

John Warnock 2008, Fotografie von Marvalous, CC-BY-SA-3.0

- ▶ 1982 wurde die kalifornische Firma Adobe Systems von John Warnock und Charles Geschke gegründet.
- ▶ Beide waren zuvor bei XEROX PARC beschäftigt und dort involviert bei der Entwicklung einer Seitenbeschreibungssprache, die (wie so viele weitere wegweisende Projekte bei PARC) nicht den Weg zur erfolgreichen Vermarktung fand.
- ▶ 1985 publizierte Adobe die Programmiersprache PostScript und lizenzierte eine erste Implementierung an Apple, die diese für ihren Apple LaserWriter einsetzte.

- METAFONT ist älter als PostScript (erste Version von 1979, die wesentlich überarbeitete und bis heute gültige Fassung ist von 1984).
- Dennoch sind wohl METAFONT und PostScript weitgehend unabhängig voneinander entwickelt worden, da die Anfänge von PostScript bis in die 70er-Jahre zurückreichen.
- Im Vergleich ist PostScript allgemeiner und mächtiger, aber METAFONT ist in einigen Bereichen (Definition von Kurven, Verwendung von Gleichungssystemen) sehr viel eleganter.

- Anfang der 80er-Jahre kam das Konzept einer Programmiersprache für die Gestaltung einer Seite für viele überraschend.
- Warum soll ausgerechnet der Drucker eine Programmiersprache interpretieren können?
- Die Motivation lag in den vielen Nachteilen der traditionellen Lösungen.

- Schriftsätze in vorgegebenen Größen waren typischerweise in den Drucker integriert oder konnten separat heruntergeladen werden.
- Befehle an den Drucker sahen dann nur die Positionierung vor (eventuell eingeschränkt), die Auswahl eines Schriftsatzes, die Ausgabe von Text und möglicherweise einen Grafik-Modus, bei dem Bitmaps heruntergeladen werden.
- Vorteile: Einfache und schnelle Lösung, kostengünstig.

- Die Portabilität war nicht gegeben, da die Schriftsätze, der Grafik-Modus und die Farbpalette vom Druckermodell abhingen.
- Das Laden von Grafiken in Form von Bitmaps über serielle Leitungen benötigte viel Zeit bei höheren Auflösungen.
- Eine Voransicht auf dem Bildschirm wurde typischerweise nicht unterstützt.

- PostScript wurde zum ersten weit akzeptierten Standard einer Seitenbeschreibungssprache.
- Ein PostScript-Dokument kann auf dem Bildschirm, auf einem Billig-Drucker und einer teuren Offsetdruck-Anlage ausgedruckt werden und im Rahmen der Möglichkeiten sieht sie überall gleich aus.
- PostScript vereinfacht die Herstellung von Grafiken.
- Der Umfang eines Dokuments ist deutlich geringer, wenn Grafiken und besondere Effekte integriert sind.

- Die Drucker mit PostScript waren deutlich teurer, da sie eine leistungsfähigere CPU und mehr Hauptspeicher benötigen.
- Der Zeitaufwand zur Berechnung einer Seite ist nicht nach oben beschränkt.
- Was passiert, wenn bei der Ausführung Fehler auftreten?
- Die Programmiersprache wurde optimiert für eine möglichst einfache Implementierung und nicht in Bezug auf die Freundlichkeit für Programmierer.

- Die Programmiersprache PostScript leitet sich primär von FORTH und Lisp ab.
- Alle Operatoren bzw. Funktionen finden ihre Operanden auf einem Stack und liefern dort ihre Ergebnisse wieder ab.
- Arrays und assoziierte Arrays (*dictionary* genannt) werden unterstützt.
- Programme sind Daten.
- Hinzu kommen Datentypen und Operationen speziell für die Seitengestaltung.

