegebenen Punkte enthalten. Die Kurve ist durch ein Interpolationspolynom zweiten Grades bestimmt ($y = ax^2 + bx + c$). Dabei kann die Krümmung der Kurve über den `curvature`-Parameter beeinflusst werden. Die vorgegeben Werte genügen in der Regel den meisten Ansprüchen, können bei Kurven, die den mittleren Punkt sehr »steil anfahren« müssen, aber schon einmal zu unzureichenden Ergebnissen führen (siehe Abschnitt 14.3.1 auf Seite 215).



05-01-2

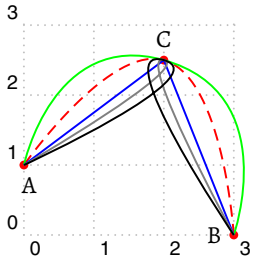
```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,3)
 \pscurve[showpoints,linecolor=red,
     linewidth=1.5pt](0,1)(2,2.5)(3,0)
 \pspolygon[linewidth=0.3pt](0,1)(2,2.5)(3,0)
 \rput[lC]{-105.7}(2,2.5){\psarc(0,0){0.25}{0}{90}
  \psset{linewidth=0.2pt,linestyle=dashed}
  \psline(0,-1)(0,1)\psline(0,0)(1.5,0)}
 \uput[-75](0,1){A}\uput[0](3,0){B}\uput[45](2,2.5){C}
\end{pspicture}
```

Wie obiges Beispiel zeigt, wird eine Kurve von `A` über `C` nach `B` gezeichnet, indem der Punkt `C` so »angefahren« wird, dass die Steigung in diesem Punkt senkrecht zur Winkel-

halbierenden des Winkels $\sphericalangle ACB$ ist. Die Winkelhalbierende ist im Beispiel gestrichelt dargestellt. Dieses Verhalten ist unabhängig sowohl von der Skalierung als auch der Wahl des `curvature` Parameters.

Die drei Parameter dürfen nur aus dem Intervall $[-1;+2]$ gewählt werden. Der erste Parameter bestimmt die Steilheit der Kurve in dem Sinne, dass für Werte kleiner als Null die Anfangssteigung kleiner als die Steigung der Strecke $\overline{AC}$ ist. Für Werte größer Null ergibt sich ein umgekehrtes Verhalten, die Steigung ist größer als die der Strecke $\overline{AC}$. Das folgende Beispiel zeigt dies für die Werte $\{2;1;0;-0.5;-1\}$. Für den Wert 0 entspricht die Kurve einem normalen Linienzug und muss dann für negative Werte eine Schleife einfügen, da sie sonst die oben genannte Bedingung der Steigung im Punkt C nicht erfüllen kann.

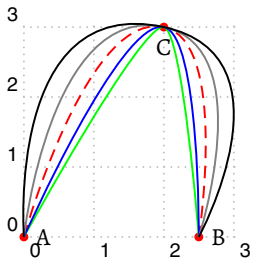
05-01-3




```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,3)
  \pscurve[showpoints,linecolor=red,%
    linestyle=dashed](0,1)(2,2.5)(3,0)
  \pscurve[linecolor=green,curvature=2 0.1 0]%
    (0,1)(2,2.5)(3,0)
  \pscurve[linecolor=blue,curvature=0.0 0.1 0]%
    (0,1)(2,2.5)(3,0)
  \pscurve[linecolor=gray,curvature=-0.5 0.1 0]%
    (0,1)(2,2.5)(3,0)
  \pscurve[linecolor=black,curvature=-1.0 0.1 0]%
    (0,1)(2,2.5)(3,0)  \uput[-75](0,1){A}
  \uput[180](3,0){B}\uput[90](2,2.5){C}
\end{pspicture}
```

Der zweite Parameter beeinflusst die Steigung rechts und links vom Zwischenpunkt, jedoch nur dann, wenn die Steigung $\overline{AC}$ bzw. $\overline{BC}$ größer als 45° ist. Um diesen Effekt zu zeigen, wurde im folgenden Beispiel der mittlere Punkt C höher gesetzt. Wie man der Abbildung entnehmen kann, erfolgt das Verhalten symmetrisch zum mittleren Punkt.

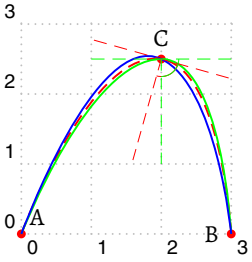
05-01-4




```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,3)
  \pscurve[showpoints,linecolor=red,%
    linestyle=dashed](0,0)(2,3)(2.5,0)
  \pscurve[linecolor=green,curvature=1 2.0 0]%
    (0,0)(2,3)(2.5,0)
  \pscurve[linecolor=blue,curvature=1 1 0]%
    (0,0)(2,3)(2.5,0)
  \pscurve[linecolor=gray,curvature=1 -0.5 0]%
    (0,0)(2,3)(2.5,0)
  \pscurve[linecolor=black,curvature=1 -1.0 0]%
    (0,0)(2,3)(2.5,0)  \uput[0](0,0){A}
  \uput[0](2.5,0){B}\uput[-90](2,3){C}
\end{pspicture}
```

Der letzte Parameter beeinflusst die Steigung in jedem Punkt. Für den Wert Null ergibt sich das oben beschriebene Verhalten. Ist der Wert dagegen gleich -1, so ist die Steigung

im Punkt C parallel zur Geraden $\overline{AB}$. Im Gegensatz zum zweiten Parameter ist das Verhalten jetzt asymmetrisch zum mittleren Punkt. Verschiebt sich die Kurve auf der linken Seite nach oben, so geht sie rechts nach unten.



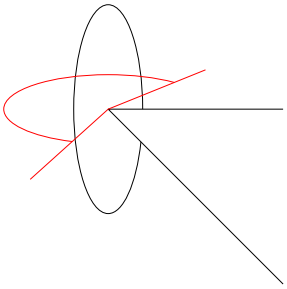
05-01-5

```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,3)
 \pscurve[showpoints,linecolor=red,%
     linestyle=dashed](0,0)(2,2.5)(3,0)
 \pscurve[linecolor=green,curvature=1 0.1 -1](0,0)(2,2.5)(3,0)
 \pscurve[linecolor=blue,curvature=1 0.1 2](0,0)(2,2.5)(3,0)
 \rput[lC]{-105.7}(2,2.5){\psset{linewidth=0.2pt,linecolor=red}
   \psline[linestyle=dashed](0,-1)(0,1)
   \psline[linestyle=dashed](0,0)(1.5,0)
   \psarc(0,0){0.25}{0}{90}}
 \rput[lC]{-90}(2,2.5){\psset{linewidth=0.2pt,linecolor=green}
   \psline[linestyle=dashed](0,-1)(0,1)
   \psline[linestyle=dashed](0,0)(1.5,0)
   \psarc(0,0){0.25}{0}{90}}  \uput[45](0,0){A}
 \uput[180](3,0){B}\uput[90](2,2.5){C}
\end{pspicture}
```

Bei Skalierungen über die `unit`-Parameter ändert sich das Verhalten der Kurve aufgrund der Vorgabe des dritten Wertes proportional. Das Verhalten der Kurve durch die ersten beiden Parameter bleibt davon unberührt.

5.1.3 `correctAngle`

Bei einem Kreis ist die Bogenlänge proportional zum Winkel $b = r \cdot \alpha$. Dies gilt nicht mehr bei einer Ellipse, da hier der Radius nicht mehr konstant ist. Um dennoch für einen doppelt so großen Winkel die doppelt so große Bogenlänge zu erhalten, wird der Winkel intern korrigiert.

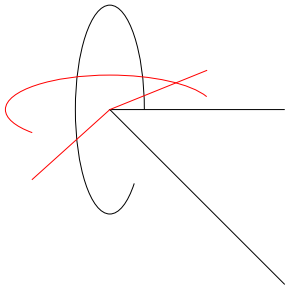


05-01-6

```
\usepackage{pstricks} \psset{unit=0.5cm}

\begin{pspicture}(-5.5,-5.5)(5.5,5.5)%
\psset{linewidth=0.4pt,linejoin=1}
\psline(5,0)(0,0)(5,-5)
\psellipticarc(0,0)(1,3){0}{315}%
\psset{linecolor=red}
\psellipticarc(0,0)(3,1){22}{222}%
\psline(3;22)\psline(3;222)
\end{pspicture}%
```

Die interne Korrektur kann durch das optionale Argument `correctAngle=false` abgeschaltet werden, sodass die Winkelangaben identisch zu denen vom Kreis sind:

05-01-7

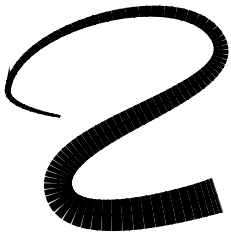


```
\usepackage{pstricks} \psset{unit=0.5cm}

\begin{pspicture}(-5.5,-5.5)(5.5,5.5)%
\psset{linewidth=0.4pt,correctAngle=false}
\psline(5,0)(0,0)(5,-5)
\psellipticarc(0,0)(1,3){0}{315}%
\psset{linecolor=red}
\psellipticarc(0,0)(3,1){22}{222}%
\psline(3;22)\psline(3;222)
\end{pspicture}%
```

5.1.4 variableLW

Um Kurvenverläufe mit unterschiedlicher Linienbreite zu erstellen, wird die gesamte Kurve mit dem PS-Befehl flattenpath in kleinstmögliche Kurvenstücke zerlegt, die dann stückweise mit unterschiedlicher Liniendicke aneinander gesetzt werden. Mit dem Parameter variableLW kann dieser Mechanismus eingeschaltet werden. Er zeigt jedoch nur bei den Makros \psellipse, \pscircle, \psarc. \psellipticarc, \pscurve, \psplot und \psparametricplot eine Wirkung.

