Computergrafik-Übungen
W. Kurth, T.Mangoldt
WS 2003/04

Einführungskurs PostScript, Teil 2

Lösungen der Übungsaufgaben aus Teil 1:

1. Quadratzahlen mit alternierender Farbe untereinander schreiben

```
/Times-Roman findfont 16 scalefont setfont
40 830 moveto                % nach oben gehen
/schreibe                    % Prozedurdeklaration
{
8 string cvs                 % ob. Stackelement in String
                             % konvertieren
dup                          % diesen String doppeln...
stringwidth                  % und durch seine Laenge und
                             % Hoehe ersetzen
pop                          % Hoehe vom Stack entfernen
neg                          % Vorzeichen der Laenge
                             % aendern...
-20 rmoveto                  % und als 1. Argument fuer
                             % rmoveto nehmen
show                         % String ausgeben
} def
1                            % Startwert
20 {                         % Argument fuer repeat
dup dup mul                  % ob. Stackelem. 2 x doppeln,
                             % quadrieren
dup 2 mod 0 eq               % Test, ob gerade Zahl
{ 1 0 0 setrgbcolor } % rot, wenn gerade
{ 0 0 0 setrgbcolor } % sonst schwarz
ifelse
schreibe                     % Quadratzahl ausgeben als String
1 add                        % aktuelle Zahl inkrementieren
} repeat
showpage
```

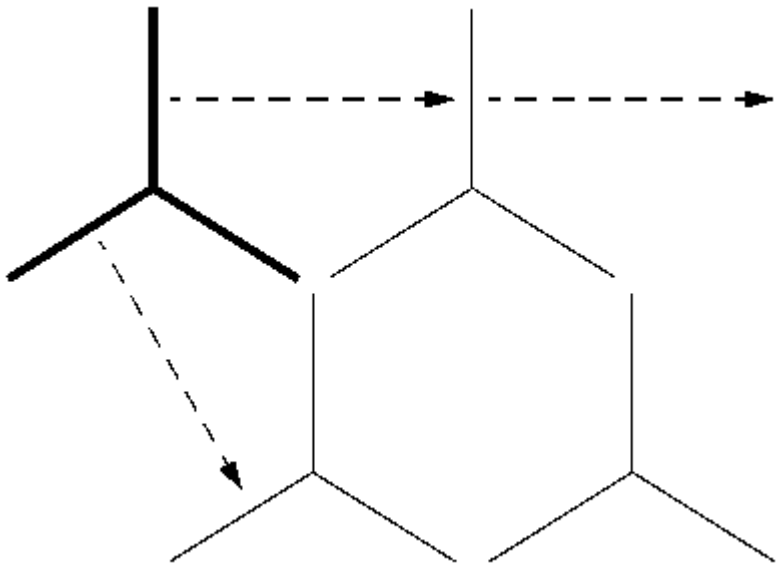
Zweite Variante: mit **for**-Schleife
for stellt Laufvariable bereits auf dem Stack zur Verfügung!
Somit Hauptblock nur noch:

```
1 1 20 {
dup dup mul
dup 2 mod 0 eq
{ 1 0 0 setrgbcolor }
{ 0 0 0 setrgbcolor }
ifelse
```

```
schreibe
} for
```

2. Regelmäßiges hexagonales Gitter

Lösung 1:
"Dreibein" als Elementarzelle, die in 2 Richtungen verschoben und repliziert wird.

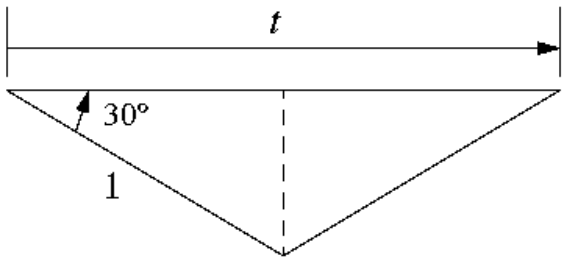


Zur Vereinfachung der "**lineto**"-Befehle verwenden wir die Seitenlänge 1 und vergrößern alles:

```
20 20 scale
0.05 setlinewidth
10 {                          % Faktor für äußeres  repeat
gsave                         % Grafikzustand sichern
10 {                          % Faktor für mittleres  repeat
2 40 moveto                   % Anfangsposition
3 {                           % innerstes repeat
0 1 rlineto
0 -1 rmoveto
120 rotate
} repeat
stroke                        % Dreibein wird gezeichnet
                              % Beginn Verschiebung
-120 rotate                   % Drehung des Koord.syst.
0 1 translate                 % Verschiebung...
60 rotate
0 1 translate
60 rotate
                              % Ende Verschiebung
} repeat                      % zeichne im neuen Koord.syst. dasselbe
grestore                      % alten Ursprung wiederherstellen
```