- Zum Experimentieren mit PostScript empfiehlt sich die interaktive Verwendung eines PostScript-Interpreters.
- GhostScript ist eine freie Implementierung von PostScript. Der Aufruf erfolgt hier mit dem Kommando *gs*.
- <https://www.ghostscript.com/>
- Wenn Sie nicht den E.44 nutzen, können Sie sich per „ssh -X“ auf die Theon oder Thales anmelden, um dort *gs* aufzurufen. Die Theon bietet die neueste Version. (Die Optionen „-X“ ist relevant, damit das X-Protokoll von der *ssh* getunnelt wird.

```
theon$ gs
GPL Ghostscript 9.23 (2018-03-21)
Copyright (C) 2018 Artifex Software, Inc. All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.
GS>3 2 sub
GS<1>==
1
GS>quit
theon$
```

- Mit `gs` wurde der Interpreter gestartet, der sich mit dem Prompt „GS>“ meldet.
- Gleich zu Beginn wird auch ein Fenster für die grafische Ausgabe eröffnet.

- Der gesamte Programmtext wird in eine Sequenz lexikalischer Symbole (Tokens) konvertiert, die sofort nach der Erkennung zur Ausführung gebracht werden.
- Tokens fallen in eine von zwei Klassen: Operanden und Operatoren.
- Operanden werden bei der Ausführung auf den Operandenstack geladen.
- Operatoren finden ihre Operanden auf dem Stack und liefert dort auch ihr Ergebnis ab.

- Neben der Ausgabe der eigentlichen Grafik-Seite gibt es auch ganz normale datei-orientierte Ein- und Ausgabeverbindungen in PostScript.
- Dazu gehört insbesondere die Standard-Ausgabe und die Standard-Fehlerausgabe, wobei beide in vielen Fällen identisch sind.
- Der Operator == nimmt das oberste Element vom Stack und gibt dieses in PostScript-Syntax auf der Standard-Ausgabe aus.

```
theon$ gs
GPL Ghostscript 9.23 (2018-03-21)
Copyright (C) 2018 Artifex Software, Inc. All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.
GS>17
GS<1>13
GS<2>12
GS<3>412
GS<4>==
412
GS<3>==
12
GS<2>==
13
GS<1>==
17
GS>quit
theon$
```

- Beim GhostScript-Interpreter verrät der Prompt, wieviele Elemente noch auf dem Operanden-Stack verbleiben.

```
theon$ gs
GPL Ghostscript 9.23 (2018-03-21)
Copyright (C) 2018 Artifex Software, Inc. All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.
GS>2 3 mult
Error: /undefined in mult
Operand stack:
  2  3
Execution stack:
  %interp_exit .runexec2 --nostringval-- --nostringval-- --nostringval-- 2  %stopped_push
Dictionary stack:
  --dict:978/1684(ro)(G)-- --dict:0/20(G)-- --dict:78/200(L)--
Current allocation mode is local
Last OS error: Resource temporarily unavailable
Current file position is 9
GS<2>quit
theon$
```

- In diesem Beispiel wurde versehentlich `mult` anstelle von `mul` für den Multiplikations-Operator angegeben.

- Als Zeichensatz wird ASCII verwendet.
- Neben den druckbaren Zeichen sind nur das Null-Byte, der Tabulator, LF, FF, CR und das Leerzeichen zulässig.
- Alle nicht druckbaren Zeichen dienen als Leerzeichen.
- Aufeinanderfolgende Leerzeichen sind äquivalent zu einem Leerzeichen (abgesehen von Leerzeichen innerhalb von Zeichenketten).
- CR, LF oder CR LF zählt als Zeilentrenner.
- Zu den Sonderzeichen gehören (,), <, >, [,], {, }, / und %.

- Kommentare beginnen mit % und enden mit dem nächsten Zeilentrenner. Sie sind äquivalent zu einem Leerzeichen.
- Zahlen können mit Vorzeichen (+ oder -) angegeben werden. Beispiele:
0 123 -14 +234123
- Es kann auch die Basis der Zahlendarstellung angegeben werden.
Beispiele:
8#644 16#AFB00 2#11011
- Gleitkommazahlen und die Exponentialdarstellung sind zulässig.
Beispiele:
1.2 .123 -3. 1E10 +1.2e-17

