Geschichte

TEX und METAFONT

Donald Knuth schrieb sein Buch „The Art of Computer Programming“. Als die Druckfahnen vom Verleger zurückkamen, ärgerte er sich so sehr über den verkorksten Satz des Buches, dass er sein eigenes Textsatzprogramm TEX schrieb. Weniger bekannt ist das Programm METAFONT, mit dem er passende Schriften dazu entwerfen konnte. Seine Schriften sind sehr stark parametrisiert, das heisst, es gibt globale Parameter, die die Serifenlänge, die x-Höhe, die Kapitalhöhe, die Unterlänge etc. festlegen.

Postscript

Die Firma Adobe, eine Gründung ehemaliger Xerox-Mitarbeiter, entwickelte eine Programmiersprache, die sich insbesondere zur Beschreibung von Gedrucktem eignet.

Vor- und Nachteile

- Vorteil von Postscript und METAFONT

  reichhaltige Befehle für das Zeichnen von Linien

- Vorteil von Postscript

  wird direkt oder indirekt von den meisten Druckern (und Plottern) verstanden

- Vorteil von METAFONT

  parametrische Kurven sind hervorragend darstellbar, Strichstärken sind variabel

- Nachteil von Postscript

  gewöhnungsbedürftige Postfix-Sprache

- Nachteil von METAFONT

  Ausgabe ist geräteabhängige Bitmap

Synthese

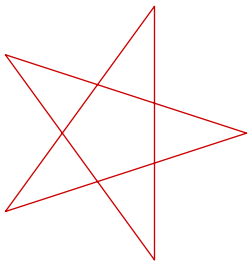
Anfang der 90er Jahre veröffentlicht John Hobby von AT&T Bell Laboratories METAPOST. Die Sprache ähnelt sehr stark METAFONT, hat aber Postscript als Ausgabeformat. Damit sind erzeugte Grafiken an andere Geräte übertragbar, auf denen kein METAFONT installiert ist und die benötigte Auflösung variiert.
Was leider verlorengeht: Die Variation von Strichstärken während eines Strichs.

Eigenschaften von METAPOST

Makrosprache

METAPOST ist eine Makrosprache wie TEX. Es ist sehr einfach, neue Makros zu definieren. Da METAPOST eine Interpretersprache ist, werden Makros quasi zur Laufzeit aufgelöst.

```
draw
  for i=0 upto 4:
    (50, 0) rotated (i*144) --
  endfor
  cycle withcolor .8*red;
```



Wichtige Datentypen

numeric

Das sind Skalare, und zwar Vielfache von 1/65536. Wenn sie als Längenangaben herhalten, sind Postscriptpunkte gemeint, es sei denn, sie sind mit einer Maßeinheit versehen.

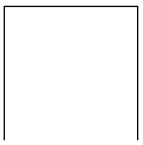
Tupel

Tupel können miteinander addiert und subtrahiert werden und mit Variablen vom Typ `numeric` multipliziert oder durch sie dividiert werden.
Es gibt drei Arten von Tupel: pair, color und transform.

- pair

  wird zur Angabe von Koordinaten verwendet.

```
pair a, b, c, d;
a := (50, 0);
b := (50, 50);
c := (0, 50);
d := (0, 0);
draw a--b--c--d;
```



- color

  gibt eine Farbe als (Rot, Grün, Blau)-Tupel an.

```
for i=0 upto 5:
  fill fullcircle xyscaled 10 shifted (i*15,0)
       withcolor (i/6*red+(1-i/6)*green);
endfor

for i=0 upto 5:
  fill fullcircle xyscaled 10 shifted (i*15,0)
       withcolor (i/6*red+(1-i/6)*green);
endfor
```

- transform

  ist ein Sextupel, das eine zweidimensionale Transfomation des Paars x in y durch $y = Ax+b$ beschreibt (A ist eine 2×2-Matrix und b ein pair). Die Transformationen sind meist einfacher ausgedrückt, und zwar als `shifted`, `rotated`, `xscaled`, `yscaled`, `xyscaled`.
  Transformationen lassen sich transformieren!

