Einführungskurs PostScript, Teil 4

Bézier-Kurven in PostScript

PostScript ermöglicht das Zeichnen kubischer Bézier-Kurven, die durch Angabe der 4 Kontrollpunkte spezifiziert werden.

Befehle:

x2 y2 x3 y3 x4 y4 **curveto**

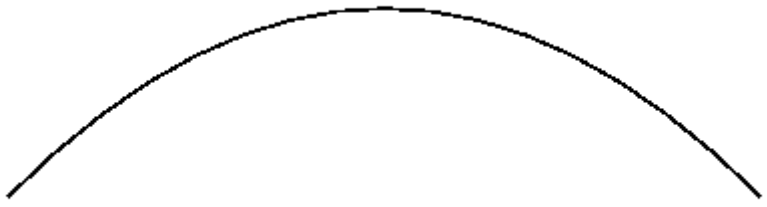
dx2 dy2 dx3 dy3 dx4 dy4 **rcurveto** % relative Positionierung

die Parameter geben den 2., 3, und 4. Kontrollpunkt (Endpunkt) an – als Anfangspunkt wird der aktuelle Punkt genommen.

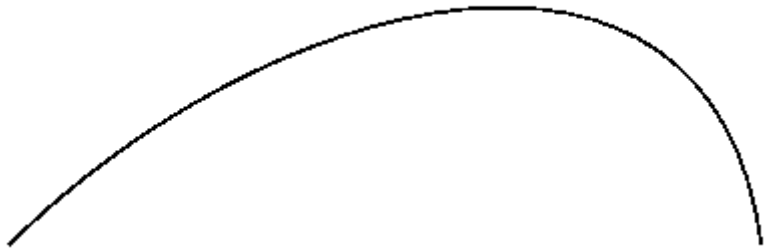
Beispiel:

```
100 100 moveto
200 200 300 200 400 100 curveto
stroke