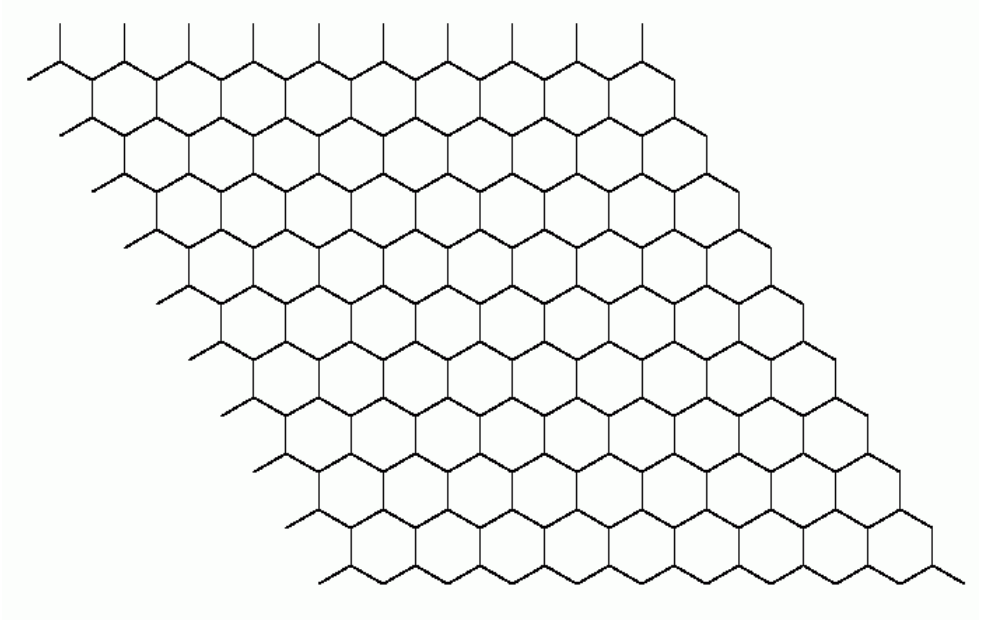
```
-120 rotate                    % wieder Änderung des Koord.syst.:
                               % neue Zeile...
0 1 translate
-60 rotate
0 1 translate
180 rotate
} repeat                       % wieder neu zeichnen im neuen Syst.
showpage
```

Der Teil zwischen "Beginn Verschiebung" und "Ende Verschiebung" kann ersetzt werden durch einen einzigen translate-Befehl, wenn man die Verschiebungslänge ausrechnet:



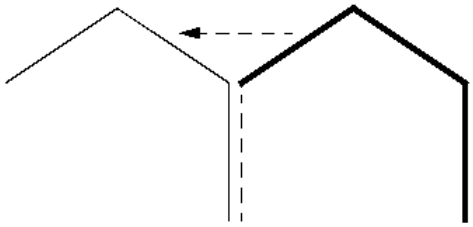
t = 2 cos 30°, also
```
30 cos 2 mul 0 translate
```

Ergebnis des obigen Programms:



Lösung 2:

- andere Elementarzelle



- Streifen als Prozedur definieren

```
20 20 scale
0.05 setlinewidth
/streifen
{
20 20 moveto
10 {
3 {
0 1 rlineto
60 rotate
} repeat
0 1 rmoveto
180 rotate
} repeat
stroke
} def
10 {
gsave
streifen
grestore
30 cos 1.5 translate
} repeat
showpage
```

Kommentar-Konventionen

Einhaltung wichtig beim Ausdrucken. (vgl. Vollenweider 1989)
Erste Zeile in der Datei:
%!PS-Adobe-2.0 EPSF-2.0

Der folgende BoundingBox-Kommentar muss im Dateikopf (also im Block der ersten Kommentare) vorhanden sein:
%%BoundingBox: *ULx ULy ORx ORy*
Untere linke und obere rechte Ecke einer Begrenzungsbox, Angabe im Koordinatensystem des PostScript-Anwenders.
Also z.B.: **%%BoundingBox: 0 0 596 843**

Weitere, empfohlene Kommentare:
%%Title: Dokument-Titel
%%Creator: Ihr Name

%%CreationDate: Datums- und Zeitangabe
%%EndComments Abschluss der Kopf-Kommentare

