- APOLLO.OPT
  Optionen für Monitoranzeige auf Apollo-Workstations mit etwas größeren Zeichensätzen.
- P6H.OPT
  Optionen für 24-Nadeldrucker mit 360×360 dpi.
- P6M.OPT
  Optionen für 24-Nadeldrucker mit 360×180 dpi.
- P6L.OPT
  Optionen für 24-Nadeldrucker mit 180×180 dpi.
- FX80.OPT
  Optionen für 9-Nadeldrucker mit 240×216 dpi.

Die Erklärung der Optionen erfolgt im nächsten Abschnitt.

`-f⟨xxx⟩`, `-f⟨xxx:yyy⟩`, `-f⟨xxx:yyy:zzz⟩` Die Formatanweisung wählt die zu bearbeitenden Seiten des Dokuments aus. `xxx` ist die Startseite, `yyy` die letzte Seite und `zzz` gibt die Schrittweite an. Die Parameter entsprechen nicht den TEX-Seitennummern, sondern der Reihenfolge der Seiten in der `DVI`-Datei! Wird die Formatanweisung angegeben, startet `DVI` sofort mit der Bearbeitung der Seiten, ansonsten wird in den interaktiven Bearbeitungsmodus geschaltet und es erscheint der Prompt `DVI>`.

`-o⟨out⟩` Das Ausgabegerät wird durch ⟨`out`⟩ bestimmt. Gültige Angaben sind sind bei der Erklärung der `output`-Option nachzulesen.

`-l⟨log⟩` Der Treiber protokolliert seine Tätigkeiten in der Datei ⟨`log`⟩.

Optionen

Die `DVI`-Optionen werden entweder im interaktiven Modus per Tastatur oder beim Aufruf per Datei mittels `-i⟨name⟩` eingegeben. Bei sichtbarem Prompt `DVI>` kann eine Optionsdatei auch nachträglich durch Eintippen ihres Namens geladen werden. Die Parameter der Optionen sind *dimen* für gültige TEX-Längenangaben wie 1.1 *cm*, 7 *pt*, 0.3 *in*, usw; *mag* für Vergrößerungsfaktoren mit Standard 1000, Faktor 1.2 = 1200, usw; *dpi* für Geräteauflösungen; *path* für Pfade.

hresolution ⟨*dpi*⟩ Horizontale Auflösung in dpi.

vresolution ⟨*dpi*⟩ Vertikale Auflösung in dpi.

pkpath ⟨*path*⟩ Pfad und Name zum Suchen der `PK`-Dateien. Mehrere Angaben werden — je nach Betriebssystem — durch $\boxed{;}$ oder $\boxed{:}$ getrennt. Kein Leerzeichen vor und nach dem $\boxed{;}$! Wie schon einmal bei `DVILW` in Abschnitt 1 erläutert, kann im Zusammenhang mit den Platzhaltern `%d`, `%h`, `%m`, `%s` und `%v` der Name und Pfad der verwendeten Zeichensätze dargestellt werden. Die Platzhalter haben folgende Entsprechungen:

%d	Zeichensatzgröße in dpi
%h	horizontale Geräteauflösung
%m	`MAG␣␣␣␣1.2` etc.
%s	Zeichensatzname, z.B. `cmr10`
%v	vertikale Geräteauflösung

Beispielsweise ergibt also eine Zuweisung wie

```
pkpath=D:\FONTS\%d\%s.pk
```

Zeichensatzpfade und -namen im Stil von

```
D:\FONTS\111\cmr10.pk.
```

tfmpath ⟨*path*⟩ Pfad zum Suchen der `TFM`-Dateien, die gebraucht werden, wenn eine `PK`-Datei nicht vorhanden ist, um wenigstens leere Boxen der richtigen Breite zu erzeugen, bzw. die bei Verwendung virtueller Zeichensätze gebraucht werden.

vfpath ⟨*path*⟩ Pfad zum Suchen der virtuellen Zeichensatzdefinitionen `.VF`.