showpage
```

Ergebnis:



Ersetzung der zweiten Zeile durch
200 200 380 250 400 100 curveto
setzt den dritten Kontrollpunkt weiter nach rechts und nach oben:



Bezierkurven programmieren

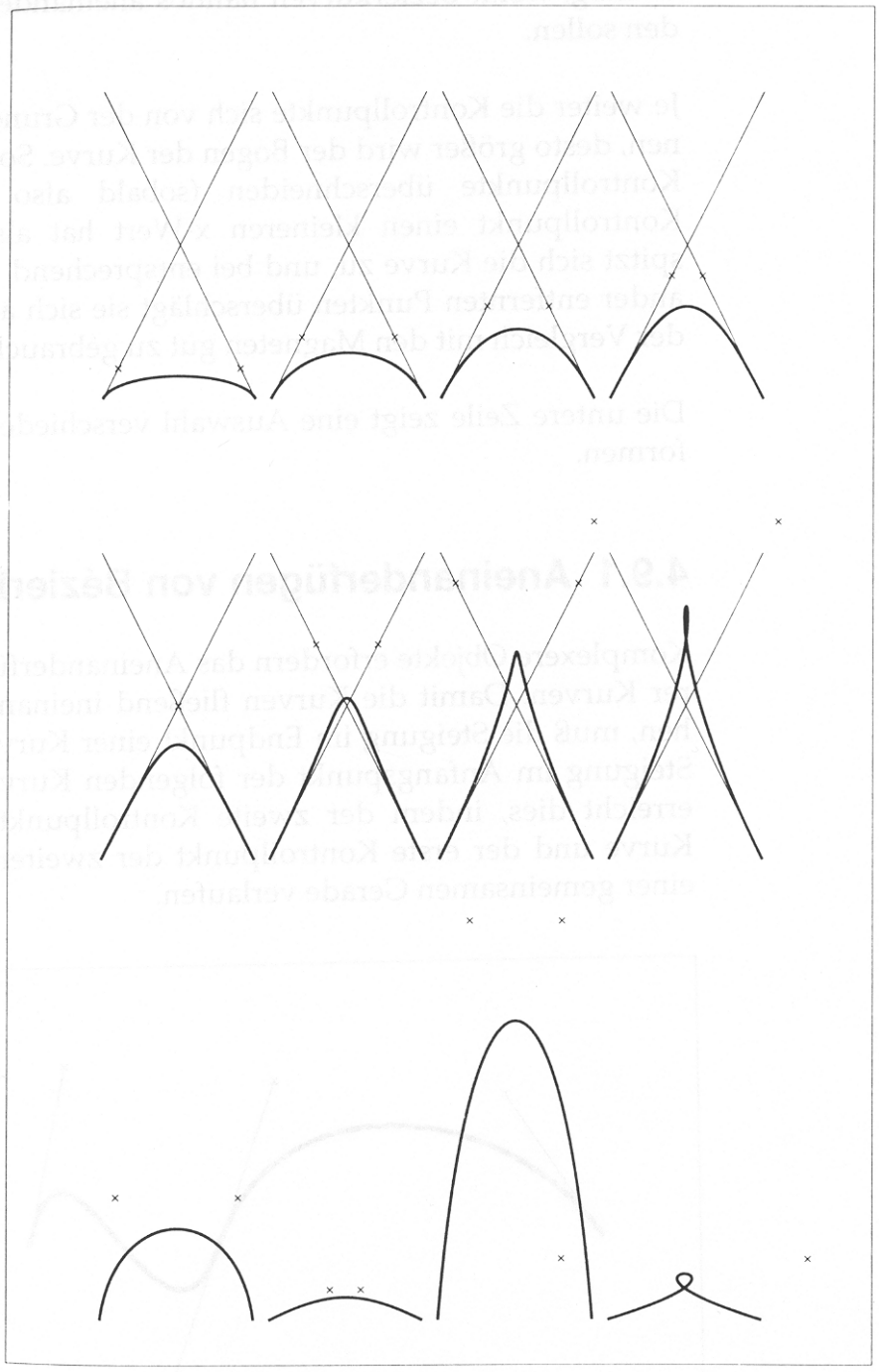
PostScript ist in der Lage, Bezierkurven zu erzeugen. Dabei handelt es sich um mathematische Kurvenbeschreibungen, die aus der Angabe von nur vier Koordinaten bestehen. Dem Start- und Endpunkt der Kurve sowie zwei Kontrollpunkten. Bezierkurven sind so flexibel, dass sich damit alle denkbaren Kurvenverläufe programmieren lassen.