Font-Namen

Erzeugung einer Liste der verfügbaren Fonts des Druckers (standardmäßig vorgegeben meist 35 Fonts):

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 0 0 596 843
/buffer 100 string def
/x 20 def
/y 750 def
FontDirectory { pop } forall % für jeden Directory-
%Eintrag werden 2 Objekte auf den Stack gelegt, eines wird
% gleich entfernt mit pop
count {                      % für jeden Stack-Eintrag mache...
dup                          % Font-Name wird für findfont und für cvs
                             % gebraucht
findfont
20 scalefont setfont
x y moveto
/y y 20 sub def              % Zuweisung an y
buffer cvs
show
} repeat
showpage
```

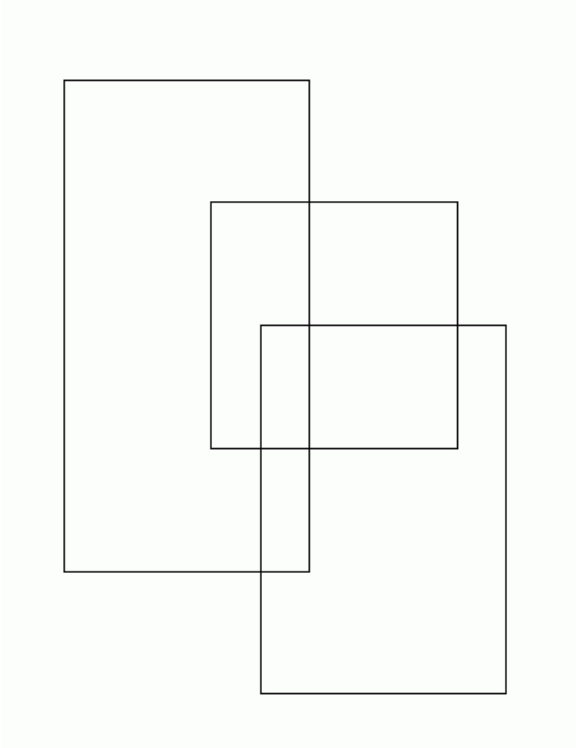
Verwendung anderer Maßeinheiten als pt
durch Definition einer Umrechnungsfunktion: z.B.
/cm { 28.346456 mul } def
Die Verwendung der neuen Maßeinheit wird dann sehr einfach:
5 cm 4.5 cm moveto

Füllregeln

Der aktuelle Pfad kann Selbstüberschneidungen haben und aus mehreren Teilpfaden bestehen. Beispiel:

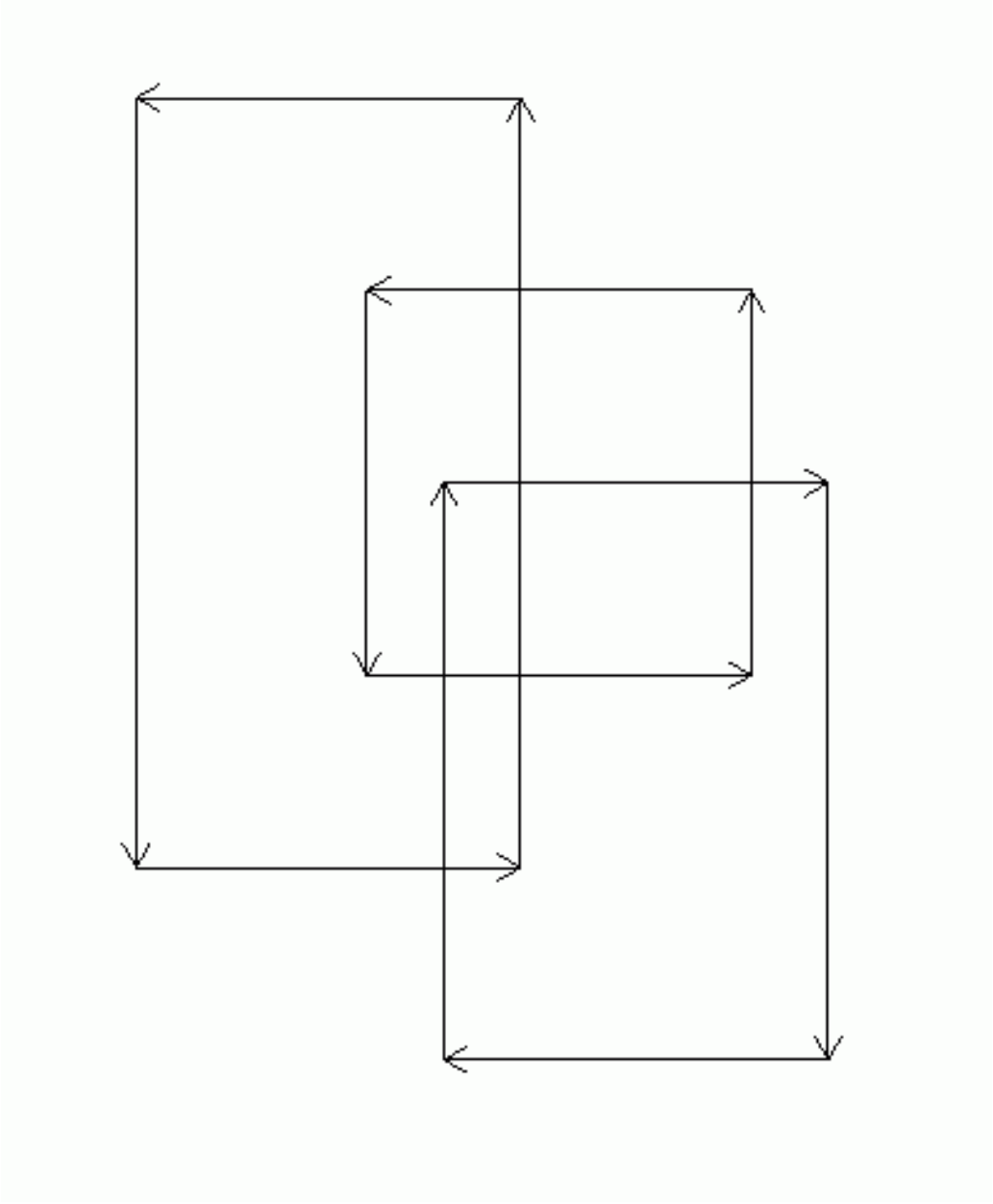
```
/cm { 28.346456 mul } def
5 cm 10 cm moveto
5 cm 0 rlineto
0 10 cm rlineto
-5 cm 0 rlineto
closepath
9 cm 7.5 cm moveto
0 7.5 cm rlineto
5 cm 0 rlineto
0 -7.5 cm rlineto
```