- Zeichenketten können in (...) eingeschlossen werden. Beispiel:
(Das ist eine Zeichenkette in PostScript)
- Klammern sind innerhalb der Zeichenkette zulässig, wenn die Klammernpaare balanciert sind. Beispiel:
(Dies ist (ein (Klammergebirge)))
- Eine leere Zeichenkette ist zulässig: ()
- Zeichenketten dürfen über mehrere Zeilen gehen. Beispiel:
(Hier geht die Zeichenkette los,
die einen Zeilentrenner enthaelt)
- Sonderzeichen sind zulässig. Beispiel: (%<>{!})

Sonderzeichen können innerhalb einer Zeichenkette mit Sequenzen, die mit einem Rückwärtsschrägstrich beginnen, eingebunden werden:

Sequenz	Bedeutung
<code>\n</code>	Zeilentrenner (LF)
<code>\r</code>	CR
<code>\t</code>	Tabulator
<code>\b</code>	Backspace
<code>\f</code>	FF
<code>\\</code>	Rückwärtsschrägstrich
<code>\(</code>	öffnende Klammer
<code>\)</code>	schließende Klammer
<code>\ddd</code>	Zeichen in Oktaldarstellung (<i>d</i> : [0-7])

- Namen sind alle Sequenzen von Zeichen, die keine Leerzeichen und keine Sonderzeichen enthalten.
- Namen können mit Ziffern beginnen und gelten als Name, solange sie nicht als Zahlenkonstante interpretiert werden können. Entsprechend ist $1E$ ein Name, während $1E1$ eine Zahl ist.
- Namen können somit ungewöhnlich sein. Beispiele:
 $1+2$ $$$$ $@\$@xX$
- Klein- und Großschreibung ist signifikant.

- Im Normalfall gilt ein Name als auszuführender Operator.
- Steht jedoch unmittelbar vor dem Namen ein Schrägstrich (ohne trennende Leerzeichen), dann wird der Name **nicht** ausgeführt und er gilt als Symbol (analog zu Lisp). Der Schrägstrich selbst gehört jedoch nicht zu dem Namen. Beispiel:
/Hallo
- Diese Unterscheidung ermöglicht es, Operatoren als ganz normale PostScript-Objekte zu behandeln.

- Ein Array kann mit den Sonderzeichen [und] konstruiert werden.

Beispiel:

```
[ 1 2 3 ]
```

- Die Elemente eines Arrays dürfen unterschiedliche Typen haben.

Beispiel:

```
[ 3.14 /Hallo (Hallo!) ]
```

- Arrays können verschachtelt werden. Beispiel:

```
[ (Charles) [ (Philip) [ (Andrew) (Alice) ] (Elizabeth II) ] ]
```

- Wenn ein Operator innerhalb einer Array-Konstruktion vorkommt, wird er sofort ausgeführt.
- Aus diesem Grunde ist
[1 2 1 2 add]
äquivalent zu [1 2 3]

- Prozeduren können mit den Sonderzeichen { und } konstruiert werden.
Beispiel:
`{ dup mul }`
- Prozeduren ähneln den Arrays. Der entscheidende Unterschied ist, dass sie beim Lesen nicht umgehend ausgeführt werden, anschließend aber als ausführbar gelten.
- Grundsätzlich verwaltet der Interpreter für jedes Objekt die Information, ob es ausführbar ist oder nicht.

- Ein assoziatives Array kann mit den Symbolen << und >> konstruiert werden. Dazwischen stehen jeweils Paare aus einem Schlüssel und dem zugehörigen Wert. Beispiel:

```
<< (Vorname) (Hans) (Nachname) (Maier) (Ort) (Ulm) >>
```

- Bei den Schlüsseln sind auch Zahlen zulässig. Zeichenketten und Namen sind bei den Schlüsseln zueinander äquivalent. Obiges Array hätte somit auch so angelegt werden können:

```
<< /Vorname (Hans) /Nachname (Maier) /Ort (Ulm) >>
```


PostScript (und auch METAFONT) unterstützen beim Spezifizieren von Grafiken das Stencil-Paint-Modell:

- Zunächst wird eine Kurve beschrieben, bestehend aus Linien, Kreisbögen, Bézier-Kurven und Zeichen zur Verfügung stehender Schriften.
- Kurven können offen oder geschlossen sein, sie dürfen sich auch selbst kreuzen.
- Kurven müssen nicht zusammenhängend sein.
- Eine Kurve dient dann als Grundlage für Zeichenoperationen.