05-01-8

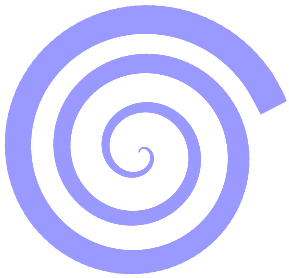


```
\usepackage{pstricks} \psset{unit=0.75}

\begin{pspicture}(-3.5,-2.5)(3.5,2.5)
\pscurve[variableLW,endLW=15pt]%
  (-1,0.5)(-2,1)(2,2)(-1,-1)(2,-1)
\end{pspicture}
```

5.1.5 startLW und endLW

Bei aktiviertem variableLW kann über die Parameter startLW und endLW der lineare Verlauf der Liniendicke festgelegt werden. Die Schrittweite wird dann aus der Kurvenlänge und der Differenz der Liniendicken bestimmt.

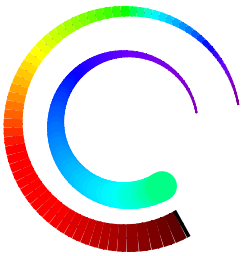
05-01-9



```
\usepackage{pst-plot}

\begin{pspicture}(-2.5,-2.5)(2.5,2.5)
\psparametricplot[variableLW,
  startLW=0.1pt,endLW=12pt,
  plotpoints=1000,linecap=2,
  linecolor=blue!40,
  algebraic]%
  {0}{20}{t/10*sin(t)|t/10*cos(t)}
\end{pspicture}
```

5.1.6 variableColor

Die Anwendung eines kontinuierlichen Farbverlaufs durch Vorgabe von `variableColor=true` setzt das Laden des Paketes `pstricks-add` voraus.



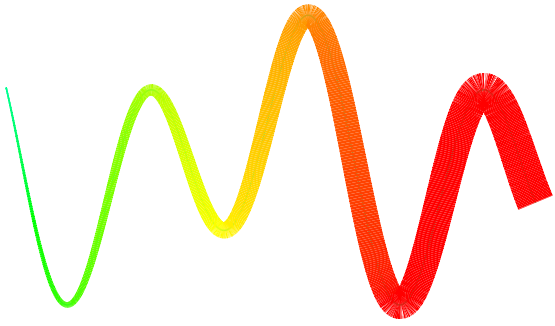
05-01-10

```
\usepackage{pstricks-add}

\begin{pspicture}(-2.5,-2.5)(2.5,2.5)
\psarc[variableLW,endLW=15pt,
  variableColor](0,0){2}{10}{300}
\psarc[variableLW,endLW=15pt,
  linecap=1,startWL=400,endWL=500,
  variableColor](0,0){1.25}{10}{300}
\end{pspicture}
```

5.1.7 startWL und endWL

Anfangs- und Endfarbe müssen als ganzzahlige Wellenlänge angegeben werden, die dann intern in jeweils ein RGB-Tripel gewandelt werden.

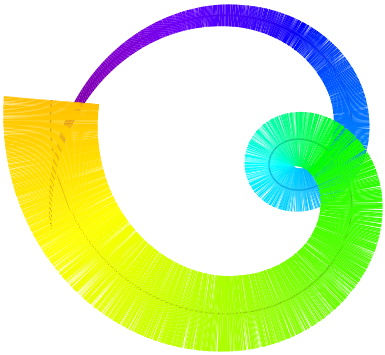
```
\usepackage{pstricks-add}

\begin{pspicture}(-5,-2)(5,3)
\psplot[variableLW,endLW=20pt,linecolor=magenta!60,variableColor,
  algebraic,plotpoints=3000,startWL=500,endWL=700]{-5}{5}{2*sin(2*x)+cos(x)}
\end{pspicture}
```



05-01-11

```
\usepackage{pstricks-add}

\begin{pspicture}(-5,-3)(5,4)
\psparametricplot[variableLW,startLW=1pt,endLW=60pt,linecolor=red,variableColor,
  algebraic,plotpoints=3000,plotstyle=curve,opacity=0.4,strokeopacity=0.4,
  endWL=600]{-5}{5}{t*sin(t) | t*cos(t)}
\end{pspicture}
```

05-01-12



5.2 Kreise und Ellipsen

PSTricks unterscheidet Kreise und Ellipsen, obwohl der Kreis bekanntlich nur ein Spezialfall der Ellipse ist. Von beiden können sowohl ganze Teile als auch Ausschnitte und Abschnitte erstellt werden.

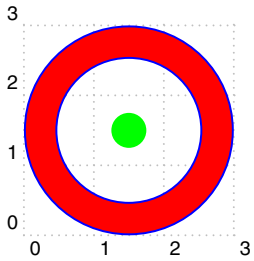
5.2.1 \pscircle

\pscircle* [*Optionen*] {*Radius*}
\pscircle* [*Optionen*] (x_M,y_M){*Radius*}

Wird kein Kreismittelpunkt angegeben, so wird automatisch der Koordinatenmittelpunkt $(0,0)$ genommen. Die Sternversion füllt das Innere des Kreises mit der aktuellen Linienfarbe und dem aktuellen Füllmuster.

Mittelpunkt

05-02-1

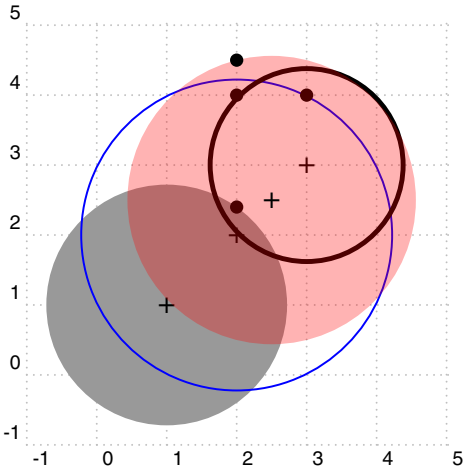



```
\usepackage{pstricks}

\begin{pspicture}[showgrid=true](3,3)
  \pscircle[linecolor=blue,doubleline=true,
     doublecolor=red,doublesep=12pt](1.5,1.5){1.5}
  \pscircle*[linecolor=green](1.5,1.5){0.25}
\end{pspicture}
```

5.2.2 \pscircleOA

\pscircleOA* [*Optionen*] (x_M,y_M)(x_A,y_A)

Das Makro erfordert die Angabe des Mittelpunktes und eines beliebigen Punktes des Kreises. Der Radius wird dann vom Makro selbst bestimmt. Die Sternversion füllt wieder das Innere des Kreises mit der aktuellen Linienfarbe und dem aktuellen Füllmuster.



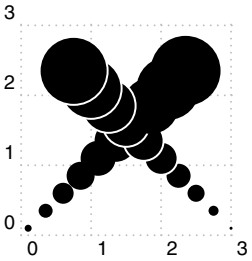
05-02-2

```
\usepackage{pstricks}

\begin{pspicture}[showgrid=true](-1,-1)(5,5)
\pscircleOA*[linecolor=black!40](1,1)(2,2.4)
\psdot[dotscale=1.5,dotstyle=+](1,1)
\psdot[dotscale=1.5](2,2.4)
\pscircleOA[linecolor=blue](2,2)(3,4)
\psdot[dotscale=1.5,dotstyle=+](2,2)
\psdot[dotscale=1.5](3,4)
\pscircleOA[linewidth=2pt](3,3)(2,4)
\psdot[dotscale=1.5,dotstyle=+](3,3)
\psdot[dotscale=1.5](2,4)
\pscircleOA*[opacity=0.3,
    linecolor=red](2.5,2.5)(2,4.5)
\psdot[dotscale=1.5,dotstyle=+](2.5,2.5)
\psdot[dotscale=1.5](2,4.5)
\end{pspicture}
```

5.2.3 \qdisk

\qdisk ist sozusagen die Minimalversion der Sternversion des Kreismakros. Parameter können nur über \psset gesetzt werden und sowohl Mittelpunkt als auch Radius müssen angegeben werden.