imgpath ⟨*path*⟩ Pfad zum Suchen der `IMG`-Dateien. Mehrere Angaben werden durch ; oder : getrennt. Kein Leerzeichen vor und nach dem Trennzeichen!

grpath ⟨*path*⟩ Pfad zum Suchen der nachträglich mit `\special{gr input ... }` einzuladenden Grafikdateien.

path ⟨*path*⟩ Pfad zum Suchen der `DVI`-Dateien. Mehrere Angaben werden durch ; getrennt. Kein Leerzeichen vor und nach dem ;!

output ⟨*device*⟩ Gibt Ausgabegerät an. Bisher sind definiert:

- screen
  Monitor mit variabler Auflösung, je nach Computer.
- file
  Datei zur Weiterverarbeitung durch `DISPLAY` (siehe auch Anhang B).
- fx80
  Epson kompatibler 9-Nadler mit 240×216 dpi.
- p6low
  NEC P6 kompatibler 24-Nadler mit 180×180 dpi.
- p6mid
  NEC P6 kompatibler 24-Nadler mit 360×180 dpi.
- p6high
  NEC P6 kompatibler 24-Nadler mit 360×360 dpi.
- hplow
  HP Laserjet kompatibler Laserdrucker mit 100×100 dpi.
- hphigh
  HP Laserjet kompatibler Laserdrucker mit 300×300 dpi.

- bj300
  Canon BJ300 kompatibler Laserdrucker mit 300×300 dpi.
- null
  Null-Device zu Testzwecken (Zeichensätze vorhanden, `DVI`-Datei in Ordnung).

format ⟨*list*⟩ Angabe der zu bearbeitenden Seite(n). Die Anwendung entspricht dem `-f` Schalter, nur daß zwischen den Parametern und der Option ein Leerzeichen stehen muß und daß bei Verwendung ohne Parameter die gesamte `DVI`-Datei bearbeitet wird.

lower, upper Ausgabesteuerung über die internen TEX-Seitenzähler \`count0` bis \`count9`.

copies ⟨*n*⟩ Druckt n Kopien von jeder Seite.

magnification ⟨*mag*⟩ Seitenvergrößerung. Falls mit Null belegt, wird der Wert aus der `DVI`-Datei verwendet.

width ⟨*dimen*⟩ Seitenbreite. Falls mit Null belegt, wird der Eintrag der `DVI`-Datei übernommen.

height ⟨*dimen*⟩ Seitenhöhe. Falls mit Null belegt, wird der Eintrag der `DVI`-Datei übernommen.

hoffset ⟨*dimen*⟩ Seitenverbreiterung links. Negative Werte mit entgegengesetztem Effekt sind möglich. Parameter wirkt additiv auf *hmargin*.

voffset ⟨*dimen*⟩ Seitenverlängerung oben. Negative Werte mit entgegengesetztem Effekt sind möglich. Parameter wirkt additiv auf *vmargin*.

hmargin ⟨*dimen*⟩ Einrückung der Druckseite nach rechts. Negative Werte mit entgegengesetztem Effekt sind möglich.

vmargin ⟨*dimen*⟩ Einrückung der Druckseite nach unten. Negative Werte mit entgegengesetztem Effekt sind möglich.

hspread ⟨*dimen*⟩ Seitenverbreiterung rechts. Negative Werte mit entgegengesetztem Effekt sind möglich.

vspread ⟨*dimen*⟩ Seitenverlängerung unten. Negative Werte mit entgegengesetztem Effekt sind möglich.

eject ⟨*on/off*⟩ Seitenvorschub nach Druckseite an/aus.

page [⟨*n*⟩] Seitenvorschub am Drucker, ggf. ⟨n⟩-fach. Sehr geeignet, um druckerwarmes Toilettenpapier zu produzieren. Für optimale Ergebnisse sollte das Papier aber nicht zu glatt sein.