Wenn eine Kurve fertig definiert ist, stehen folgende Optionen zur Verfügung:

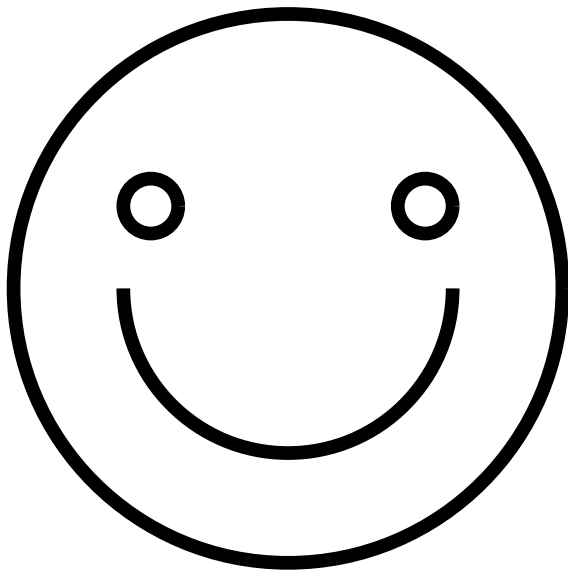
- Die Kurve kann mit einem (ziemlich flexiblen) Stift gezeichnet werden.
- Der Inhalt der Kurve kann mit einer Farbe ausgefüllt werden.
- Die Kurve kann als Schablone (*clipping path*) verwendet werden.

- Für die grafische Ausgabe steht ein Koordinatensystem zur Verfügung, das in Punkten rechnet, wobei ein Punkt typischerweise für $\frac{1}{72}$ Inch steht.
- $(0,0)$ ist erwartungsgemäß links unten.
- PostScript unterscheidet zwischen dem Koordinatensystem im Programm und dem Koordinatensystem des Ausgabegeräts. Beliebige affine Transformationen (Rotieren, Skalieren, Verschieben) sind möglich.

smiley.eps

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -5 -5 205 205
newpath          % eine neue Kurve wird angelegt
100 100 100 0 360 arc % Kreis um (100,100) zeichnen
40 100 moveto    % linke Oberseite des Mundes
100 100 60 180 0 arc % Mund zeichnen
60 130 moveto    % zum linken Auge
50 130 10 0 360 arc % linkes Auge
160 130 moveto   % zum rechten Auge
150 130 10 0 360 arc % rechtes Auge

5 setlinewidth   % Liniendicke definieren
stroke           % Zeichnen
```



smiley.eps

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -5 -5 205 205
```

- Diese Kommentare sind nicht notwendig, um das Beispiel mit `gs smiley.eps` auszuführen.
- Ohne diese Kommentare klappt es jedoch nicht mit `gv` oder dem Einbetten als Grafik in ein Textsystem (wie beispielsweise \LaTeX).



```
newpath          % eine neue Kurve wird angelegt
100 100 100 0 360 arc % Kreis um (100,100) zeichnen
```

- Mit `newpath` beginnt eine neue Kurve (und wirft zuvor die alte Kurve weg). Da zuvor noch keine existierte, hätte das hier auch wegfallen können.
- `arc` erwartet 5 Parameter: Koordinaten des Mittelpunkts, Radius, Winkel, an dem der Kreisbogen beginnt, und der Winkel, an dem der Kreisbogen endet.
- Winkel werden in Grad angegeben. 0 Grad bedeutet genau rechts vom Mittelpunkt, dann geht es im mathematisch positiven Sinne weiter.

smiley.eps

```
40 100 moveto          % linke Oberseite des Mundes  
100 100 60 180 0 arc  % Mund zeichnen
```

- Ein Kreisbogen beginnt implizit mit einer Linie ausgehend vom aktuellen Punkt.
- Um eine unerwünschte Linie zwischen dem Gesicht und dem Mund zu vermeiden, ist eine Positionierung mit `moveto` sinnvoll.
- Alternativ hätten wir auch mit mehreren getrennten Pfaden operieren können.