Die mathematische Definition der Kurven bewirkt, dass sie bei Vergrößerungen oder Verkleinerungen für die neue Auflösung auch neu berechnet werden können. Dadurch bleiben Rundungen immer rund. Auf diese Weise funktionieren auch die beliebig skalierbaren PostScript-Zeichensätze: Die Zeichen sind ebenfalls mit Bezierkurven definiert worden.

Zwei Befehle dienen der Programmierung von Bezierkurven: *curveto* und *rcurveto.* Allerdings greifen auch *arc, arcn* und *arcto* intern auf diese Kurvenform zurück.

Wer zum ersten Mal Bezierkurven erzeugen soll, wird es schwierig finden, die vier Parameter so zu wählen, dass die beabsichtigte Kurve entsteht.

Es ist daher wichtig, einiges über den Erzeugungsmechanismus zu wissen. Man kann sich die beiden Kontrollpunkte zum Beispiel als Magneten vorstellen, die die Kurve zu sich heranziehen. Eine korrektere Vorstellung demonstriert die nächste Abbildung:



Die Wirkungsweise der Kontrollpunkte

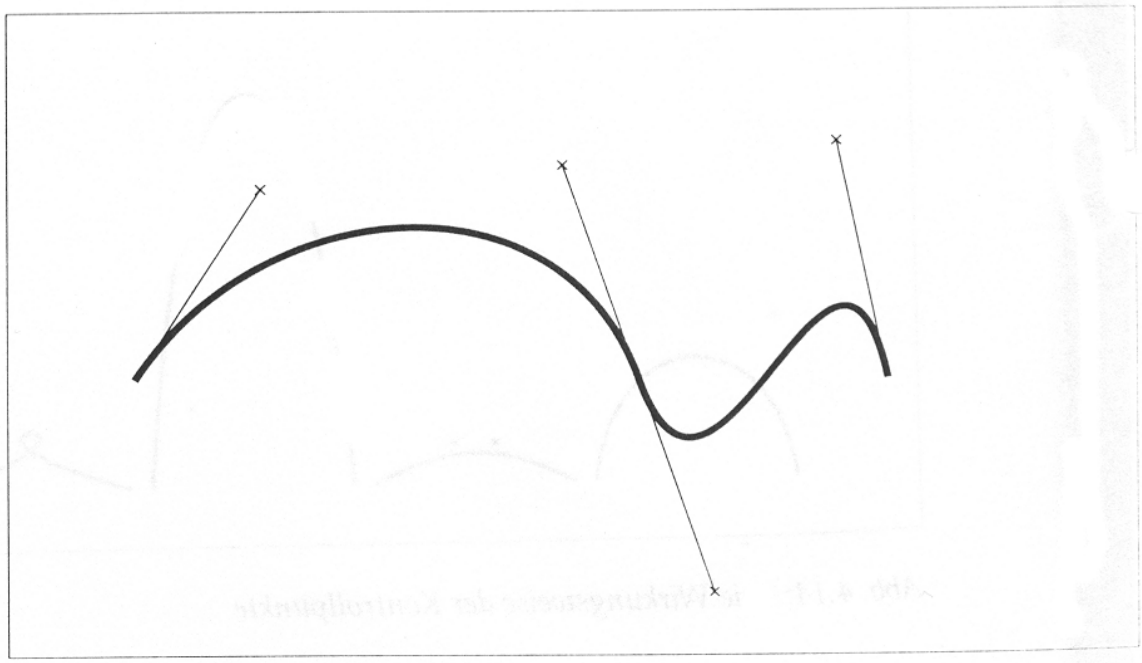
Betrachten Sie zunächst die oberen beiden Zeilen: Der erste Kontrollpunkt bildet mit dem Startpunkt der Kurve (der durch den aktuellen Punkt bestimmt wird) eine Gerade. Ebenso bildet der zweite Kontrollpunkt mit dem Endpunkt
Betrachten Sie zunächst die oberen beiden Zeilen: Der erste Kontrollpunkt bildet mit dem Startpunkt der Kurve (der durch den aktuellen Punkt bestimmt wird) eine Gerade. Ebenso bildet der zweite Kontrollpunkt mit dem Endpunkt
eine Gerade. Die beiden Geraden bestimmen die Steigung der Kurve im Start-, bzw. Endpunkt: Dort ist die Kurvensteigung gleich der Geradensteigung. Diese Tatsache wird wichtig, wenn Bezierkurven nahtlos aneinandergefügt werden sollen.

Je weiter die Kontrollpunkte sich von der Grundlinie entfernen, desto größer wird der Bogen der Kurve. Sobald sich die Kontrollpunkte überschneiden (sobald also der zweite Kontrollpunkt einen kleineren x-Wert hat als der erste), spitzt sich die Kurve zu, und bei entsprechend weit voneinander entfernten Punkten überschlägt sie sich auch. Hier ist der Vergleich mit den Magneten gut zu gebrauchen.

Die untere Zeile zeigt eine Auswahl verschiedener Kurvenformen.

Aneinanderfügen von Bezierkurven

Komplexere Objekte erfordern das Aneinanderfügen mehrerer Kurven. Damit die Kurven fließend ineinander übergehen, muss die Steigung im Endpunkt einer Kurve gleich der Steigung im Anfangspunkt der folgenden Kurve sein. Man erreicht dies, indem der zweite Kontrollpunkt der ersten Kurve und der erste Kontrollpunkt der zweiten Kurve auf einer gemeinsamen Gerade verlaufen.



Aneinanderfügen zweier Bézierkurven

In der Theorie klingt die Programmierung der Kurvenfunktionen einleuchtend, aber in der Praxis ist es mühsam, die richtigen Koordinaten für fließende Kurvenübergänge auszurechnen. Das ist der Grund, warum die an sich faszinierenden Möglichkeiten der Bezierkurven in der Praxis so selten selbst programmiert werden. Im nächsten Abschnitt finden Sie deshalb eine Befehlserweiterung, die es Ihnen ermöglicht, auf komfortable Weise mit Bezierkurven umzugehen. Dabei übernimmt PostScript selbst die mühsame Rechenarbeit.