```
closepath
8 cm 12.5 cm moveto
5 cm 0 rlineto
0 5 cm rlineto
-5 cm 0 rlineto
closepath
stroke
showpage
```

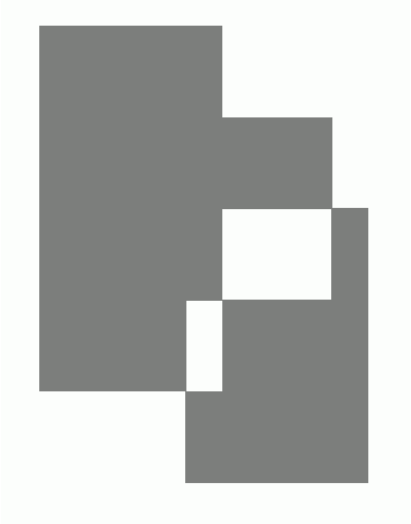


Füllen nach der Non-Zero Winding Rule (Befehl **`fill`**,
d.h. setze **`0.5 setgray fill`** statt **`stroke`** ein):

- waagerechte Test-Strahlen durch alle Flächenteile
- Zähler *z* für jeden Strahl links mit 0 initialisieren
- trifft Strahl auf nach oben gerichteten Kurvenzug: *z*++
- trifft Strahl auf nach unten gerichteten Kurvenzug: *z*--
- fülle alle Flächen, wo *z* ungleich 0 ist



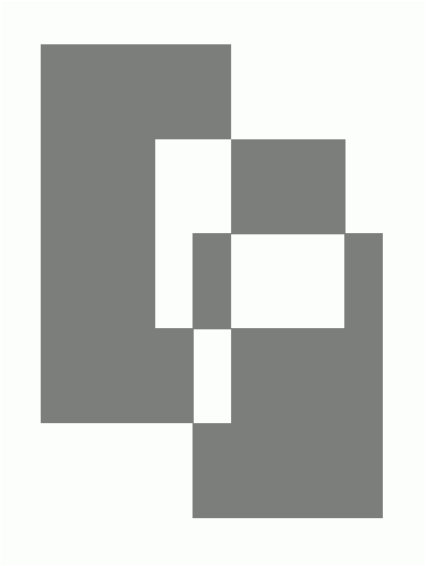
Ergebnis:



Füllen nach der Even-Odd Rule (Befehl **`eofill`** statt **`fill`**):

- *z* wird für jeden Übergang inkrementiert (unabh. von der Richtung),
- fülle alle Flächen, wo *z* ungerade ist.