smiley.eps

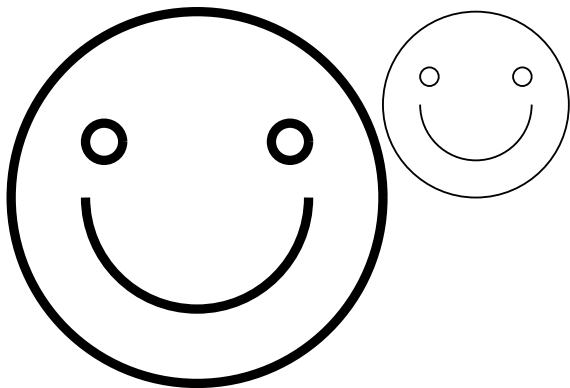
```
5 setlinewidth      % Liniendicke definieren
stroke              % Zeichnen
```

- Zu den vielen Parametern, die den Zeichenstift definieren, gehört auch die Dicke des Stifts, die mit `setlinewidth` verändert werden kann.
- Mit `stroke` geht der Zeichenstift entlang der aktuellen Kurve in Aktion.

smiley2.eps

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -5 -5 305 205
/Smiley {
  newpath          % eine neue Kurve wird angelegt
  100 100 100 0 360 arc % Kreis um (100,100) zeichnen
  40 100 moveto    % linke Oberseite des Mundes
  100 100 60 180 0 arc % Mund zeichnen
  60 130 moveto    % zum linken Auge
  50 130 10 0 360 arc % linkes Auge
  160 130 moveto   % zum rechten Auge
  150 130 10 0 360 arc % rechtes Auge
} def

gsave Smiley 5 setlinewidth stroke grestore
gsave 200 100 translate 0.5 0.5 scale
Smiley 2 setlinewidth stroke grestore
```



smiley2.eps

```
/Smiley {  
  newpath                                % eine neue Kurve wird angelegt  
  100 100 100 0 360 arc % Kreis um (100,100) zeichnen  
  40 100 moveto                          % linke Oberseite des Mundes  
  100 100 60 180 0 arc % Mund zeichnen  
  60 130 moveto                          % zum linken Auge  
  50 130 10 0 360 arc % linkes Auge  
  160 130 moveto                         % zum rechten Auge  
  150 130 10 0 360 arc % rechtes Auge  
} def
```

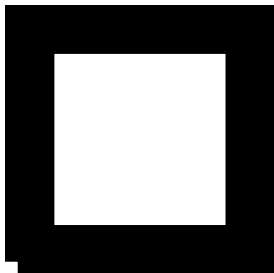
- Nach `/Smiley { ... }` liegen ein Name und ein ausführbares Array auf dem Stack.
- `def` holt sich einen Namen und ein Objekt und trägt das Paar in dem assoziativen Array (*dictionary*) ein, das ganz oben beim Stack der assoziativen Arrays liegt.

- Namensräume werden in PostScript durch assoziative Arrays geschaffen.
- Auf dem Stack der assoziativen Arrays (*dictionary stack*) liegen mehrere solcher Namensräume.
- Wenn für einen Namen das zugehörige Objekt gesucht wird, geht der Interpreter alle Namensräume in dem entsprechenden Stack durch, bis er fündig wird.
- Standardmäßig sind auf diesem Stack `systemdict`, `globaldict` und `userdict`.
- In `systemdict` sind Operatoren wie `add` enthalten, `globaldict` dient als globaler Namensraum und `userdict` wird von `save` und `restore` beeinflusst.

smiley2.eps

```
gsave Smiley 5 setlinewidth stroke grestore
```

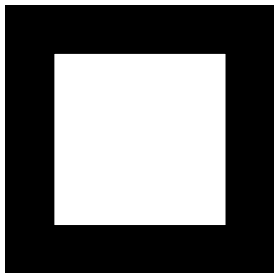
- Zum grafischen Zustand gehören u.a. die aktuelle Position, der aktuelle Pfad, die Konfiguration des Zeichenstifts und die aktuelle Abbildung des Koordinatensystems.
- Mit `gsave` kann der aktuelle grafische Zustand auf einen eigenen Stack gesichert werden.
- Mit `grestore` wird der vorherige grafische Zustand wieder restauriert.



badbox.eps