Befehlserweiterung: Bezierkurven leicht gemacht

Das folgende Programm gestattet es, Bezierkurven fließend an den aktuellen Pfad anzuhängen. Es können beliebig viele Kurven folgen, die allesamt fließend ineinander übergehen. Mit Hilfe eines neuen line-Befehls ist es auch möglich, Kurven fließend in einer Geraden auslaufen zu lassen (an die sich dann weitere Kurven anschließen könnten).

```
% Programm 4-12

/4sichern {
    6 -2 roll
    pop pop
} bind def

/slope {
    gsave
        flattenpath
        mark
        -1 -1 -1 -1
        {4sichern} {4sichern} {} {} pathforall
        exch
        4 -1 roll
        sub
        3 1 roll
        exch
        sub
        exch
        atan
        /CSL exch def
        cleartomark
    grestore
} bind def

/tan {
    dup
    sin
    exch
    cos
    div
} bind def

/anti {
    180 exch sub 360 exch sub
} bind def

/UL {
    /Awert exch def
    CSL 90 eq CSL 270 eq or {
        CSL 90 eq {/Sx1 0 def /Sy1 Awert def} if
        CSL 270 eq {/Sx1 0 def /Sy1 Awert neg def} if
    }{
```

```
        CSL 180 gt {/modus -1 def}
        {/modus 1 def} ifelse
        /Alt 1 CSL tan dup mul add sqrt def
        /Faktor Alf Awert div def
        /Sx1 1 Faktor div abs def
        CSL 90 gt CSL 270 lt and {/Sx1 Sx1 neg def} if
        /Sy1 CSL tan Faktor div abs modus mul def
    } ifelse
    Sx1 Sy1 rlineto
} bind def

/UC {
    userdict /CEW known not {/CEW 150 def} if
    userdict /CAW known not {/CAW 150 def} if
    userdict /CSL known not {slope} if
    /Ewert CEW def
    /Est exch def
    /Ey exch def
    /Ex exch def
    /Awert CAW def
    /Ast CSL def

    currentpoint
    /Ay exch def
    /Ax exch def
    /CSL Est anti def

    Ex Ax le {
            % ja, verläuft so, also Werte tauschen
        Ax Ay Ex Ey
        /Ay exch def
        /Ax exch def
        /Ey exch def
        /Ex exch def
        Ast Est
        /Ast exch def
        /Est exch def
        /richtung -1 def
    }{
        /richtung 1 def
    } ifelse

    Ast 90 eq Ast 270 eq or {
        Ast 90 eq {/Sx1 0 def /Sy1 Awert def} if
        Ast 270 eq {/Sx1 0 def /Sy1 Awert neg def} if
    }{
        /Alt 1 Ast tan dup mul add sqrt def
        /Faktor Alf Awert div def
        /Sx1 1 Faktor div abs def
        Ast 180 gt {/modus -1 def} {/modus 1 def} ifelse
        /Sy1 Ast tan Faktor div abs modus mul def
        Ast 90 gt Ast 270 lt and {/Sx1 Sx1 neg def} if
    } ifelse
```

```
    Est 90 eq Est 270 eq or {
        Est 90 eq {/Sx2 0 def ISy2 Ewert def} if
        Est 270 eq {/Sx2 0 def ISy2 Ewert neg def} if
    }{
        /Elt 1 Est tan dup mul add sqrt def
        /Faktor Elf Ewert div def
        /Sx2 1 Faktor div abs def
        Est 180 gt {/modus -1 def} {/modus 1 def} ifelse
        /Sy2 Est tan Faktor div abs modus mul def
        Est 90 lt Est 270 gt or {/Sx2 Sx2 neg def} if
    } ifelse

    /Ex Ex Ax sub def
    /Ey Ey Ay sub def



    /Sx2 Ex Sx2 sub def
    /Sy2 Ey Sy2 neg sub def

    richtung 1 eq {
        Sx1 Sy1 Sx2 Sy2 Ex Ey rcurveto
    }{
        Ex neg Sx2 add
        Ey neg Sy2 add
        Ex neg Sx1 add
        Ey neg Sy1 add
        Ex neg
        Ey neg
        Rcurveto
    } ifelse
} bind def

300 600 moveto
/CSL 90 def
/CAW 150 def
/CEW 150 def
100 600 90 UC
300 200 90 UC
100 200 270 UC
stroke
5 setlinewidth
300 600 moveto
/CSL 100 def
/CAW 200 def
/CEW 200 def
100 600 80 UC
300 200 100 UC
100 200 280 UC
300 200 100 anti UC
stroke
100 100 moveto
```

```
300 300 lineto
slope
/CEW 90 def
/CAW 80 def
460400110 UC
30 UL
stroke


showpage
```

Es gibt drei globale Variablen, die Sie definieren müssen:

CSL	"Current SLope"	Steigung der Kurve im Anfangspunkt
CAW	"Current AnfangsWert"	Entfernung des ersten Kontrollpunktes
CEW	"Current EndWert"	Entfemung des zweiten Kontrollpunktes

Wenn bereits ein Pfad existiert, dann können Sie die Prozedur "slope" aufrufen, die die Steigung des Pfades im Endpunkt berechnet und so automatisch "CSL" setzt. Wenn noch kein aktueller Pfad existiert, dann können Sie die Steigung selbst bestimmen. Die Steigung wird allerdings als Winkel angegeben, denn sonst könnte es keine senkrecht auf- oder ablaufenden Kurvensegmente geben, weil dafür die Steigung im herkömmlichen Sinn "unendlich" ist. Es können Winkel von 0 bis 360 Grad eingegeben werden.