\qdisk(x_M,y_M){*Radius*}



05-02-3

```
\usepackage{pstricks,multido}

\begin{pspicture}[showgrid](3,3)
 \multido{\rA=0.1+0.25,\rB=0.05+0.05,%
  \rC=3.0+-0.25}{10}{%
   \qdisk(\rA,\rA){\rB}%
   \pscircle[linecolor=white,
     fillcolor=black,
     fillstyle=solid](\rC,\rA){\rB}}
\end{pspicture}
```

5.2.4 \psarc

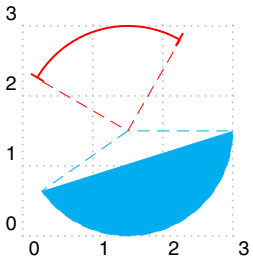
\psarc kann nicht nur Kreisbögen, sondern auch gleichzeitig Kreisabschnitte zeichnen, wenn die Füllfunktion verwendet wird. Der Kreisbogen wird im mathematisch positiven Sinne gezeichnet, also links herum (counter clockwise).

\psarc* [*Optionen*] {*Pfeile*} {*Radius*}{*Winkel1*}{*Winkel2*}
\psarc* [*Optionen*] {*Pfeile*} (x_M,y_M){*Radius*}{*Winkel1*}{*Winkel2*}

Mittelpunkt Wird kein Kreismittelpunkt angegeben, so wird standardmäßig der Koordinatenursprung $(0,0)$ genommen. Die Sternversion füllt die Fläche mit der aktuellen Linienfarbe und

dem aktuellen Füllmuster, die entsteht, wenn die Enden des Kreisbogens durch eine Sekante verbunden werden. Die Option `showpoints` bewirkt bei `\psarc` im Gegensatz zum allgemeinen Verhalten, dass ausgehend vom Mittelpunkt gestrichelte Linien zum Anfangs- und zum Endpunkt des Kreisbogens gezogen werden.

05-02-4




```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,3)
  \psarc*[showpoints,linecolor=cyan]%
    (1.5,1.5){1.5}{215}{0}
  \psarc[showpoints,linecolor=red]{|-|}%
    (1.5,1.5){1.5}{60}{150}
\end{pspicture}
```

5.2.5 \psarcn

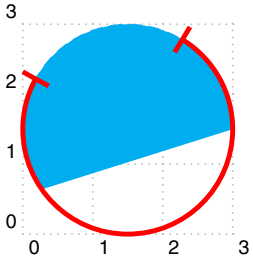
`\psarcn` ist faktisch identisch zu `\psarc` mit dem einzigen Unterschied, dass der Kreisbogen mit dem Uhrzeigersinn (clockwise), also im mathematisch negativen Sinne rechtsherum gezeichnet wird. Dasselbe lässt sich nicht mit `\psarc` erreichen, wenn man beide Winkelangaben vertauscht, denn hier wird nach wie vor entgegen dem Uhrzeigersinn gezeichnet. Insbesondere bei der Anwendung von `\pscustom`, wo es unter anderem darum geht, geschlossene Linienzüge zu zeichnen, erweist sich `\psarcn` als sehr hilfreich.

```
\psarcn * [Optionen] {Pfeile} {Radius}{Winkel1}{Winkel2}
\psarcn * [Optionen] {Pfeile} (x_M,y_M){Radius}{Winkel1}{Winkel2}
```

Wird kein Kreismittelpunkt angegeben, so wird standardmäßig `(0,0)` genommen. Die Sternversion füllt die Fläche mit der aktuellen Linienfarbe und dem aktuellen Füllmuster, die entsteht, wenn die Enden des Kreisbogens durch eine Sekante verbunden werden. Die Option `showpoints` bewirkt bei `\psarcn` im Gegensatz zum allgemeinen Verhalten, dass ausgehend vom Mittelpunkt gestrichelte Linien zum Anfangs- und zum Endpunkt gezogen werden.

Mittelpunkt

05-02-5



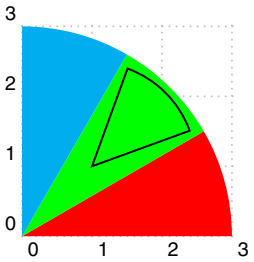

```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,3)
  \psarcn*[showpoints,linecolor=cyan]%
    (1.5,1.5){1.5}{215}{0}
  \psarcn[linecolor=red,linewidth=2pt]{|-|}%
    (1.5,1.5){1.5}{60}{150}
\end{pspicture}
```

5.2.6 \pswedge

`\pswedge` zeichnet einen Kreisausschnitt beginnend mit dem ersten angegebenen Winkel im mathematischen Sinne bis zum zweiten Winkel.

```
\pswedge* [Optionen] {Radius}{Winkel1}{Winkel2}
\pswedge* [Optionen] (x_M,y_M){Radius}{Winkel1}{Winkel2}
```

Wird kein Kreismittelpunkt angegeben, so wird standardmäßig $(0,0)$ genommen. Die
Mittelpunkt Sternversion füllt das Innere des Kreisausschnittes mit der aktuellen Linienfarbe und dem aktuellen Füllmuster.




```
\usepackage{pstricks}

\begin{pspicture}[showgrid=true](3,3)
  \pswedge*[linecolor=red]{3}{0}{30}
  \pswedge*[linecolor=green]{3}{30}{60}
  \pswedge*[linecolor=cyan]{3}{60}{90}
  \pswedge(1,1){1.5}{20}{70}
\end{pspicture}
```

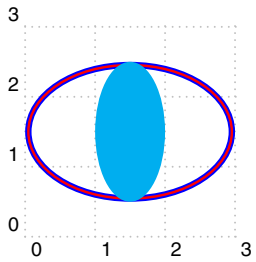
05-02-6

5.2.7 \psellipse

Wie bereits erwähnt wurde, stellt ein Kreis den Spezialfall einer Ellipse dar. Somit unterscheidet sich die Ellipse vom Kreis nur durch die erweiterte Radiusangabe, was bei einer Ellipse üblicherweise die Angabe der beiden Halbachsen ist.

```
\psellipse* [Optionen] (a,b)
\psellipse* [Optionen] (x_M,y_M)(a,b)
```

Wird kein Ellipsenzentrum angegeben, so wird automatisch der Koordinatenursprung
Mittelpunkt $(0,0)$ genommen. Die Sternversion füllt das innere der Ellipse mit der aktuellen Linienfarbe und dem aktuellen Füllmuster.

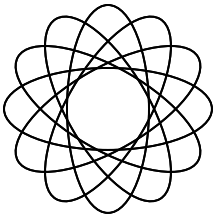



```
\usepackage{pstricks}

\begin{pspicture}[showgrid=true](3,3)
  \psellipse[linecolor=blue,doubleline=true,
     doublecolor=red](1.5,1.5)(1.5,1)
  \psellipse*[linecolor=cyan](1.5,1.5)(0.5,1)
\end{pspicture}
```

05-02-7

Eine Ellipse wird standardmäßig symmetrisch zu den Koordinatenachsen gezeichnet; die Halbachsen liegen parallel zu diesen. Mit dem Parameter `rot` kann eine Rotation der Ellipse erreicht werden. Die Drehrichtung wird im mathematisch positiven Sinne gezählt, also links herum.

05-02-8



```
\usepackage{pstricks,multido}

\begin{pspicture}(3,3)
\multido{\iA=0+30}{12}{%
  \psellipse[rot=-\iA](1.5,1.5)(1.5,0.6)}
\end{pspicture}
```

5.2.8 \psellipticarc

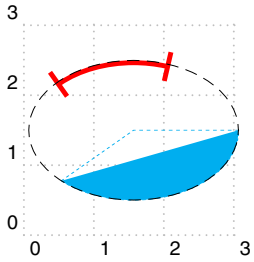
\psellipticarc kann nicht nur Ellipsenbögen, sondern auch Ellipsenabschnitte zeichnen, wenn die Füllfunktion verwendet wird. Der Ellipsenbogen wird im mathematisch positiven Sinne gezeichnet, also links herum (counter clockwise).

```
\psellipticarc* [Optionen] {Pfeile} (a,b){Winkel1}{Winkel2}
\psellipticarc* [Optionen] {Pfeile} (x_M,y_M)(a,b){Winkel1}{Winkel2}
```

Wird kein Ellipsenmittelpunkt angegeben, so wird standardmäßig der Koordinatenmittelpunkt (0,0) genommen. Die Sternversion füllt die Fläche mit der aktuellen Linienfarbe und dem aktuellen Füllmuster, die entsteht, wenn die Enden des Ellipsenbogens durch eine Sekante verbunden werden. Bei Anwendung der Option showpoints werden ausgehend vom Mittelpunkt gestrichelte Linien zum Anfangs- und zum Endpunkt gezogen, was im Gegensatz zum sonstigen Verhalten der Option steht.