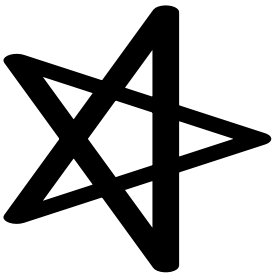
```
for i=0 upto 5:
  fill fullcircle xyscaled 10
       shifted (15i,0)
       withcolor (i/6*red+(1-i/6)*green);
endfor

for i=0 upto 5:
  fill fullcircle xyscaled 10
  shifted (15i,0) rotated 120
  shifted ((90,0) rotated -60)
```

```
    withcolor (i/6*green+(1-i/6)*blue);
  endfor

  for i=0 upto 5:
    fill fullcircle xyscaled 10
    shifted (15i,0) rotated 240
    shifted (90,0)
    withcolor (i/6*blue+(1-i/6)*red);
  endfor
```

Pfade

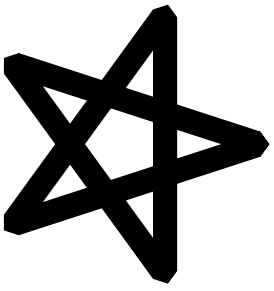
Zwei Arten sind hier zu unterscheiden: path und pen.
Ein pen ist ein geschlossener Pfad, der eine konvexe Figur beschreibt. Damit kann man dann malen.
Im nächsten Beispiel ist der Stift als Ellipse vordefiniert.

```
pickup pencircle xscaled 10 yscaled 5;
draw
for i=0 upto 4:
(50, 0) rotated (i*144) --
endfor
cycle;
```



Im nächsten Beispiel ist der Stift ein Fünfeck, das sehr ähnlich definiert wird wie das Pentagramm.

```
pickup makepen (
for i=0 upto 4:
(5, 0) rotated (i*72) --
endfor
cycle)
draw
for i=0 upto 4:
(50, 0) rotated (i*144) --
endfor
cycle;
```



Pfade näher betrachtet

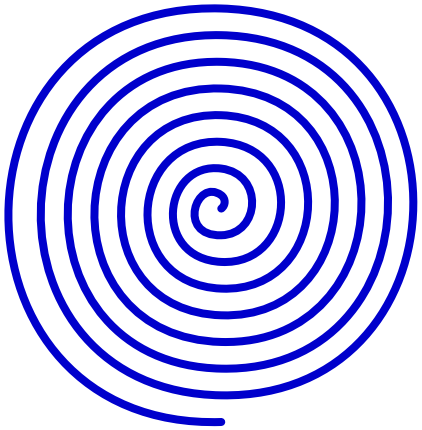
gekrümmte Pfade

Häufig möchte mann nicht nur Polygonzüge, sondern auch krumme Linien malen. METAPOST unterstützt kubische Splines. Ein kubischer Spline über eine Abfolge von Punkten hat folgende Eigenschaften:

- Er berührt alle Punkte.
- Zwischen zwei Punkten ändert sich die Krümmung gleichmäßig.
- Die Krümmung in einem Punkt ist stetig (ändert sich nicht abrupt).

Für einen gekrümmten Pfad werden die beteiligten pairs einfach mit „..“ anstatt mit „–“ verbunden.

```
pickup pencircle scaled 3;
draw
    (0,0)
  for i=1 upto 8:
    ..(0,(i-.5)*10)..(0,-(i+.0)*10)
  endfor
  withcolor .8blue;
```



Zwei oder drei Minuszeichen

Man kann einen geraden Pfad in einen krummen Pfad übergehen lassen.

```
pickup pencircle xscaled 10 yscaled 5;
draw
for i=0 upto 4:
(50, 0) rotated (i*144)
if i<2: .. else: -- fi
endfor
cycle;
```

```
pickup pencircle xscaled 10 yscaled 5;
draw
for i=0 upto 4:
(50, 0) rotated (i*144)
if i<2: .. else: --- fi
endfor
cycle;
```

Pfade lassen sich aufteilen

Man kann einen Pfad in Abschnitte zerlegen. Ein Pfad beginnt bei 0 und geht bis Anzahl der Punkte minus eins. So lassen sich die Pfeile, leicht aufgespreizt, genau parallel zu den Windungen zeichnen (um beispielsweise eine Gebrauchsanweisung für das Abrollen einer Lakritzschnecke zu illustrieren).

```
numeric windungen;
path lakritz, pfeil[];

windungen := 2;
lakritz :=
   for i=windungen step -1 until 1:
    (0,(i+.0)*10){right}..(0,-(i-.5)*10){left}..
  endfor
    (0,0){right}
  for i=1 upto windungen:
    ..(0,(i-.5)*10){left}..(0,-(i*10)){right}
  endfor;

% Der erste und der letzte Abschnitt des Pfades
pfeil.1 := subpath (0.5, 0.1) of lakritz;
pfeil.2 := subpath (4windungen-0.5, 4windungen-0.1) of lakritz;

pickup pencircle scaled 3;
draw lakritz
  withcolor black;
```