Die Variablen "CAW" und "CEW" geben an, wie weit die Kontrollpunkte vom Anfangs- bzw. Endpunkt entfernt sein sollen (Abstand in Einheiten des Benutzerkoordinatensystems). Die genaue Position berechnet das Programm selbst aus den Steigungsangaben. Wie in Abbildung zu sehen war, bestimmen Sie mit diesen beiden Werten die Amplitude der Kurven.

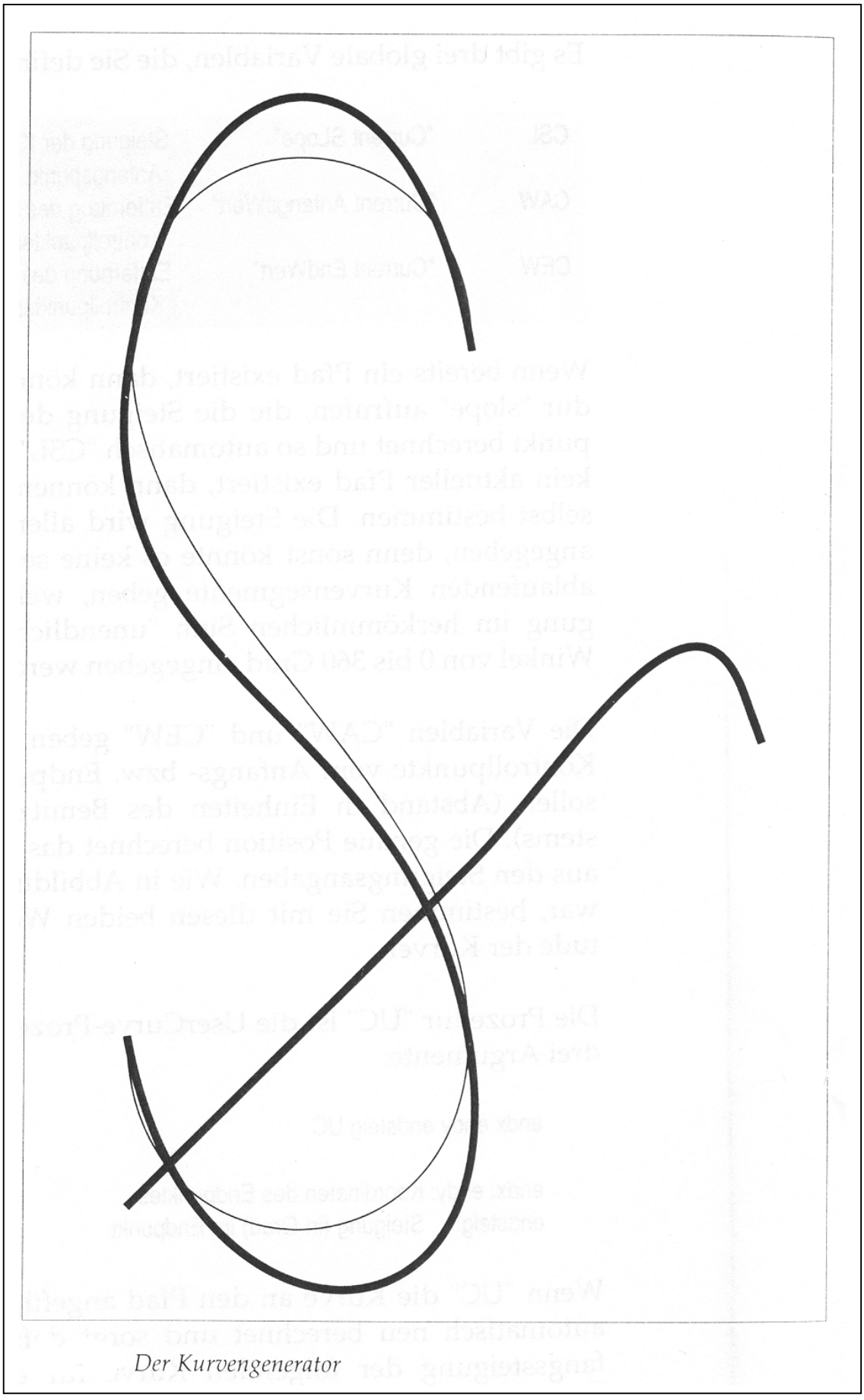
Die Prozedur "UC" ist die UserCurve-Prozedur. Sie verlangt drei Argumente:

```
endx endy endsteig UC
```

endx, endy:	Koordinaten des Endpunktes
endsteig:	Steigung (in Grad) im Endpunkt

Wenn "UC" die Kurve an den Pfad angefügt hat, wird CSL automatisch neu berechnet und sorgt dafür, dass die Anfangssteigung der folgenden Kurve für einen fließenden Übergang geeignet ist.