Mittelpunkt

05-02-9




```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,3)
  \psellipticarc*[showpoints,
    linecolor=cyan](1.5,1.5)(1.5,1){215}{0}
  \psellipticarc[linecolor=red,
    linewidth=2pt]{|-|}(1.5,1.5)(1.5,1){60}{150}
  \psellipse[linestyle=dashed,
    linewidth=0.1pt](1.5,1.5)(1.5,1)
\end{pspicture}
```

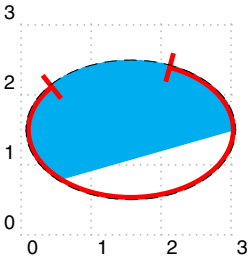
5.2.9 \psellipticarcn

\psellipticarcn ist faktisch identisch zu \psellipticarc mit dem einzigen Unterschied, dass der Ellipsenbogen mit dem Uhrzeigersinn (clockwise), also rechtsherum gezeichnet wird. Dasselbe lässt sich nicht mit \psellipticarc erreichen, wenn man beide Winkelangaben vertauscht. Insbesondere bei der Anwendung von \pscustom, wo es unter anderem darum geht, geschlossene Linienzüge zu zeichnen, erweist sich \psellipticarcn als sehr hilfreich.

```
\psellipticarcn* [Optionen] {Pfeile} (a,b){Winkel1}{Winkel2}
\psellipticarcn* [Optionen] {Pfeile} (xM,yM)(a,b){Winkel1}{Winkel2}
```

Mittelpunkt

Wird kein Ellipsenmittelpunkt angegeben, so wird standardmäßig der Koordinatenursprung `(0,0)` genommen. Die Sternversion füllt die Fläche, die entsteht, wenn die Enden des Ellipsenbogens durch eine Sekante verbunden werden, jeweils mit der aktuellen Linienfarbe und dem aktuellen Füllmuster. Die Option `showpoints` bewirkt bei `\psellipticarcn` im Gegensatz zum allgemeinen Verhalten, dass ausgehend vom Mittelpunkt gestrichelte Linien zum Anfangs- und Endpunkt gezogen werden.



05-02-10

```
\usepackage{pstricks}

\begin{pspicture}[showgrid](3,3)
  \psellipticarcn*[showpoints,
    linecolor=cyan](1.5,1.5)(1.5,1){215}{0}
  \psellipticarcn[linecolor=red,
    linewidth=2pt]{|-|}(1.5,1.5)(1.5,1){60}{150}
  \psellipse[linestyle=dashed,
    linewidth=0.1pt](1.5,1.5)(1.5,1)
\end{pspicture}
```

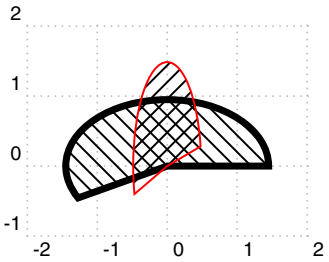
5.2.10 \pselliptic wedge

Dieses Makro ist faktisch identisch zu `\pswedge` (Abschnitt 5.2.6 auf Seite 71) mit dem einzigen Unterschied, dass ein Ellipsenausschnitt gezeichnet wird.

```
\psellipticwedge* [Optionen] (a,b){Winkel1}{Winkel2}
\psellipticwedge* [Optionen] (xM,yM)(a,b){Winkel1}{Winkel2}
```

Mittelpunkt

Wird kein Ellipsenmittelpunkt angegeben, so wird standardmäßig der Koordinatenursprung `(0,0)` genommen. Die Sternversion füllt die Fläche, die entsteht, wenn die Enden des Ellipsenbogens durch eine Sekante verbunden werden mit der aktuellen Linienfarbe und dem aktuellen Füllmuster.



05-02-11

```
\usepackage{pstricks}

\begin{pspicture}[showgrid=true](-2.2,-1)(2.2,2)
  \psellipticwedge[fillstyle=vlines,
    linewidth=0.1](0,0)(1.5,1){0}{200}
  \psellipticwedge[fillstyle=hlines,
    linecolor=red](0,0)(0.5,1.5){30}{220}
\end{pspicture}
```

5.3 Kurven

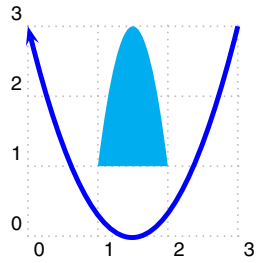
5.3.1 \psparabola

Dieses Makro benötigt die Angabe des Scheitelpunktes SP(x_S;y_S) und eines beliebigen Kurvenpunktes P(x_P;y_P), um die Parabel zeichnen zu können. Von diesem Punkt aus wird dann die Parabel bis zum Spiegelpunkt gezeichnet.

```
\psparabola* [Optionen] {Pfeile} (xP,yP)(xS,yS)
```

Die Sternversion füllt die Fläche vom Scheitelpunkt bis zu ($y = yP$) mit der aktuellen Linienfarbe und dem aktuellen Füllmuster. *Sternversion*

05-03-1




```
\usepackage{pstricks}

\begin{pspicture}[showgrid=true](3,3)
  \psparabola*[linecolor=cyan](1,1)(1.5,3)
  \psparabola[linecolor=blue,
    linewidth=2pt]{->}(3,3)(1.5,0)
\end{pspicture}
```

5.3.2 \psbezier und \pscbezier

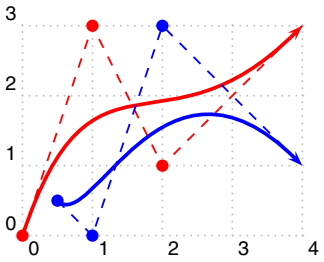
Bézierkurven sind ein wichtiges Hilfsmittel beim Zeichnen von nicht-linearen Kurvenverläufen. PS verfügt intern ebenfalls über einen entsprechenden Befehl zum Zeichnen von Bézierkurven. PSTricks benutzt prinzipiell genau diese Prozedur, die vier Punkte voraussetzt, wobei die mittleren beiden die so genannten Stützpunkte Stützpunkt darstellen, die in der Regel nur die Krümmung der Kurve angeben.[1] Innerhalb eines \pscustom-Makros kann die zweite und folgende Bézierkurve nur drei Koordinatenpaare haben, da der aktuelle Punkt als Anfangspunkt und somit viertes Wertepaar herangezogen wird.

\pscbezier unterscheidet sich nur durch das grundsätzliche Schließen des Kurvenzuges, womit hier auch keine Pfeiloptionen zur Verfügung stehen; das *c* steht beim Makronamen für »closed«.

```
\psbezier* [Optionen] {Pfeile} (x1,y1)(x2,y2)(x3,y3)
\psbezier* [Optionen] {Pfeile} (x0,y0)(x1,y1)(x2,y2)(x3,y3)
```

Werden nur drei Punkte angegeben, so wird standardmäßig der aktuelle Punkt als Startpunkt gewählt, wobei dies der Koordinatenursprung (0,0) ist, wenn die Bézierkurve nicht an einen bestehenden Kurvenzug anschließt. Die Sternversion füllt bei \pscbezier die eingeschlossene Fläche mit der aktuellen Linienfarbe und dem aktuellen Füllmuster. Bei den anderen Makros wird vorher eine Verbindungslinie vom Ende der Kurve zum Anfang gezogen wird. *Mittelpunkt*

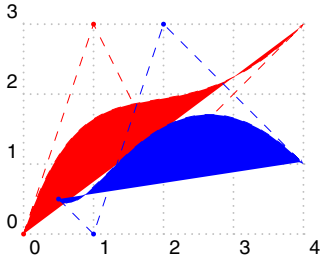
[1] Bézierkurven höherer Ordnung stellt das Paket pst-func bereit. Siehe dazu Abschnitt 27.2.2 auf Seite 564.



05-03-2

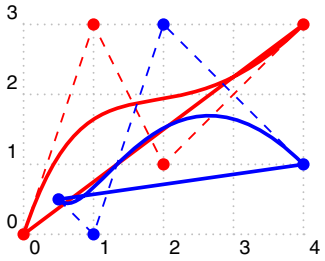
```
\usepackage{pstricks}

\begin{pspicture}[showgrid](4,3)
  \psbezier[linewidth=1.5pt,linecolor=red,
     showpoints]{->}(1,3)(2,1)(4,3)
  \psbezier[linewidth=1.5pt,linecolor=blue,
     showpoints]{->}(0.5,0.5)(1,0)(2,3)(4,1)
\end{pspicture}
```

Die Anwendung der Sternvariante macht hier nicht wirklich Sinn, denn die entstehende Fläche ist nicht unbedingt aussagekräftig.



05-03-3

```
\usepackage{pstricks}% \psbezier

\begin{pspicture}[showgrid](4,3)
  \psbezier*[linewidth=1.5pt,linecolor=red,
     showpoints]{->}(1,3)(2,1)(4,3)
  \psbezier*[linewidth=1.5pt,linecolor=blue,
     showpoints]{->}(0.5,0.5)(1,0)(2,3)(4,1)
\end{pspicture}
```



05-03-4

```
\usepackage{pstricks}

\begin{pspicture}[showgrid](4,3)
  \pscbezier[linewidth=1.5pt,linecolor=red,
     showpoints](1,3)(2,1)(4,3)
  \pscbezier[linewidth=1.5pt,linecolor=blue,
     showpoints](0.5,0.5)(1,0)(2,3)(4,1)
\end{pspicture}
```

5.3.3 \pscurve

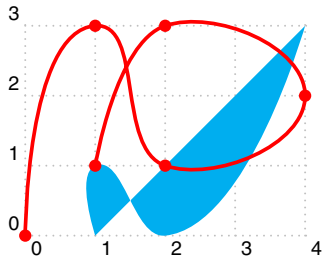
\pscurve erwartet mindestens drei Punkte, da ansonsten kein Interpolationspolynom zweiten Grades gebildet werden kann, anhand dessen PSTricks die Punkte verbindet. Zu beachten ist, dass PSTricks keine Fehlermeldung ausgibt, wenn weniger Koordinatenpaare angegeben werden! In der Regel wird aber der PS-Interpreter bei der Darstellung der Ausgabe entsprechend reagieren.