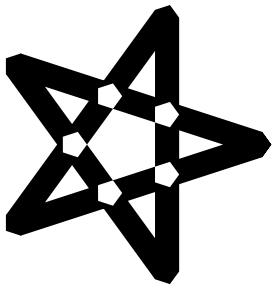
```
% Diese Abschnitte werden leicht aufgespreizt gezeichnet
drawarrow pfeil.1 xyscaled 1.2 withcolor .5white;
drawarrow pfeil.2 xyscaled 1.2 withcolor .5white;
```

An diesem Beispiel sieht man auch, dass man Kurven in bestimmten Punkten eine Richtung aufzwingen kann, in diesem Fall durch left und right.

Überschneidungen

Es lassen sich Überschneidungen von Pfaden finden, auch eine Überschneidung eines Pfades mit sich selbst. Der Befehl dazu ist `p intersectionpoint q`, der ein `pair` zurückgibt. Wenn zwei Pfade einander mehrmals schneiden, wird nur ein Punkt zurückgegeben.

```
path stern;
stern :=
  for i=0 upto 4:
     (50, 0) rotated (i*144) --
  endfor
  cycle;

pickup makepen (
for i=0 upto 4:
(5, 0) rotated (i*72) --
endfor
cycle)

draw stern;

for i=1 upto length stern:
  undraw (subpath(i+1,i+2) of stern) intersectionpoint
         (subpath(i-1,i) of stern);
endfor;
```

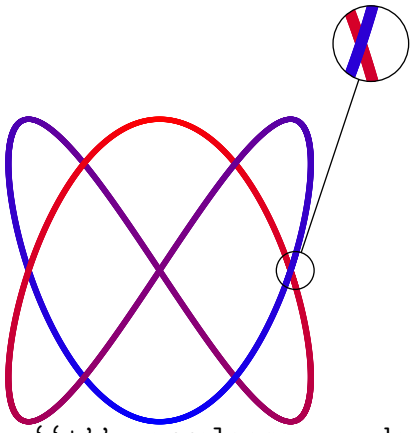


Aber wir waren noch bei den Datentypen

Der Datentyp `boolean` wird ebenfalls verwendet. Er kann `true` und `false` sein und mit `and`, `or` und `not` verknüpft werden.

Datentyp: Bild

Ein weiterer Datentyp ist `picture`, mit dem ganze Bilder gespeichert werden können. Eine wichtige Variable dieses Typs ist `currentpicture`. Beispiel: Ausschnittvergrößerung.

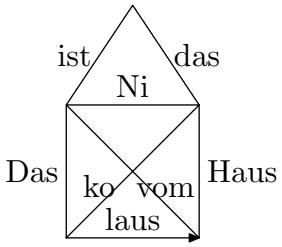


```
picture fl, cl;
numeric i, col, vr, vf;
path vkreis, vkreisg;
pair vp, dp;

for i=0 step .1 until 360:
  col := i/360;
  draw 2cm* (sind(2*i), cosd(3*i))
  withpen pencircle scaled 2bp
  withcolor
    if col>.5: (2*(1-col)) [red, blue]
    else:     (1-2col) [blue, red] % hier kann ‘‘*’’ weggelassen werden
    fi;
endfor;

fl := currentpicture;
vp := (1.8cm,0);
vr := .5cm;
vf := 2;

dp := (1cm, 3cm);

vkreis := fullcircle xscaled vr;
vkreis := vkreis yscaled vr;
vkreis := vkreis shifted vp ;
vkreisg := vkreis shifted -vp scaled vf shifted (vp+dp);
clip currentpicture to vkreis;
cl := currentpicture;

draw fl;
draw vkreis;

draw cl shifted -vp scaled vf shifted (vp+dp);
draw vkreisg;
draw vp--(vp+dp) cutbefore vkreis
     cutafter vkreisg;
```

Anmerkung: Der Kreis muss erst wieder auf die Null zentriert werden, bevor er mit Inhalt vergrößert wird.