separate ⟨*on/off*⟩ Steuert Druckqualität. Falls auf `on`, wird die Anzahl der Druckdurchgänge verdoppelt, wobei bei einem Durchgang nur jeweils *nichtbenachbarte* Nadeln drucken.

singlesheet ⟨*on/off*⟩ Umschlaten auf Einzelblattbetrieb mit Wartestellung nach jeder Seite bzw. Endlospapier ohne Warten.

landscape ⟨*on/off*⟩ Drehung des Ausdrucks um 90°. Natürlich darf die Seite nicht zu lang sein und für die Outputroutinen fx80 und p6mid sind spezielle Zeichensätze nötig (216×240 dpi und 180×360 dpi, logisch!).

pictures ⟨*on/off*⟩ Schalter zum Weglassen der `IMG`-Grafiken. An Stelle der Grafiken wird ein leerer Rahmen angezeigt. Dadurch wird der Druckseitenaufbau deutlich schneller!

density ⟨n⟩ Faktor zum Einstellen des „Dunkelwertes“ bei `IMG`-Grafiken, der zwischen 0.0 und 1.0 liegen soll. Ein Wert von $n = 1.0$ entspricht der dunkelsten Darstellung.

thinout ⟨*on/off*⟩ Das „Ausdünnen“der erzeugten Seite wird an-/ausgeschaltet.

port ⟨n⟩ Setzt auf `IBM PC`-Kompatiblen den Ausgabeport. Gültige Werte liegen zwischen 0 und 3, entsprechend den Ports `LPT0`– `LPT3`.

callmf ⟨*name*⟩ Diese Option existiert nur auf Amiga-Implementationen und erhält den Namen eines `ARexx`-Scripts, das einen nichtvorhandenen Zeichensatz evtl. zur Laufzeit generiert. Es stehen die Scripts `MakePkFont.rexx` und `MakePk-FontBatch.rexx` zur Verfügung, die im `REXX:`-Verzeichnis stehen müssen.

memory ⟨*on/off*⟩ Bei eingeschalteter Option wird die gesamte `DVI`-Datei in den Speicher gelesen und dadurch schneller bearbeitet.

showfonts ⟨*on/off*⟩ An- bzw. Ausschalten des Zeichensatzprotokolls.

tracemem ⟨*on/off*⟩ An- bzw. Ausschalten des Speicherprotokolls.

tracechars ⟨*on/off*⟩ An- bzw. Ausschalten des Zeichenprotokolls. Vorsicht, für jedes einzelne verwendete Zeichen wird angegeben, aus welchem Zeichensatz es stammt und welches Zeichen ausgegeben wurde!

maxmem ⟨n⟩ Auf Multitaskingsystemen kann durch Angabe einer maximalen Speichergröße ⟨n⟩ Bytes Platz für andere Applikationen freigehalten werden. Ansonsten nimmt sich `DVI` soviel Speicher, als nötig.

pixmem ⟨n⟩ Für den Aufbau der Textbitmap werden nur ⟨n⟩ Bytes verwendet, anstatt soviel als vorhanden bzw. notwendig. Auf diese Weise kann bei Maschinen mit wenig Speicher bei sehr vielen verwendeten Zeichensätzen durch Beschränkung der Bitmapgröße dennoch ein Ausdruck erzeugt werden, jedoch in mehreren Durchgängen. Mit dieser Variable sollte experimentiert werden, wenn die `DVI`-Meldung „*Out of memory*“ erscheint!

pathmem ⟨n⟩ Mit diesem Befehl kann der Speicherplatz für Clipoperationen der Grafikbefehle festgelegt werden. Evtl. kann die Voreinstellung bei komplizierten Operationen zu niedrig sein bzw. kann dieser Speicher für kritische Texte ohne Grafik zum Aufbau der Textbitmap herangezogen werden.

redirect ⟨*name*⟩ Falls mit dieser Variable ein Dateiname angegeben wird, wird nicht direkt auf dem Drucker gedruckt, sondern in die angegebene Datei. Diese Datei kann dann auf einem anderen Rechner ausgedruckt werden, ohne daß ein TeX-System installiert ist.