Die Prozedur "UL" ermöglicht es Ihnen, eine Gerade an eine Kurve anzufügen, wobei der Übergang wieder fließend ist.
UL erwartet nur ein Argument: Die Länge der Linie (in Einheiten des Benutzerkoordinatensystems).



Der Kurvengenerator

(aus Weltner 1991)

Zusammenfassung zu Linienattributen

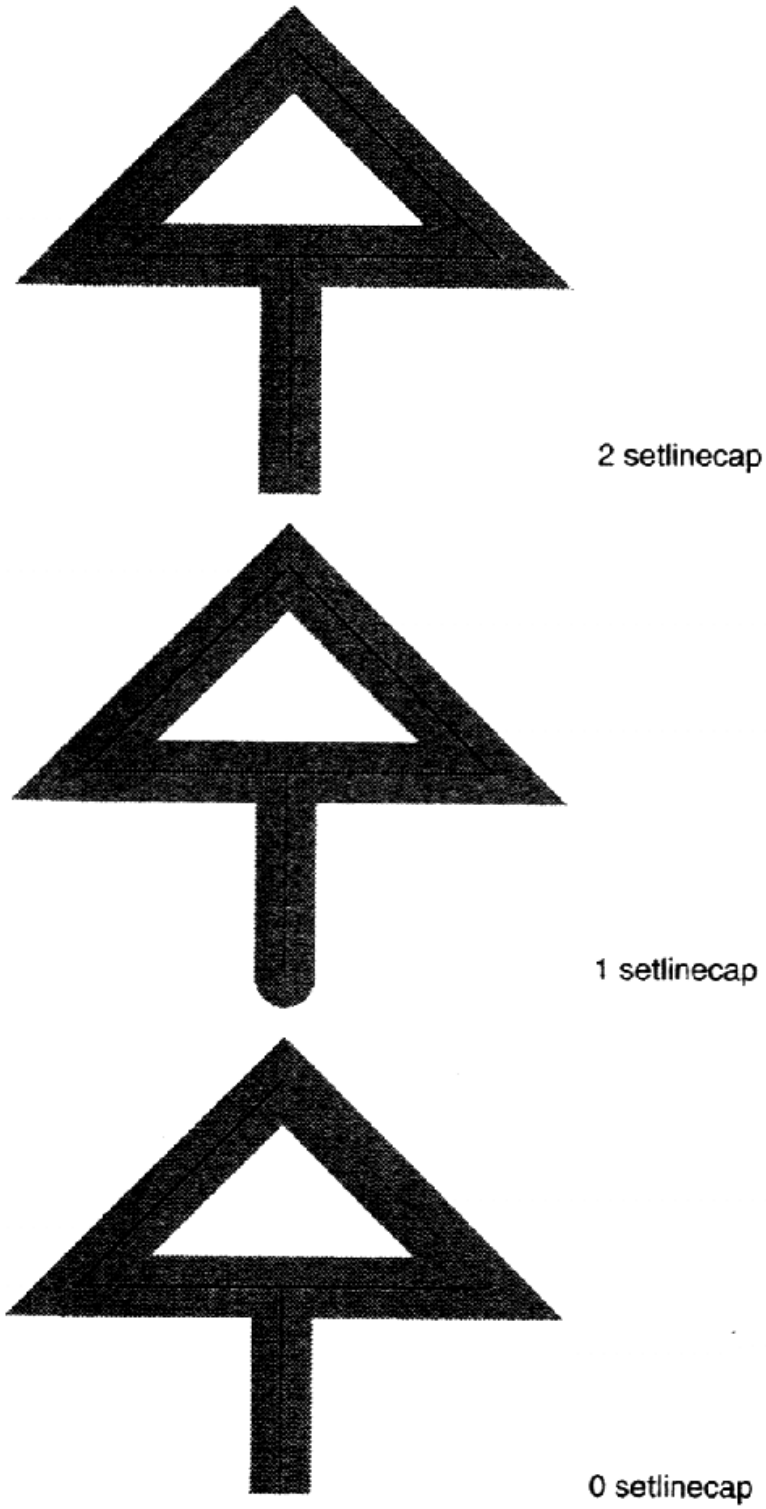
Der Befehl **`stroke`** benutzt Umgebungsvariablen, die vom Benutzer abgefragt und verändert werden können.