Strings

```
path haus;
numeric u;

u:=25pt;
haus := (0u,0u)--(0u,2u)
      --(1u,3.5u)--(2u,2u)
      --(2u,0u)--(0u,2u)--(2u,2u)
      --(0u,0u)--(2u,0u);

drawarrow haus;

label.lft("Das",  point 0.5 of haus);
label.lft("ist",  point 1.5 of haus);
label.rt("das",   point 2.5 of haus);
label.rt("Haus", point 3.5 of haus);
label.top("vom",  point 4.25 of haus);
label.top("Ni",   point 5.5 of haus);
label.top("ko",   point 6.75 of haus);
label.top("laus", point 7.5 of haus);
```

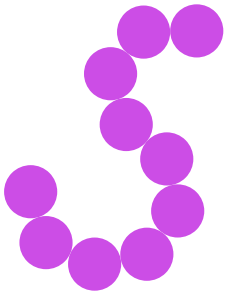


Komplexere Konstrukte

Makros

Makros können selbst definiert werden. Als Argumente können alle Datentypen übergeben werden. Ein Beispiel: Reihe Perlen an eine Schnur. Argumente: Der Radius einer Perle als `numeric` und die Schnur als `path`.

```
def perlenkette(expr r, p) =
path pp;
numeric t;
t := ypart (fullcircle scaled r shifted point 0 of p
            intersectiontimes p);
pp:=p;
if t>-1:
  forever:
    pp := subpath (t, infinity) of pp;
    fill fullcircle scaled r shifted point 0 of pp
    withcolor .8red+.3green+.9blue;
    t := xpart (pp intersectiontimes
    (fullcircle scaled (2*r) shifted point 0 of pp));
    exitif t=-1;
  endfor
fi

enddef;

path faden;
numeric radius, u;

u :=10pt;
radius := 2u;

faden := (0u,0u)..(5u,0u)..(3u,2u)..(7u,4u);

perlenkette(radius,faden);
```



Richtungspunkte

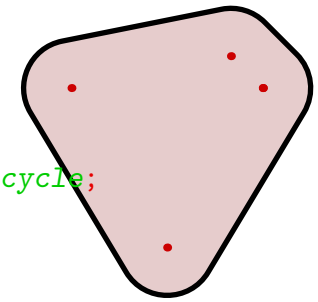
Ein Beispiel zum Zeichnen einer konvexen Hülle über eine Punktmenge verdeutlicht einen Leckerbissen bei METAPOST: Die Fähigkeit, Punkte zu berechnen, an denen ein Pfad eine bestimmte Richtung hat.

```
def encircle(expr pfad, radius) =
  directionpoint ((point 1 of pfad)-(point 0 of pfad))
              of (fullcircle scaled radius shifted (point 0 of pfad))
  for i=1 upto (length pfad)-1:
    ---directionpoint ((point i of pfad)-(point i-1 of pfad))
```

```
                    of (fullcircle scaled radius shifted (point i of pfad))
    ..directionpoint ((point i+1 of pfad)-(point i of pfad))
                    of (fullcircle scaled radius shifted (point i of pfad))
endfor
---directionpoint ((point 0 of pfad)-(point ((length pfad)-1) of pfad))
                of (fullcircle scaled radius shifted (point (length pfad) of pfad))
..cycle

enddef;
```

Dieses Makro gibt einen Pfad zurück, den man zeichnen oder ausfüllen kann.

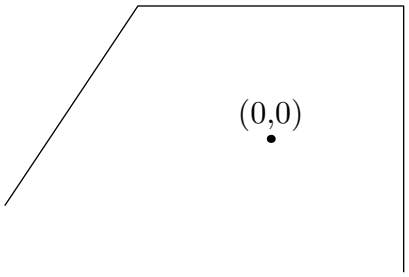
```
numeric u;
path punkte, rand;

u := 12pt;
punkte := (5u,5u)..(4u, 6u)..(-1u, 5u)..(2u, 0u)..cycle;

rand := encircle(punkte, 3u);

pickup pencircle scaled 2pt;
fill rand withcolor .8white+.1red;
draw rand withcolor black;

pickup pencircle scaled 3pt;
for i=0 upto length(punkte):
  draw point i of punkte withcolor .8red;
endfor;
```