dvifile ⟨*name*⟩ Neue `DVI`-Datei *name* bearbeiten, ohne `DVI` zu verlassen.

logfile ⟨*name*⟩ Statistiken, wie gewohnt. Per Voreinstellung ausgeschaltet!

input ⟨*name*⟩, *name* Die Optionsdatei ⟨*name*⟩ bzw. ⟨*name.opt*⟩ wird geladen. Entspricht dem Schalter `-i`.

options, *options* ⟨*name*⟩ Ohne Parameter werden die Optionen mit ihren aktuellen Parametern ausgegeben. Mit Parameter wird eine Optionsdatei ⟨`name`⟩ geschrieben.

exit `DVI` verlassen. Der Druckpuffer von 16 KByte wird allerdings noch fertig ausgedruckt.

Sämtliche Optionen können auch abgekürzt verwendet werden, soweit dies möglich ist. Anstelle von „*format*“ kann auch „*f*“ oder anstelle von „*output*“ kann „*out*“ angegeben werden, usw. Einfach probieren!

Environment

Auf PC-kompatiblen Rechnern kann über die Environmentvariable `GRAFX` eine Anpassung an bestimmte Grafikkarten vorgenommen werden, die der Erkennungsalgorithmus nicht automatisch vornehmen kann bzw. kann man andere Grafikmodi einstellen, als die automatische Erkennung. Die allgemeine Syntax des `GRAFX`-Environments ist wie folgt:

```
SET GRAFX=mode,lines,bytes,code[,fgc,bgc]
```

wobei grundsätzlich die ersten vier Parameter angegeben werden müssen, jeweils durch Komma *ohne zusätzliche Leerzeichen* getrennt. Die Bedeutung der Parameter ist wie folgt:

`mode` Der einzustellende Grafikmodus der Grafikkarte in Dezimal.

`lines` Die Anzahl der Bildschirmzeilen im eingestellten Grafikmodus.

`bytes` Die Anzahl Bytes pro Bitplane einer Bildschirmzeile.

`code` Die Auswahl der Funktion, die das Bitmuster einer TEX-Seite in den Bildschirmspeicher kopiert. Dabei stehen folgende Möglichkeiten zur Auswahl (Großschreibung zwingend vorgeschrieben!):

- LOW
  Bildschirmzeilen mit gerader Zeilennummer sind beginnend bei Segmentadresse `B800` abgespeichert, solche mit ungerader Zeilennummer bei Segmentadresse `BA00`. Dies entspricht dem `CGA`-Grafikmodus 6 bei einer Auflösung von 640×200 Bildschirmpixeln.

- MID
  Die Bildschirmzeilen sind entsprechend einer modulo 4-Rechnung beginnend bei den Segmentadressen `B800`, `BC00`, `A800` bzw. `AC00` abgespeichert, wobei in jeweils zwei aufeinanderfolgenden Bits ein Bildschirmpixel codiert ist, also vier Farben zur Verfügung stehen. Dies entspricht einem speziellen `CGA`-Grafikmodus 9 bei einer Auflösung von 640×400 Bildschirmpixeln, der bei älteren Rechnern vom Typ `NCR PC8` verfügbar ist.

- HIGH
  Die Grafikinformation ist beginnend bei Segmentadresse `A000` in vier Bitplanes abgespeichert. Dies entspricht den hochauflösenden Grafikmodi 16 (`EGA`) bzw. 18 (`VGA`) bei einer Auflösung von 640×350 bzw. 640×480 Bildschirmpixeln. Das Bitmuster wird durch direkte Programmierung des Grafikkontrollers der Karte übertragen. Dieser Modus funktioniert normalerweise auch bei sog. `Super-VGA`-Karten in den 16-Farb-Grafikmodi.