Befehl zum Setzen	*Befehl zum Abfragen*	*Wirkung*
`setdash`	**`currentdash`**	Linienmuster
`setflat`	**`currentflat`**	Kurven-genauigkeit
`setgray`	**`currentgray`**	Grauwert
`setlinecap`	**`currentlinecap`**	Form der Enden
`setlinejoin`	**`currentlinejoin`**	Linienverknüpfung
`setlinewidth`	**`currentlinewidth`**	Linienstärke
`setmiterlimit`	**`currentmiterlimit`**	Parameter zum Abschneiden spitzer Ecken

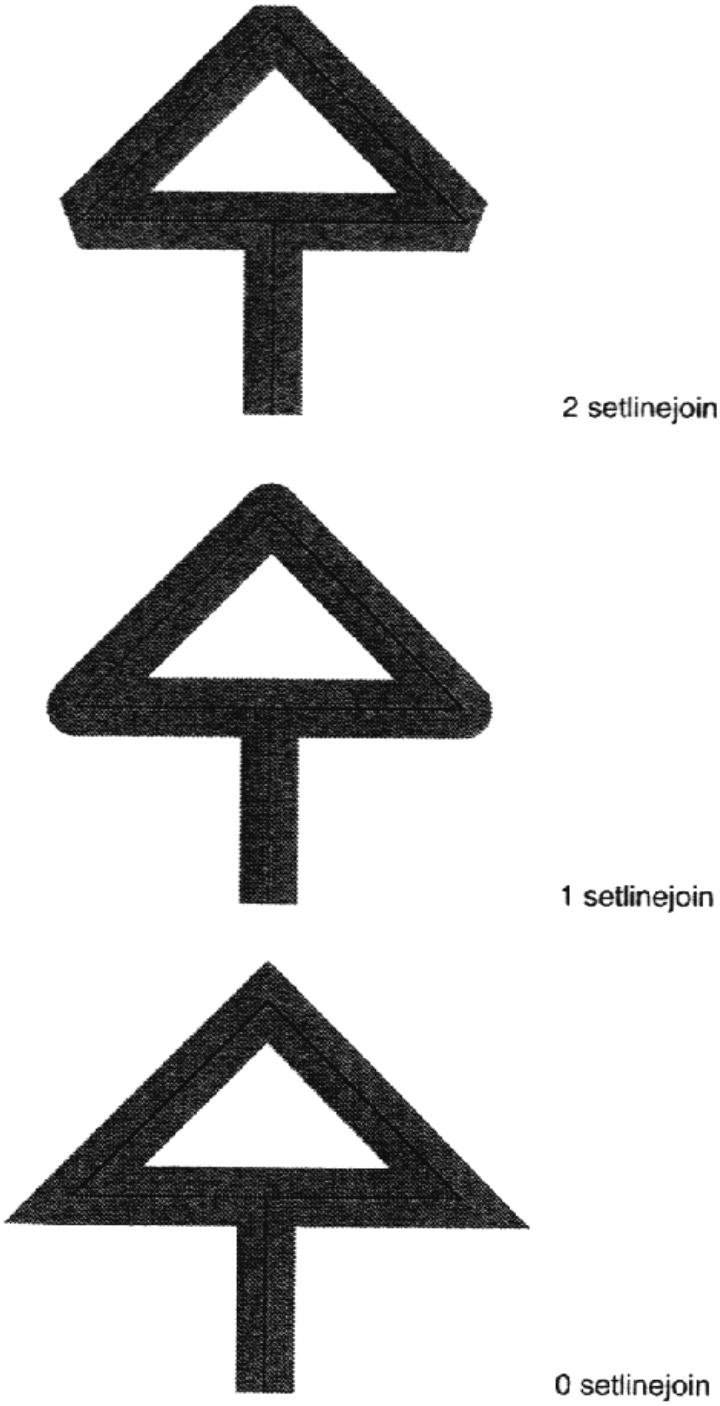
Die bisher noch nicht eingeführten Parameter werden kurz erklärt:

Linienenden (**`setlinecap`**):
3 mögliche Werte:
0 = abgeschnitten (default)
1 = abgerundet
2 = verlängert und abgeschnitten

Beispiel:

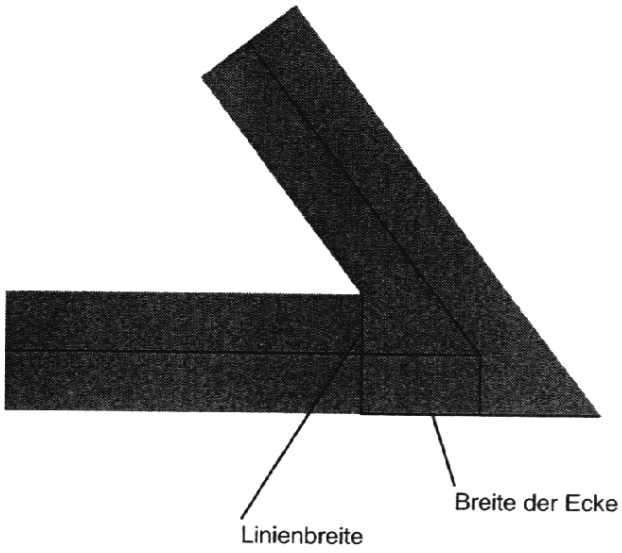


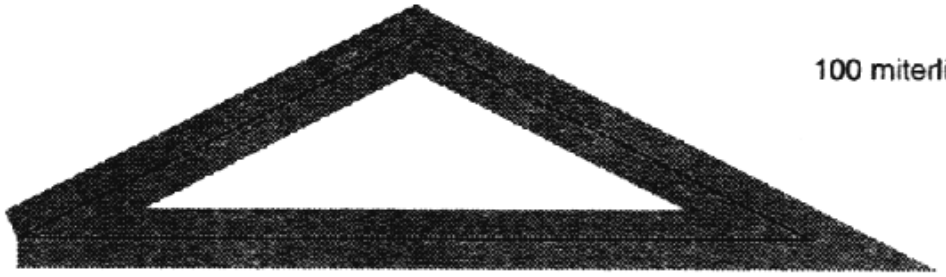
Linienverknüpfungs-Modus (**`setlinejoin`**)
0 = spitze Ecken
1 = abgerundete Ecken
2 = abgekantete Ecken



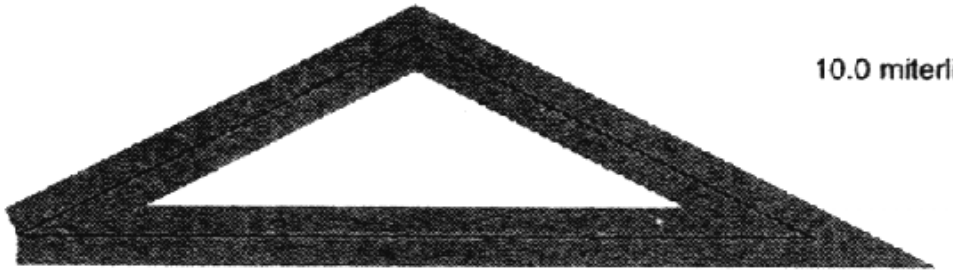
Konditionales Kappen (Abkanten) der Ecken bei spitzen Winkeln:
bei LineJoin-Wert von 0 (spitze Ecken) kann ein Schwellenwert festgelegt werden, der bei spitzen Ecken Kappung (wie bei LineJoin = 2) bewirkt
"miterlimit", Befehl **`setmiterlimit`**
in typografischer Einheit angegeben (Verhältnis zwischen der Länge der spitzen Ecke und der Linienbreite), aber umrechenbar in Grad:
(Grenzwinkel / 2) **`sin`** = 1/miterlimit

Miterwert	*Winkel*	
1	180°	schneidet immer ab
1.414	90°	
2	60°	
2.61	45°	
3.86	30°	
7.66	15°	
11.47	10°	
22.92	5°	





100 miterlimit



10.0 miterlimit

1.0 miterlimit

Annäherungsgenauigkeit für gekrümmte Kurven

Ein weiterer Umgebungsparameter, "flatness", legt fest, mit wievielen Geradensegmenten ein gekrümmtes Linienstück modelliert wird.
Kleine flatness = hohe Genauigkeit.

Einstellung des Wertes: **`setflat`**

Abfrage:

```
/b 10 string def
currentflat
b cvs print
```

Der Defaultwert ist für die meisten Anwendungen ausreichend.