Ergebnis:



Clipping

Ausblenden von Teilen der Seite durch eine Maske.

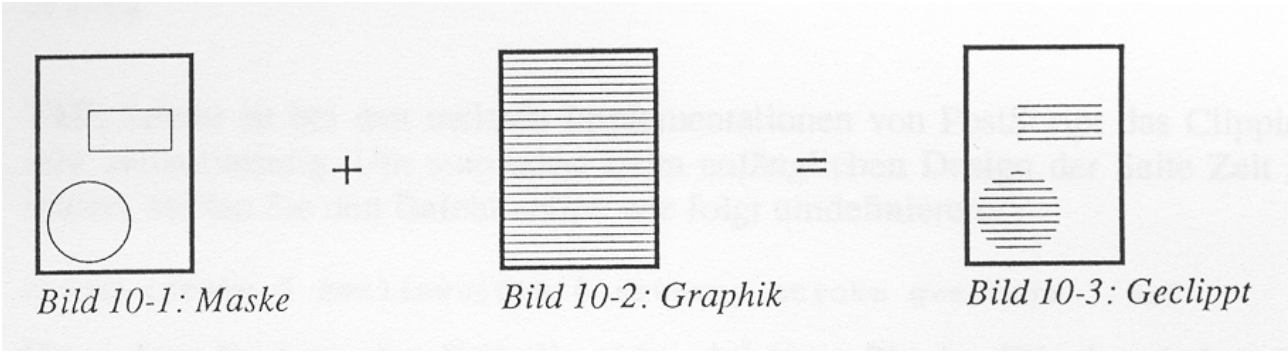


Bild 10-1: Maske *Bild 10-2: Graphik* *Bild 10-3: Geclippt*

Erst Definition der Maske, dann der Grafik, die geclippt werden soll.

Clipping-Befehl hat dauerhafte Wirkung und muss mit **`gsave`** ... **`grestore`** gekapselt werden, wenn danach noch ohne Clipping gezeichnet werden soll!

`clip`	Übernahme des aktuellen Pfades als Clipping-Pfad, danach Zeichnung nur noch dort, wo Färbung bei **`fill`** einträte (*non-zero winding rule*)
`eoclip`	analog, aber mit *even-odd rule*

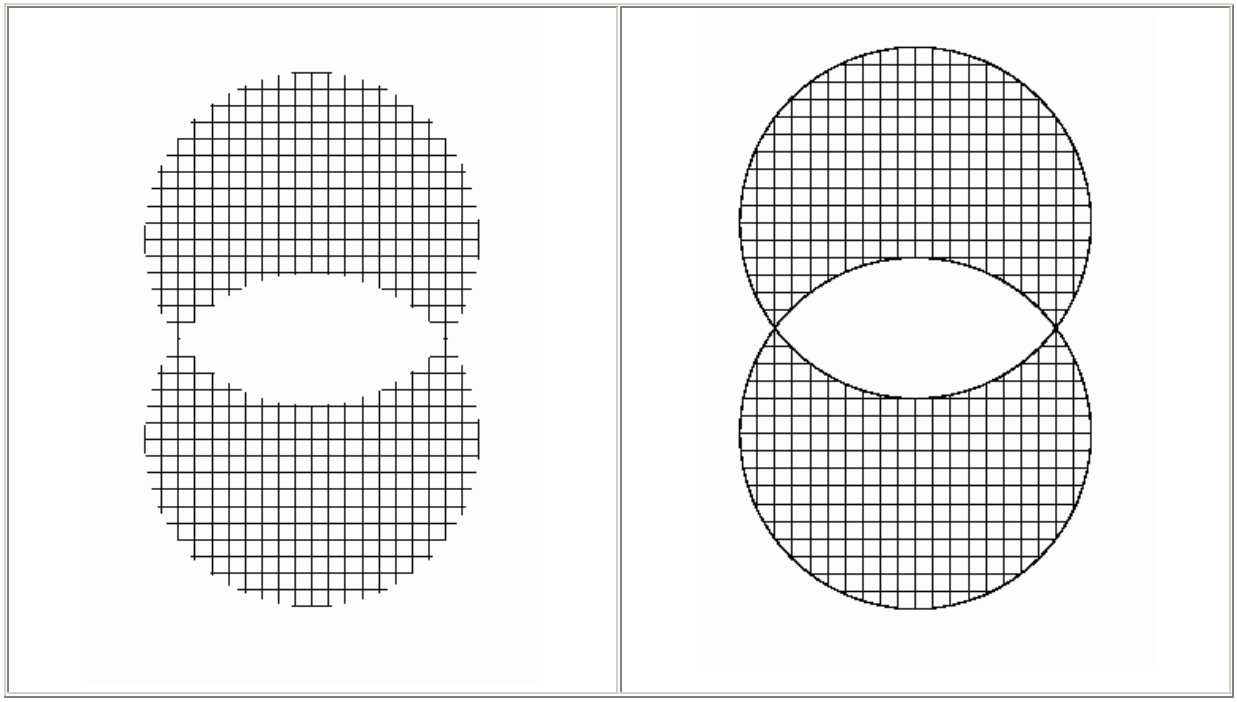
Beachte: Bei beiden Befehlen bleibt der aktuelle Pfad erhalten (und kann z.B. mit **stroke** gezeichnet werden).

Beispiel:

```
/Kreis1
{
150 300 100 0 360 arc
closepath
} def
/Kreis2
{
250 420 moveto
150 420 100 0 360 arc
closepath
} def
/Vertikalen
{
newpath
0 10 500
{
0 moveto
0 800 rlineto
} for
stroke
} def
/Horizontalen
{
newpath
0 10 800
{
0 exch moveto
500 0 rlineto
} for
stroke
} def
Kreis1 Kreis2 eoclip
% stroke
Vertikalen
Horizontalen
showpage
```

Ergebnis:

ohne **stroke**	mit **stroke**



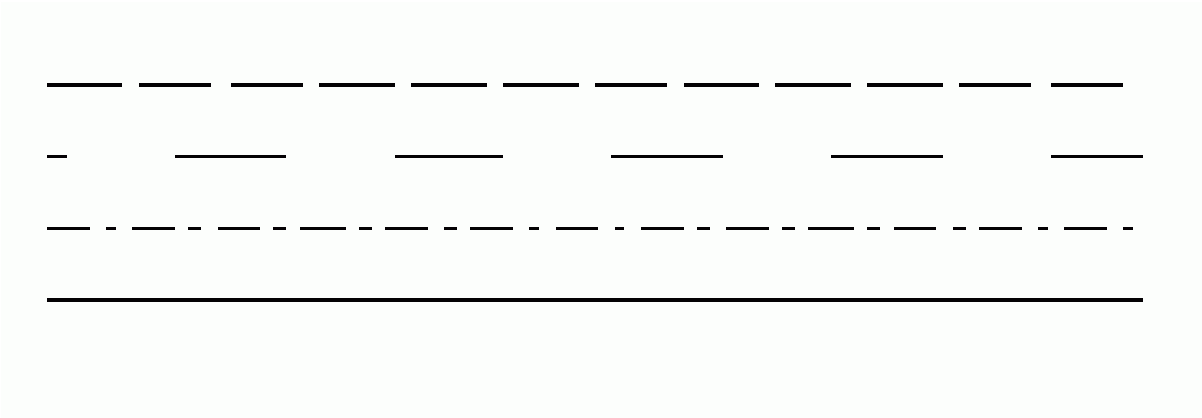
Gestrichelte Linien

Befehl **setdash**
2 Argumente: Array der Strichelungsintervalle, Offset.
Die Array-Einträge werden automatisch periodisch wiederholt.

Beispiel:

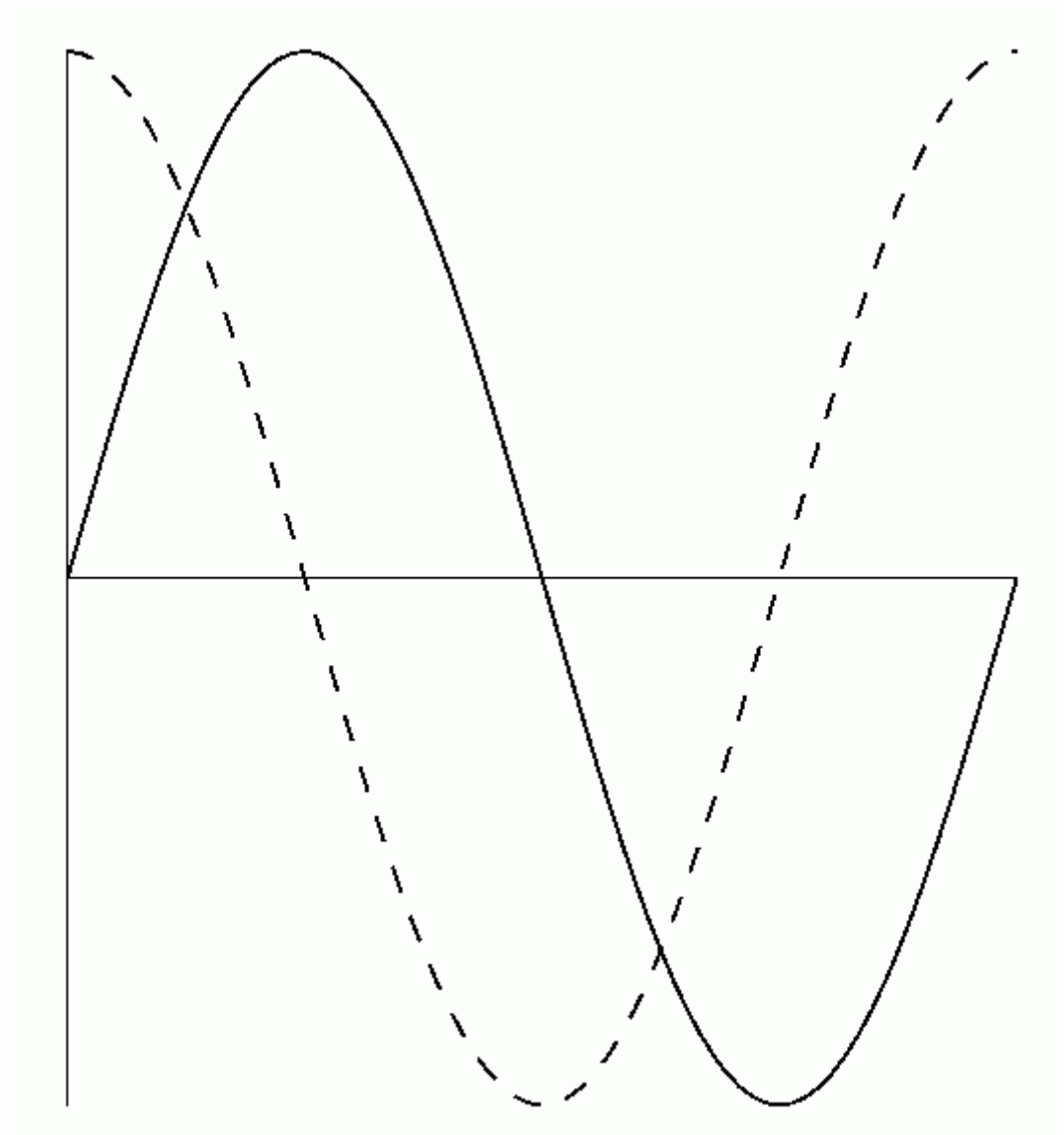
```
/mm { 2.834646 mul } def
/Linie {
300 0 rlineto
stroke
} def
20 800 moveto
[ 20 5 ] 0 setdash Linie
20 780 moveto
[ 30 ] 25 setdash Linie
20 760 moveto
[ 4 mm 1.6 mm 1 mm 1.6 mm ] 0 setdash Linie
20 740 moveto
[] 0 setdash Linie
showpage
```

Ergebnis:



Anwendungsbeispiel: Funktionsverläufe

Sinus- und Kosinusfunktion in einem Diagramm überlagert