Sternversion

Die Sternversion füllt das innere der Fläche mit der aktuellen Linienfarbe und dem aktuellen Füllmuster. Es ist die Fläche, die entsteht, wenn der Endpunkt der Kurve mit dem Anfangspunkt durch eine Linie verbunden wird . Dies gilt für alle Varianten dieses Abschnitts.

\pscurve* [*Optionen*] {*Pfeile*} $(x_1,y_1)(x_2,y_2)(x_3,y_3)\ldots(x_n,y_n)$

05-03-5



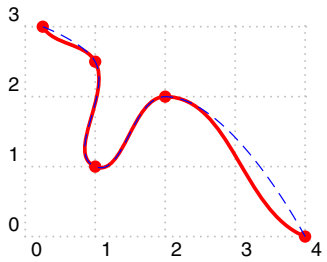

```
\usepackage{pstricks}

\begin{pspicture}[showgrid](4,3)
  \pscurve*[linecolor=cyan,
    linewidth=1.5pt](1,0)(1,1)(2,0)(4,3)
  \pscurve[linecolor=red,
    linewidth=1.5pt,showpoints]%
      (0,0)(1,3)(2,1)(4,2)(2,3)(1,1)
\end{pspicture}
```

5.3.4 \psecurve

\psecurve ist eine Abkürzung für »Endkurve« und erwartet ebenfalls mindestens drei Punkte, da ansonsten kein Interpolationspolynom gebildet werden kann. Hierbei ist zu bemerken, dass selbst drei Punkte wenig Sinn machen, da \psecurve zwar Anfangs- und Endpunkt zur Bildung des Interpolationspolynoms heranzieht, jedoch nur $n-2$ Punkte zeichnet! Zu beachten ist, dass PSTricks keine Fehlermeldung ausgibt, wenn weniger Koordinatenpaare angegeben werden! Dadurch, dass der erste und der letzte Punkt nicht dargestellt werden, kann der Kurve an den »sichtbaren« Endpunkten ein definiertes Verhalten gegeben werden, was mit \pscurve nicht möglich ist. Im folgenden Beispiel ist zum Vergleich \pscurve ohne Angabe von Anfangs- und Endpunkt eingezeichnet. Eine eher praktische Anwendung zeigt Beispiel 35-00-44 auf Seite 836.

\psecurve * [*Optionen*] {*Pfeile*} $(x_1,y_1)(x_2,y_2)(x_3,y_3)\ldots(x_n,y_n)$

05-03-6

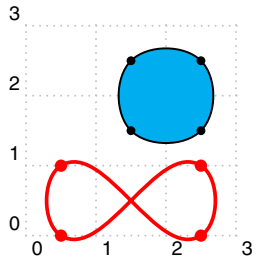



```
\usepackage{pstricks}

\begin{pspicture}[showgrid](4,3)
  \psecurve[showpoints,linecolor=red,
     linewidth=1.5pt](.125,5)(.25,3)(1,2.5)%
       (1,1)(2,2)(4,0)(8,.125)
  \pscurve[linecolor=blue,
    linewidth=0.5pt,linestyle=dashed]%
      (0.25,3)(1,2.5)(1,1)(2,2)(4,0)
\end{pspicture}
```

5.3.5 \psccurve

\psccurve ist eine Abkürzung für »closed curve« und erwartet mindestens drei Punkte, da ansonsten wieder kein Interpolationspolynom gebildet werden kann. \psccurve erstellt grundsätzlich eine geschlossene Kurve, indem die Kurve vom Endpunkt einfach zum Anfang weitergeführt wird. Zu beachten ist, dass PSTricks keine Fehlermeldung ausgibt, wenn weniger Koordinatenpaare angegeben werden!

\psccurve * [*Optionen*] $(x_1,y_1)(x_2,y_2)(x_3,y_3)\ldots(x_n,y_n)$




```
\usepackage{pstricks}
```

05-03-7

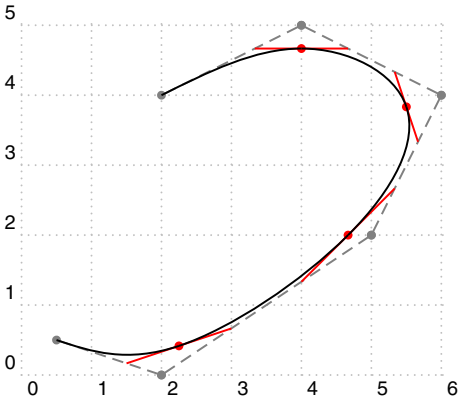
```
\begin{pspicture}[showgrid](3,3)
  \psccurve*[linecolor=cyan]%
     (1.5,1.5)(2.5,1.5)(2.5,2.5)(1.5,2.5)
  \psccurve[showpoints]%
     (1.5,1.5)(2.5,1.5)(2.5,2.5)(1.5,2.5)
  \psccurve[showpoints,linecolor=red,
     linewidth=1.5pt](.5,0)(2.5,1)(2.5,0)(.5,1)
\end{pspicture}
```

5.4 Kubische B-Splines

Ein Spline n-ten Grades ist eine Funktion, die stückweise aus Polynomen mit maximalem Grad n zusammengesetzt wird. Dabei werden an den Knoten, an denen zwei Polynomstücke zusammentreffen, bestimmte Bedingungen gestellt, beispielsweise eine $(n-1)$-malige stetige Differenzierbarkeit. Das Paket `pst-bspline` von Michael Sharpe unterstützt die Anwendung kubischer B-Splines.[2]

`\psBspline` [*Optionen*] {*Präfix*} $(x_1,y_1)(x_2,y_2)\ldots(x_n,y_n)$
`\psBsplineC*` [*Optionen*] {*Präfix*} $(x_1,y_1)(x_2,y_2)\ldots(x_n,y_n)$
`\psBsplineE` [*Optionen*] {*Präfix*} $(x_1,y_1)(x_2,y_2)\ldots(x_n,y_n)$

Der einzige spezielle Parameter ist `showframe`, mit dem zum einen ein gestrichelter Polygonzug der Kontrollpunkte und zum anderen die Tangenten der berechneten Kurvenpunkte erstellt werden. Die berechneten Punkte werden ebenfalls markiert.




```
\usepackage{pst-bspline,multido}
```

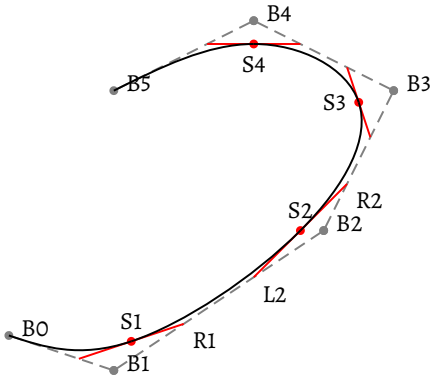
05-04-1

```
\begin{pspicture}[showgrid](-.5,-.5)(6,5)
\psBspline[showframe]%
  (0.5,0.5)(2,0)(5,2)(6,4)(4,5)(2,4)
\end{pspicture}
```

Durch das optionale Argument *Präfix* kann auf die Punkte als Knoten zugegriffen werden. Im folgenden Beispiel lauten diese Punkte $B0, B1, \ldots$, $BL0, BL1, \ldots$, $BS0, BS1, \ldots$ und $BR0, BR1, \ldots$, wobei L für Links und R für Rechts steht.