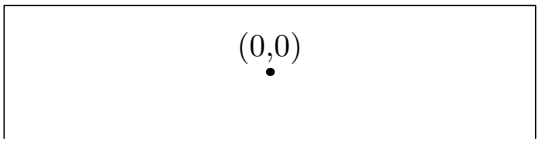
Vorrang: primär, sekundär, tertiär

Wenn man ein Makro mit `def heiopei(expr a)` definiert, wird bei der Auswertung der Makroaufruf durch den Ersetzungstext ersetzt. Manchmal möchte man aber auch Makros auf die Klammerschreibweise verzichten. Dann muss man den Vorrang angeben: Mit welchem Vorrang bindet das Makro seine Argumente?
Das kann man mit den Definitionen `primarydef` und `tertiarydef` einstellen. Im nachfolgenden Beispiel bindet der definierte binäre Operator `squeezed` einmal stärker, einmal schwächer als das „Strichziehmakro“ `--`.

```
primarydef p squeezed s =
  p xscaled s yscaled (1/s) enddef;

numeric u;
pair a, b, c, d;

u := 50pt;
a := (1u, -1u);
b := (1u, 1u);
c := (-1u, 1u);
d := (-1u, -1u);
draw a--b--c--d squeezed 2;

pickup pencircle scaled 3;
draw (0, 0);
label.top ("(0,0)", (0,0));
```



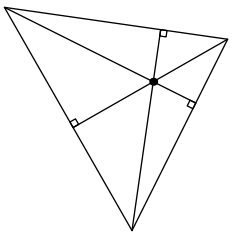

```
tertiarydef p squeezed s =
  p xscaled s yscaled (1/s) enddef;

numeric u;
pair a, b, c, d;

u := 50pt;
a := (1u, -1u);
b := (1u, 1u);
c := (-1u, 1u);
d := (-1u, -1u);
draw a--b--c--d squeezed 2;

pickup pencircle scaled 3;
draw (0, 0);
label.top ("(0,0)", (0,0));
```



Gleichungen

Der Ausdruck `:=` weist zu. Man kann mit `=` lineare Beziehungen zwischen Punkten und Pfaden festlegen. In METAPOST ist `a:=a+1` möglich, `a=a+1` hingegen ein Widerspruch.

Dafür sind die Variablennamen `x<zahl>`, `y<zahl>` und `z<zahl>` praktisch: METAPOST nimmt automatisch an, dass `z4=(x4, y4)` ist.
Beispiel: Finde den Schnittpunkt der Höhen im Dreieck.



```
numeric u;
u=12pt;

z1 = (2u,-4u) ;
z2 = (5u,2u) ;
z3 = (-2u,3u);

% Lot von z1 auf z2--z3
z23 = (z2-z3) rotated 90 shifted z1;

% Lot von z3 auf z1--z2
z12 = (z1-z2) rotated 90 shifted z3;

% Lot von z2 auf z1--z3
z13 = (z1-z3) rotated 90 shifted z2;

% Berechne die Schnittpunkte
z10 = whatever[z1,z23] = whatever[z2,z3];
z30 = whatever[z3,z12] = whatever[z1,z2];
z20 = whatever[z2,z13] = whatever[z1,z3];

%  Schnittpunkt der Hoehen
z0 = whatever[z2,z20] = whatever[z1,z10];

draw z1--z2--z3--cycle;

draw z1--z10;
draw unitsquare scaled .2u rotated angle(z1-z10) shifted z10;

draw z2--z20;
draw unitsquare scaled .2u rotated angle(z2-z20) shifted z20;

draw z3--z30;
draw unitsquare scaled .2u rotated angle(z3-z30) shifted z30;

show x0, y0;
```

```
pickup pencircle scaled 3pt;
draw z0;
```

Die Ausgabe von Metapost ist nun: `>> 31.88026` und `>> 7.9701 [1]`.

Fazit

METAPOST ist ein mächtiges und ausbaufähiges Makropaket zur Erzeugung von Vektorgrafiken.

Literatur und Links:

- Donald Knuth, *The* METAFONT*book,* Addison Wesley, 1986.
- John Hobby, *A User's Manual for MetaPost*, 1991.
- Uwe Siart, Ansprechende technische Illustration mit METAPOST, *Die TEXnische Komödie*, 1/2002.
- Hans Hagen, MetaFun, 2002,
  `http://www.pragma-ade.com`.
- Denis Roegels Seite, mit 3D-Grafik:
  `http://www.loria.fr/~roegel/metapost.html`
- Hans Hagen, Puzzling Graphics with MetaPost,
  `http://www.tug.org/applications/pdftex/metafun.pdf`
- Vincent Zoonekynd, Métapost: exemples,
  `http://www.math.jussieu.fr/~zoonek/LaTeX/Metapost/metapost.html`