```
/Schrittw 5 def
/xtr { 50 add } def
/ytr { 200 mul 400 add } def
/Xachse
{
0 xtr 0 ytr moveto
360 xtr 0 ytr lineto
stroke
} def
/Yachse
{
0 xtr -1 ytr moveto
0 xtr 1 ytr lineto
stroke
} def
/Sinfkt
{
0 xtr 0 sin ytr moveto
0
360 Schrittw idiv
{
Schrittw add
dup dup sin
exch xtr exch ytr lineto
} repeat
pop
stroke
} def
/Cosfkt
{
0 xtr 0 cos ytr moveto
0 360 Schrittw idiv
{
Schrittw add
dup dup cos
exch xtr exch ytr lineto
} repeat
pop
[ 10 ] 0 setdash
stroke
} def
Xachse Yachse Sinfkt Cosfkt
showpage
```

Übergabe von Parametern an Prozeduren

durch
/pa$_n$ **exch def**
...
/pa$_2$ **exch def**

/pa_1 **exch def**
am Anfang der Prozedurdeklaration
dann kann auf pa_1, pa_2, ..., pa_n im Inneren der Prozedur zurückgegriffen werden.

Aufruf: **p1 p2 ... pn Prozedurname**
(**pi**: aktueller Wert des Parameters pa_i).

Beachte: Beim Aufruf wird der Prozedurname ganz oben auf den Stack gelegt. Beim Abarbeiten der Prozedur wird dieser sukzessive mit allen **pi** getauscht, und diese werden dabei an die pa_i übergeben.

Beispiel: Länge eines Vektors.

$$\|(x,y)\| = \sqrt{x^2 + y^2}$$