[2] Zur Theorie siehe auch `http://www.math.ucla.edu/~baker/149.1.02w/handouts/dd_splines.pdf` und `http://mathworld.wolfram.com/B-Spline.html`

05-04-2

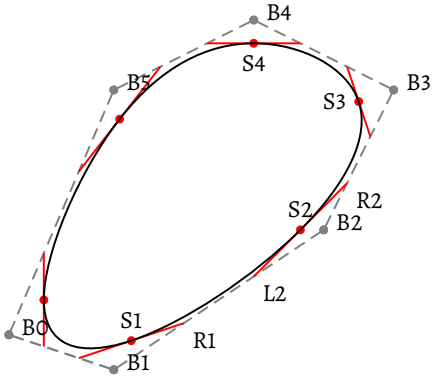



```
\usepackage{pst-bspline,multido}

\begin{pspicture}(-0.5,-0.5)(6,5)
\psBspline[showframe]{B}%
  (0.5,0.5)(2,0)(5,2)(6,4)(4,5)(2,4)
\multido{\i=0+1}{6}{\uput[20](B\i){B\i}}
\uput[90](BS1){S1} \uput[90](BS2){S2}
\uput[180](BS3){S3}\uput[270](BS4){S4}
\uput[-45](BR1){R1}\uput[-45](BL2){L2}
\uput[-45](BR2){R2}
\end{pspicture}
```

Geschlossene Kurven erhält man mit `\psBsplineC`, welches prinzipiell dem Makro `\psccurve` entspricht, wobei die Sternversion den Inhalt automatisch mit der Linienfarbe füllt. Für die »Periodizität« gilt:

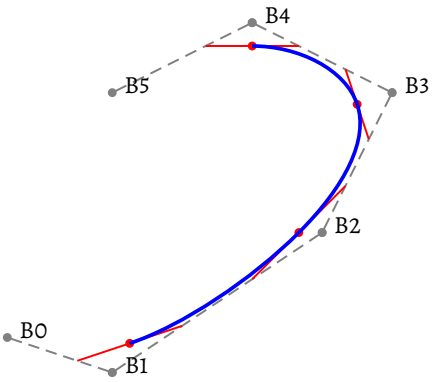
- B_i wird auf eine Periode $n+1$ erweitert, sodass $B_{n+1} = B_0$ und $B_{n+2} = B_1$.
- R_i, L_i für $0 < i < n+2$, wird wie im Normalfall ermittelt.
- S_k ist der Mittelpunkt von $L_k R_k$ für $0 < k < n+2$.
- Festlegung von $S_0 = S_{n+1}$.
- Für $0 < k \leq n+1$ wird die kubische Bézierkurve mit den Kontrollpunkten S_{k-1}, R_{k-1}, L_k, S_k konstruiert, wobei $k-1 \leq t \leq k$.

05-04-3




```
\usepackage{pst-bspline,multido}

\begin{pspicture}(-0.5,-0.5)(6,5)
\psBsplineC[showframe]{B}%
  (0.5,0.5)(2,0)(5,2)(6,4)(4,5)(2,4)
\multido{\i=0+1}{6}{\uput[20](B\i){B\i}}
\uput[90](BS1){S1} \uput[90](BS2){S2}
\uput[180](BS3){S3}\uput[270](BS4){S4}
\uput[-45](BR1){R1}\uput[-45](BL2){L2}
\uput[-45](BR2){R2}
\end{pspicture}
```

Das Makro `\psBsplineE` verhält sich analog zu `\psecurve`; der Anfangs- und der Endpunkt werden nicht gezeichnet, aber für die Berechnung der Kurve berücksichtigt.




```
\usepackage{pst-bspline,multido}

\begin{pspicture}(-0.5,-0.5)(6,5)
\psBsplineE[showframe]{B}%
  (0.5,0.5)(2,0)(5,2)(6,4)(4,5)(2,4)
\multido{\i=0+1}{6}{\uput[20](B\i){B\i}}
\psBsplineE[linewidth=1.5pt,linecolor=blue]%
  (0.5,0.5)(2,0)(5,2)(6,4)(4,5)(2,4)
\end{pspicture}
```

05-04-4

Die bisher angegeben drei Makros erwarten die explizite Angabe der Punkte, wohingegen die folgenden Makros nur das Knotenpräfix und die Koordinaten, beziehungsweise den höchsten Index $(n-1)$ der gegebenen Knoten erwarten.

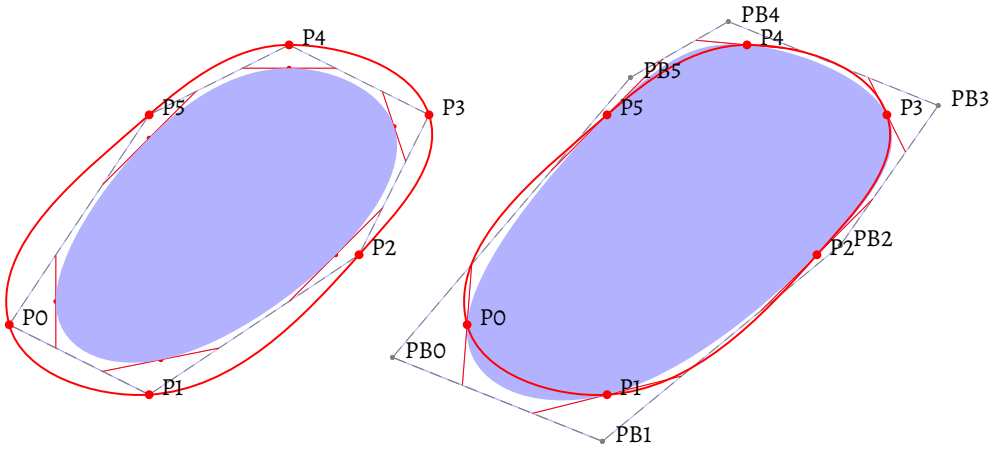
`\pnodes{`*Präfix*`}`$(x_0,y_0)(x_1,y_1)(x_n,y_n)$
`\psBsplineNodes` [*Optionen*] `{`*Präfix*`}{`*MaxIndex*`}`
`\psBsplineNodesC` [*Optionen*] `{`*Präfix*`}{`*MaxIndex*`}`
`\psBsplineNodesE` [*Optionen*] `{`*Präfix*`}{`*MaxIndex*`}`
`\psBsplineInterp{`*Präfix*`}{`*MaxIndex*`}`

Mit `\pnodes{Foo}(2,1.5)(3,4)(5,1)` würden durch interne Anwendung des Makros `\pnode` (siehe Abschnitt 15.3.3 auf Seite 264) die Knoten `Foo0`, `Foo1` und `Foo2` mit den angegebenen Koordinaten definiert. Eine entsprechende Bézierkurve kann dann mit `\psBsplineNodes{Foo}{2}` gezeichnet werden. Im folgenden Beispiel ist zum Vergleich eine Kurve mit dem Makro `\psccurve` gezeichnet worden.

Mit dem Makro `\psBsplineInterp` kann man die Problemstellung umkehren; die gegebenen Punkte werden als Kontrollpunkte aufgefasst und daraus die eigentlichen Punkte berechnet. Das Makro `\psBsplineInterpC` hängt dabei das Suffix B an die bekannten Knotennamen an, sodass im folgenden Beispiel die Knoten `PB0`, `PB1`, … entstehen.

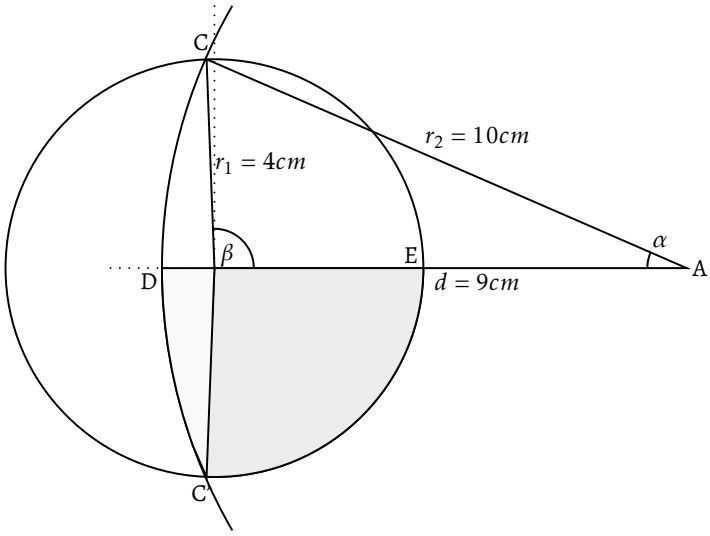
```
\usepackage{pstricks,pst-bspline,multido}

\begin{pspicture}(-.5,-.5)(6,5)
\pnodes{P}(0,1)(2,0)(5,2)(6,4)(4,5)(2,4)
\psBsplineNodesC*[showframe,linecolor=blue!30]{P}{5}
\psccurve[linecolor=red,showpoints](0,1)(2,0)(5,2)(6,4)(4,5)(2,4)
\multido{\i=0+1}{6}{\uput[20](P\i){P\i}}
\end{pspicture}
\begin{pspicture}(-.5,-.5)(6,5)
\pnodes{P}(0,1)(2,0)(5,2)(6,4)(4,5)(2,4)  \psBsplineInterpC{P}{5}
\psBsplineNodesC*[showframe=true,linecolor=blue!30]{PB}{5}
\psccurve[linecolor=red,showpoints](0,1)(2,0)(5,2)(6,4)(4,5)(2,4)
\multido{\i=0+1}{6}{\uput[20](P\i){P\i}} \multido{\i=0+1}{6}{\uput[20](PB\i){PB\i}}
\end{pspicture}
```