```
/Box {  
  newpath  
  100 100 moveto  
  200 100 lineto  
  200 200 lineto  
  100 200 lineto  
  100 100 lineto  
} def  
  
Box 30 setlinewidth stroke
```

- ▶ Der Anfang und das Ende eines mit *stroke* gezogenen Striches unterliegt der mit *setlinecap* veränderbaren Konfiguration.
- ▶ In jedem Falle unterscheidet sich das von einem durchgezogenen Pfad.

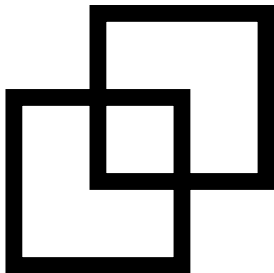


goodbox.eps

```
/Box {  
  newpath  
  100 100 moveto  
  200 100 lineto  
  200 200 lineto  
  100 200 lineto  
  closepath  
} def  
  
Box 30 setlinewidth stroke
```

- ▶ Wenn ein Pfad explizit ringförmig geschlossen werden soll, dann geht dies mit *closepath*.
- ▶ Das Ziehen einer geraden Linie von dem letzten Punkt zu dem Anfang der Kurve erfolgt dabei implizit.

twoboxes.eps



```
/TwoBoxes {  
  newpath  
  100 100 moveto 200 100 lineto  
  200 200 lineto 100 200 lineto  
  closepath  
  150 150 moveto 250 150 lineto  
  250 250 lineto 150 250 lineto  
  closepath  
} def  
  
TwoBoxes 10 setlinewidth stroke
```

- ▶ Ein Pfad kann aus beliebig vielen zusammenhängenden Kurven bestehen.
- ▶ *closepath* schließt dabei nur die letzte Kurve und keinesfalls den gesamten Pfad.



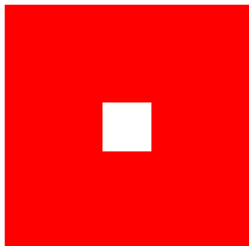
filledboxes.eps

```
/TwoBoxes {  
  newpath  
  100 100 moveto 200 100 lineto  
  200 200 lineto 100 200 lineto  
  closepath  
  100 300 moveto 200 300 lineto  
  200 400 lineto 100 400 lineto  
  closepath  
} def  
  
TwoBoxes 1 0 0 setrgbcolor fill
```

- ▶ Der *fill*-Operator schließt alle möglicherweise noch offenen Kurven eines Pfads und füllt diese dann mit der aktuellen Farbe.
- ▶ Wie bei *stroke* wird danach implizit der *newpath*-Operator aufgerufen.

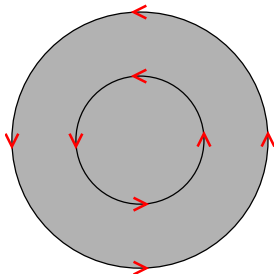
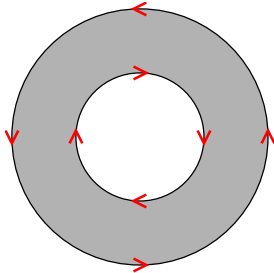


nestedboxes.eps

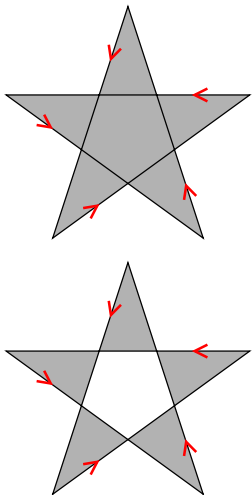


```
/NestedBoxes {  
  newpath  
  100 100 moveto 200 100 lineto  
  200 200 lineto 100 200 lineto  
  closepath  
  140 140 moveto 140 160 lineto  
  160 160 lineto 160 140 lineto  
  closepath  
} def  
  