```
/Times-Roman findfont 20 scalefont setfont
100 700 moveto
/Laenge
{
/y exch def
/x exch def
x dup mul y dup mul add sqrt
} def
3 4 Laenge
10 string cvs show
showpage
```

Anwendungsbeispiele:

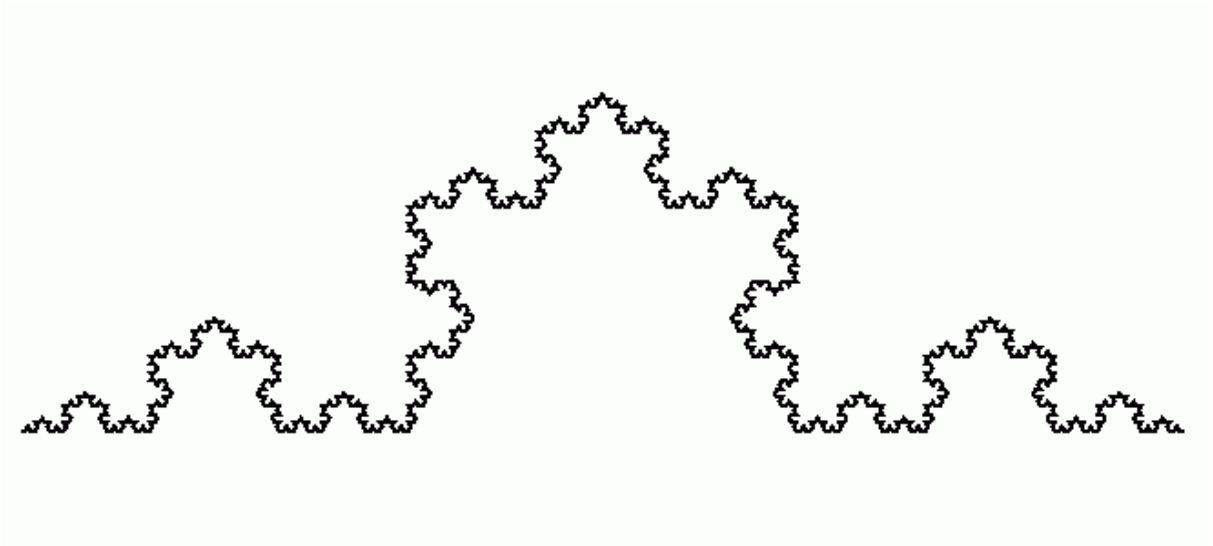
z.B. Prozedur für Pfeile (statt **rlineto**), **rplineto** (selbstdefiniert):

```
/rplineto
{
/y exch def
/x exch def
/laenge
{
/yy exch def
/xx exch def
xx dup mul yy dup mul add sqrt
} def
/sk { x y laenge div 10 mul } def
                                    % Skalierung auf feste Länge 10
x y rlineto
150 rotate
x sk y sk rlineto
```

```
x sk neg y sk neg rmoveto
60 rotate
x sk y sk rlineto
x sk neg y sk neg rmoveto
-210 rotate                    % Ausgangsorientierung wiederherstellen
} def
```

weiteres Anwendungsbeispiel:

Ein einfaches Fraktal
Koch'sche Kurve



Realisierung mit rekursivem Prozeduraufruf:

```
/iterzahl 5 def
/koch
{
/tiefe exch def
tiefe iterzahl lt
{  % solange Iterationszahl noch nicht ausgeschöpft...
1 3 div dup scale
tiefe 1 add
dup dup dup  % Argument wird 4mal gebraucht
koch
60 rotate
koch
-120 rotate
koch
60 rotate
koch
3 3 scale  % alte Skalierung wiederherstellen!
}
{
```

```
400 0 rlineto
} ifelse
} def
50 600 moveto
0 koch
stroke
showpage
```

Last modification: October 22.10.2003