05-04-5

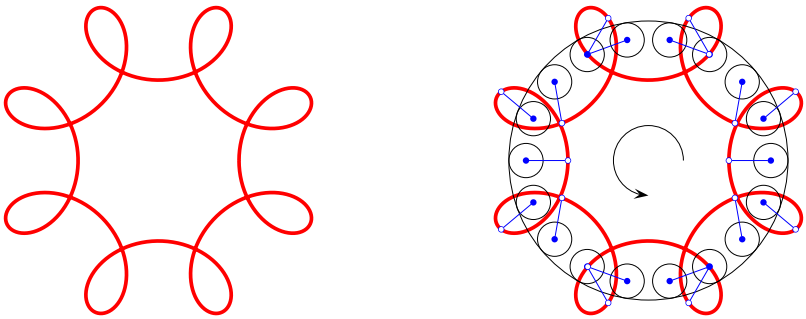


5.5 Ergänzende Beispiele

05-05-1



05-05-2



05-05-3

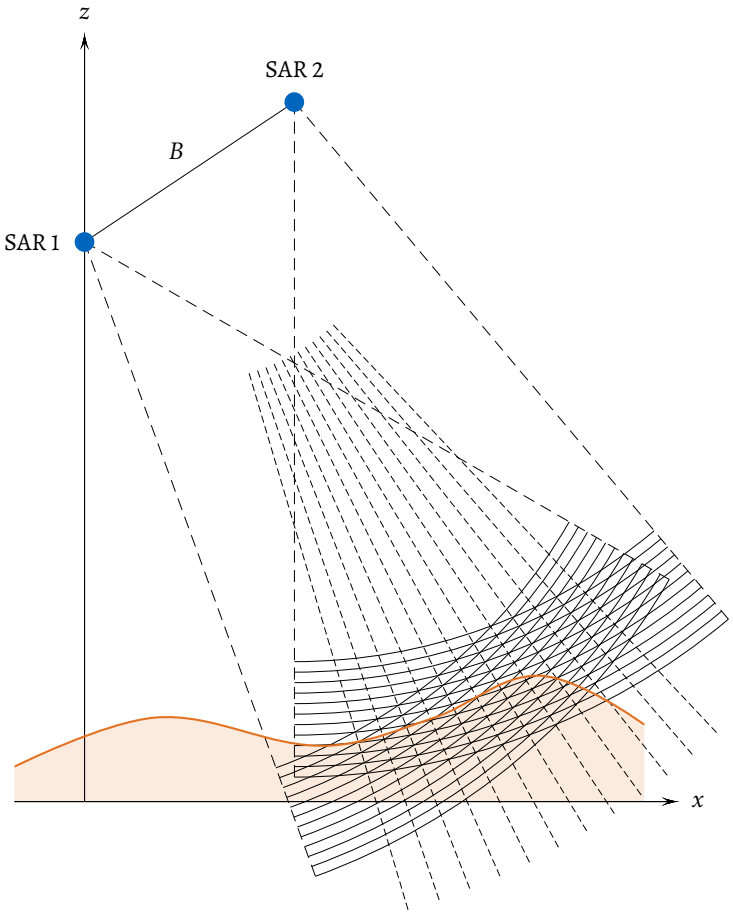


Abbildung 5.1: (Uwe Siart)

Kapitel 6

Punkte

Unter Punkt (dot) wird hier alles verstanden, was sich über die Option `dotstyle` als solcher festlegen lässt. Bei einigen, beispielsweise dem senkrechten Strich, wäre die Bezeichnung Symbol sicherlich angebrachter, denn diese »Punkte« können jede beliebige Form und Größe annehmen. Insbesondere die Symbole des `ZapfDingbats`-Fonts bieten sich an, hier verwendet zu werden, zumal mit `\ding{`*Zeichennummer*`}` ein leichter Zugriff auf die einzelnen Zeichen möglich ist. Am Ende dieses Kapitels wird gezeigt, wie derartige Symbole als neue Punkte definiert werden können. Die Definition von Punktstil und -größe hat unmittelbaren Einfluss auf die in Abschnitt 4.1.11 auf Seite 55 behandelte Option `showpoints`.

6.1 Parameter

Die folgende Tabelle zeigt die möglichen Optionen für die Punktmakros.

Tabelle 6.1: Zusammenfassung aller Parameter für Punktdarstellungen

Name	*Werte*	*Vorgabe*
`dotstyle`	*Stilname* (siehe Tabelle 6.2)	`*`
`dotsize`	*Wert Einheit Wert*	`2pt 2`
`dotscale`	*Wert1 Wert2*	`1`
`dotangle`	*Winkel*	`0`

6.1.1 `dotstyle`

Es existiert eine große Anzahl an vordefinierten Stilen für das Setzen von Punkten, die alle in Tabelle 6.2 zusammengefasst sind, wobei die rechte Spalte jeweils die Sternversion

zeigt, bei der »offene« Symbole mit der aktuellen Linienfarbe gefüllt werden (verfügbar erst ab Version 1.13). Zur besseren Darstellung wurden die Symbole mit `dotscale=1.5` gesetzt (vgl. Abschnitt 6.1.3 auf der nächsten Seite). Weitere Symbole lassen sich einfach definieren, sodass hier prinzipiell keinerlei Einschränkung gegeben ist (vergleiche Abschnitt 6.3 auf Seite 86).

Tabelle 6.2: Zusammenfassung der Punktstile

06-01-1

Name	\psdot	\psdot*	*Name*	\psdot	\psdot*
*	● ● ●	● ● ●	o	○ ○ ○	● ● ●
Bo	○ ○ ○	● ● ●	x	× × ×	× × ×
+	+ + +	+ + +	B+	+ + +	+ + +
Add	+ + +	+ + +	BoldAdd	+ + +	+ + +
Oplus	⊕ ⊕ ⊕	● ● ●	BoldOplus	⊕ ⊕ ⊕	● ● ●
SolidOplus	✪ ✪ ✪	● ● ●	Hexagon	⬡ ⬡ ⬡	⬢ ⬢ ⬢
BoldHexagon	⬡ ⬡ ⬡	⬢ ⬢ ⬢	SolidHexagon	⬢ ⬢ ⬢	⬢ ⬢ ⬢
asterisk	✻ ✻ ✻	✻ ✻ ✻	Basterisk	✻ ✻ ✻	✻ ✻ ✻
Asterisk	✻ ✻ ✻	✻ ✻ ✻	BoldAsterisk	✻ ✻ ✻	✻ ✻ ✻
SolidAsterisk	❂ ❂ ❂	● ● ●	oplus	⊕ ⊕ ⊕	⊕ ⊕ ⊕
otimes	⊗ ⊗ ⊗	⊗ ⊗ ⊗	Otimes	⊗ ⊗ ⊗	● ● ●
BoldOtimes	⊗ ⊗ ⊗	● ● ●	SolidOtimes	⊗ ⊗ ⊗	● ● ●
Mul	× × ×	× × ×	BoldMul	× × ×	× × ×
|	❙ ❙ ❙	❙ ❙ ❙	B|	❙ ❙ ❙	❙ ❙ ❙
Bar	| | |	| | |	BoldBar	❙ ❙ ❙	❙ ❙ ❙
Bullet	● ● ●	● ● ●	Circle	○ ○ ○	● ● ●
BoldCircle	○ ○ ○	● ● ●	square	□ □ □	■ ■ ■
Bsquare	□ □ □	■ ■ ■	square*	■ ■ ■	■ ■ ■
Square	□ □ □	■ ■ ■	BoldSquare	□ □ □	■ ■ ■
SolidSquare	■ ■ ■	■ ■ ■	diamond	◊ ◊ ◊	♦ ♦ ♦
Bdiamond	◊ ◊ ◊	♦ ♦ ♦	diamond*	♦ ♦ ♦	♦ ♦ ♦
Diamond	◊ ◊ ◊	♦ ♦ ♦	BoldDiamond	◊ ◊ ◊	♦ ♦ ♦
SolidDiamond	♦ ♦ ♦	♦ ♦ ♦	triangle	△ △ △	▲ ▲ ▲
Btriangle	△ △ △	▲ ▲ ▲	triangle*	▲ ▲ ▲	▲ ▲ ▲
Triangle	△ △ △	▲ ▲ ▲	BoldTriangle	△ △ △	▲ ▲ ▲
SolidTriangle	▲ ▲ ▲	▲ ▲ ▲	pentagon	⬠ ⬠ ⬠	⬟ ⬟ ⬟
Bpentagon	⬠ ⬠ ⬠	⬟ ⬟ ⬟	pentagon*	⬟ ⬟ ⬟	⬟ ⬟ ⬟
Pentagon	⬠ ⬠ ⬠	⬟ ⬟ ⬟	BoldPentagon	⬠ ⬠ ⬠	⬟ ⬟ ⬟
SolidPentagon	⬟ ⬟ ⬟	⬟ ⬟ ⬟	Hexagon	⬡ ⬡ ⬡	⬢ ⬢ ⬢
BoldHexagon	⬡ ⬡ ⬡	⬢ ⬢ ⬢	SolidHexagon	⬢ ⬢ ⬢	⬢ ⬢ ⬢
Octogon	○ ○ ○	● ● ●	BoldOctogon	○ ○ ○	● ● ●
SolidOctogon	● ● ●	● ● ●			

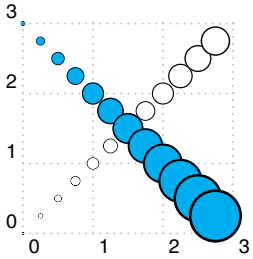
6.1.2 `dotsize`

Die Symbolgröße (Durchmesser oder Höhe) setzt sich zusammen aus der normalen Größenangabe und zusätzlich (additiv) aus einer optionalen Zahl, die ein Vielfaches von `linewidth` ist (Abschnitt 4.1.1 auf Seite 46). Damit ist es möglich, die Punktgröße ausschließlich auf die Liniendicke zu beziehen. Für Kreise bezieht sich die Angabe von `dotsize` auf den Durchmesser.