NestedBoxes 1 0 0 setrgbcolor fill
```

- ▶ In einfachen Fällen erscheint es trivial, was innen ist und somit gefüllt werden soll.
- ▶ Bei mit sich selbst überschneidenden Kurven und Flächen ist es nicht mehr so trivial.



- ▶ Ist ein Punkt p innerhalb oder außerhalb des Pfades?
- ▶ Bei der *Non-zero winding number rule* wird ausgehend von dem Punkt p ein beliebiger Strahl gewählt. Die *winding number* wird zu Beginn auf 0 gesetzt. Wenn dieser Strahl eine Kurve schneidet, die von links nach rechts verläuft, wird die *winding number* um 1 erhöht, bei Kurven von rechts nach links um 1 gesenkt. Wenn all die Schnittpunkte berücksichtigt worden sind und die *winding number* gleich 0 ist, dann ist der Punkt außerhalb, ansonsten innen.
- ▶ Siehe Abschnitt 4.5.2 im *PostScript Language Reference Manual*



- ▶ Bei der etwas einfacheren und sich sehr viel effizienter umsetzbaren *even-odd rule* werden ausgehend von dem Strahl in p in eine beliebige Richtung nur die Schnittpunkte gezählt. Ist die Zahl ungerade, wird der Punkt als innen betrachtet.
- ▶ Der *eofill*-Operator in PostScript arbeitet nach dieser Regel.
- ▶ Das obere Pentagramm wurde mit *fill* gefüllt, das untere mit *eofill*.

- Die Koordinate (x, y) wird abgebildet zu

$$(x, y) \begin{pmatrix} a & b \\ c & d \end{pmatrix} + (t_x, t_y)$$

- In PostScript wird dies zu einem sechs-elementigen Array zusammengefasst: $[a \ b \ c \ d \ t_x \ t_y]$.
- Die Abbildung von (x, y) zu (x', y') lässt sich auch durch eine 3×3 -Matrix darstellen:

$$(x', y', 1) = (x, y, 1) \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{pmatrix}$$

- Operation: t_x t_y translate
- Verschiebung der Koordinaten um (t_x, t_y) .
- Die entsprechende 3×3 Transformationsmatrix:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix}$$

- Operation: s_x s_y scale
- Skalieren der Koordinaten um (s_x, s_y) .
- Die entsprechende 3×3 Transformationsmatrix:

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Operation: θ rotate
- Drehen der Koordinaten um θ .
- Die entsprechende 3×3 Transformationsmatrix:

$$\begin{pmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Zum grafischen Zustand gehört die aktuelle Transformationsmatrix (*CTM*).
- Die Operatoren `translate`, `scale` und `rotate` multiplizieren jeweils die angegebenen 3×3 Transformationsmatrizen mit der *CTM*, um daraus die neue *CTM* zu bestimmen.
- Mit `concat` kann die Matrix auch explizit angegeben werden:

matrix `concat`

Dies entspricht $CTM = matrix \times CTM$

smiley3.eps

```
% x y radius linewidth DrawSmiley -  
/DrawSmiley {  
  4 dict begin  
  /linewidth exch def  
  /radius exch 100 div def  
  /y exch 100 radius mul sub def  
  /x exch 100 radius mul sub def  
  gsave  
  x y translate  
  radius radius scale  
  Smiley  
  linewidth setlinewidth  
  stroke  
  grestore  
  end  
} def  
  
100 100 100 5 DrawSmiley  
250 150 50 2 DrawSmiley
```

smiley3.eps

```
4 dict begin
/linewidth exch def
/radius exch 100 div def
/y exch 100 radius mul sub def
/x exch 100 radius mul sub def
% ...
end
```

- Mit `4 dict` wird ein leeres assoziatives Array mit einer Mindestkapazität von 4 angelegt.
- Mit `begin` wird ein assoziatives Array auf den *dictionary stack* befördert.
- Mit `end` wird das oberste Element des *dictionary stack* abgeräumt.

smiley3.eps

```
4 dict begin
/linewidth exch def
/radius exch 100 div def
/y exch 100 radius mul sub def
/x exch 100 radius mul sub def
% ...
end
```

- Die Parameter müssen in umgekehrter Reihenfolge (LIFO!) abgeholt werden.
- `exch` vertauscht die beiden obersten Elemente auf dem Stack.
- Da Smiley den Mittelpunkt nicht auf $(0,0)$, sondern auf $(100,100)$ setzt mit einem Radius von 100, müssen die Parameter entsprechend angepasst werden.