06-01-2

```
\usepackage{pstricks,pst-node}

\psdot[dotsize=0pt 10,dotstyle=square](0,0)%
\psdot[dotsize=0pt 10,dotstyle=square](2,0)%
\pcline[nodesep=5\pslinewidth,
  linewidth=10\pslinewidth](0,0)(2,0)
```

06-01-3

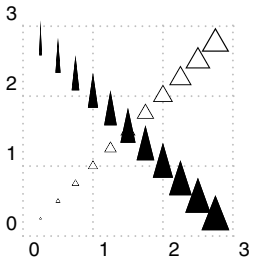



```
\usepackage{pstricks,multido}

\begin{pspicture}[showgrid=true](3,3)
  \multido{\nA=1+1,\rA=0.0+0.25}{12}{%
    \psdot[dotsize=\nA pt,dotstyle=o](\rA,\rA)}
  \psset{fillcolor=cyan}
  \multido{\nA=1+1,\rA=0.0+0.25,\rB=3+-0.25}{12}{%
    \psdot[dotsize=\nA pt \nA,dotstyle=o](\rA,\rB)}
\end{pspicture}
```

6.1.3 dotscale

Die Skalierung kann über zwei Angaben vorgenommen werden, wobei der zweite Wert optional ist. Fehlt er, so gibt der erste Parameter den Wert für die Skalierung in horizontaler und vertikaler Richtung an, andernfalls gibt der erste die Skalierung in horizontaler und der zweite die Skalierung in vertikaler Richtung an.

06-01-4

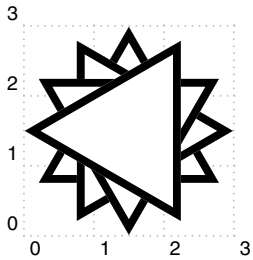



```
\usepackage{pstricks,multido}

\begin{pspicture}[showgrid=true](3,3)
  \multido{\rA=0.25+0.25}{11}{%
    \psdot[dotscale=\rA,dotstyle=triangle](\rA,\rA)}
  \multido{\rA=0.25+0.25,\rB=2.75+-0.25}{11}{%
    \psdot*[dotscale=\rA\space 4,
        dotstyle=triangle](\rA,\rB)}
\end{pspicture}
```

6.1.4 dotangle

Nach Anwendung der anderen Parameter wie `dotsize` und `dotscale`, wird das Symbol um den Winkel `dotangle` gedreht. Die Drehrichtung erfolgt wieder im mathematisch positiven Sinne, also links herum.

06-01-5

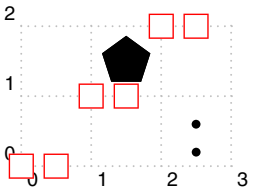



```
\usepackage{pstricks,multido}

\begin{pspicture}[showgrid=true](3,3)
  \multido{\nA=0+30}{12}{%
    \psdot[dotsize=2.25cm,dotstyle=triangle,
      dotangle=\nA](1.5,1.5)}
\end{pspicture}
```

6.2 \psdot und \psdots

Wie bereits in der Einleitung zu diesem Kapitel erwähnt wurde, versteht man unter »dot« bzw. Punkt oder Symbol, alles was sich über den Parameter `dotstyle` (siehe dazu Abschnitt 6.1.1 auf Seite 83) definieren lässt.

```
\psdot* [Optionen]
\psdot* [Optionen] (x,y)
\psdots* [Optionen] (x1,y1)(x2,y2)...(xn,yn)
```




```
\usepackage{pstricks}

\begin{pspicture}[showgrid=true](3,2)
  \psdot*[dotstyle=pentagon,dotscale=5](1.5,1.5)
  \psdots[dotsize=.4cm,dotstyle=square,
    linecolor=red](0,0)(0.5,0)(1,1)(1.5,1)(2,2)(2.5,2)
  \psdots*(2.5,0.6)\psdot(2.5,0.2)
\end{pspicture}
```

06-02-1

Koordinaten

Ohne Angabe von Koordinaten werden die aktuellen TEX-Koordinaten genommen, was der Textposition oder innerhalb einer `pspicture`-Umgebung dem Koordinatenursprung entspricht.

6.3 TEXnisches

Grundsätzlich kann man sich seine eigenen Symbole mit den folgenden Makros definieren.

```
\newpsfontdot{Name}[xW xS yW yS xO yO]{Fontname}{<n>}
\newpsfontdot{Name}[xW xS yW yS xO yO]{PSTricksFontDot}{(x)}
\newpsfontdotH{Name}[xW xS yW yS xO yO]{PSTricksFontDot}{(x)}{(y)}
```

Die Werte des zweiten Arguments entsprechenden den Elementen der so genannten Transformationsmatrix, ihre Bedeutung ist in der Tabelle 6.3 erklärt. Die zur Verfügung stehenden PS-Fonts hängen zum einen vom verwendeten Treiber und zum anderen natürlich von den Druckerzeichensätzen ab. Alternativ kann man auch den `PSTricks`-internen PS-Font `PSTricksFontDot` verwenden, dessen Zeichen in der Headerdatei `pst-dots.pro` definiert werden. Für diesen Fall ist die Platznummer innerhalb des 256 Zeichen umfassenden Fonts durch ein korrespondierendes ASCII-Zeichen anzugeben, beispielsweise `(A)` für das 65. Zeichen (siehe Beispiel auf Seite 88). Mit `\newpsfontdotH` werden grundsätzlich Zeichen definiert, die sich aus der Überlagerung zweier Zeichen ergeben, die man mit unterschiedlichen Farben versehen kann, so dass nach der Überlagerung der Eindruck entsteht, als könne man Linien- und Füllfarbe getrennt wählen.

Type 3

Beim Font `PSTricksFontDot` handelt es sich formal um einen Type3-Font, obwohl es auch ein sogenannter Vektorfont ist! Type3 bedeutet nach Definition lediglich, dass der Font nicht den Type1-Konventionen folgt; es muss daher nicht notwendigerweise ein

Bitmap-Font sein, der bei starken Vergrößerungen zur Treppenbildung neigt, wie sonst üblich. Alle in PSTricks definierten Symbole haben Vektorcharakter, auch wenn sie in der Fontliste als Type3 auftauchen. Dies ist wichtig zu wissen, da viele Druckereien immer wieder der Meinung sind, dass das Dokument Bitmapfonts eingebunden hätte. Der Grund für die Type3-Konvention ist die Tatsache, dass es für normale Type1-Zeichen nicht möglich ist, Linien- und Füllfarbe getrennt zu wählen.

xW	x-Skalierungsfaktor	xS	x-Scherungsfaktor
yW	y-Skalierungsfaktor	yS	y-Scherungsfaktor
xO	x-Offset	yO	y-Offset
Fontname	PSTricks oder PS-Font	n	Zeichennummer (Hexadez.)
PSTricksDotFont	—	(x)	Kürzel Zeichen1
		(y)	Kürzel Zeichen2

Tabelle 6.3: Die Parameter des Makros \newpsfontdot.

Exemplarisch sollen hier einmal vier neue Symbole CirclePlus, CirclePlus45, CircleMultiply und Flower in verschiedenen Varianten definiert und angewendet werden, die allerdings alle aus den Standard-PostScript-Zeichensätzen stammen. Ein Beispiel zu einem selbst definierten Symbol findet man auf den folgenden Seiten. Die ersten drei Symbole sind im Symbol- und das Flower-Symbol im ZapfDingbats-Font von PS enthalten. [42]

```
\usepackage{pstricks}
\newpsfontdot{CircMultiply}[2 0 0 2 -.78 -.7]{Symbol}{<C4>}
\newpsfontdot{CircPlus}[2 0 0 2 -.78 -.7]{Symbol}{<C5>}
\newpsfontdot{CircPlus45}[2 2 -2 2 -.78 -.7]{Symbol}{<C5>}
\newpsfontdot{Flower}[2 0 0 2 -.78 -.7]{Dingbats}{<60>}
% ... Andere Definitionen siehe TeX-Code ...


\begin{pspicture}[showgrid](4,3)
 \psset{dotscale=2.5}           \psdot[dotstyle=Flower](4,0)
 \psdot[dotstyle=Flower45](4,1) \psdot[dotstyle=Flower90](4,2)
 \psdot[dotstyle=Flower135](4,3)\psdots[dotstyle=hFlower](0,0)
 \psdot[dotstyle=vFlower](1,1)  \psdot[dotstyle=hvFlower](0,1)
 \psdot[dotstyle=xsFlower](2,2) \psdot[dotstyle=ysFlower](3,3)
 \psdot[dotstyle=dxyFlower](2,3)\psdot[dotstyle=CircPlus](3,0)
 \psdots[dotstyle=CircPlus45](3,2)(1,2)(2,1)
 \psdots[dotstyle=CircMultiply](0,3)
\end{pspicture}
```

06-03-1