- MONO
  Die Bildschirmzeilen sind entsprechend einer modulo 4-Rechnung beginnend bei den Segmentadressen `B000`, `B200`, `B400` bzw. `B800` abgespeichert, wobei einem Bit jeweils ein Bildschirmpixel entspricht, also zwei Farben zur Verfügung stehen. Dies entspricht dem Grafikmodus einer `Hercules`-Karte bei einer Auflösung von 720×348 Bildschirmpixeln.

`fgc, bgc` Bei den `HIGH`-Modi kann zusätzlich die Farbe der Bildschirmdarstellung gewählt werden. Seit die Treiber Freeware Status haben, ist dies die erste Verbesserung von Benutzerseite. Für die Implementierung und Dokumentation gilt an dieser Stelle Herrn Hans-Joachim Töpfer unserer besonderer Dank! Es bedeuten:

fgc : die Vordergrundfarbe,
bgc: die Hintergrundfarbe.

Die Farben werden entsprechend dem `EGA`-Standard erzeugt. Die Syntax für die Farbeinträge ist folgende:

fgc|bgc::=color
color::=number|{LIGHT|BRIGHT|DARK|secondary_color}primary_color
secondary_color|primary_color::=color_name
color_name::=BLACK|BLUE|GREEN|CYAN|RED|MAGENTA|
BROWN|GRAY|YELLOW|WHITE

Die Farbnummern ergeben sich aus einer 6-stelligen Dualzahl `rgbRGB`. `R`, `G`, bzw. `B` aktivieren die Farben Rot, Grün, bzw. Blau in der Primär-Intensität, `r`, `g`, bzw. `b` in der schwächeren Sekundär-Intensität.

Beim EGA-Standard wird folgende Codierung verwendet:

BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	20
GRAY	7
YELLOW	62
WHITE	63

Bei den Farbnamen im `GRAFX`-Eintrag definiert die primary_color die `RGB`-Bits, die secondary_color die `rgb`-Bits.

Die Zusätze haben folgende Bedeutung:

`DARK`	$primary\_color = BLACK$;
`BRIGHT`	$secondary\_color = primary\_color$;
`LIGHT`	$secondary\_color = WHITE$;

Es gibt z.B.

`Red`	(codiert als 4)
`DarkRed`	(codiert als 32)
`BrightRed`	(codiert als 36)
`LightRed`	(codiert als 60)
`WhiteRed`	(codiert als 60)
`RedWhite`	(codiert als 39)

Bei Grafikkarten mit dem `ET4000`-Chip und 1 MB Bildschirmspeicher funktionieren nun auch die Modi mit 1024×768 und 1280×1024 Pixeln.

Die Farbnamen können der besseren Übersichtlichkeit halber mit großen und kleinen Buchstaben (auch gemischt) geschrieben werden.

Tabelle 2.4 gibt einige Beispiele mit kurzer Erklärung. Alle Fälle, die mit *Standard* bezeichnet werden, sollten durch den Grafikkartenerkennungsalgorithmus automatisch eingestellt werden, wobei jeweils der höchstauflösende Modus verwendet wird. Wer jedoch gerne auf seiner `Super-VGA` mit 640×200 Pixeln im `CGA`-Modus arbeitet bzw. arbeiten muß, weil der Monitor nicht mitspielt, der kann seiner Grafikkarte zwangsweise Schonkost verordnen …

`SET GRAFX=6,200,80,LOW`	Standard `CGA`
`SET GRAFX=16,350,80,HIGH`	Standard `EGA`
`SET GRAFX=18,480,80,HIGH`	Standard `VGA`
`SET GRAFX=135,348,90,MONO`	Standard `Hercules`
`SET GRAFX=?,600,100,HIGH`	`Super-VGA`
`SET GRAFX=41,600,100,HIGH,BLACK,WHITE`	`ET4000` 800×600 Schwarz auf Weiß
`SET GRAFX=55,768,128,HIGH,GREEN,BLACK`	`ET4000` 1024×768 Grün-Monochrom
`SET GRAFX=61,1024,160,HIGH`	`ET4000` 1280×1024

Abbildung 2.4: Beispiele für `GRAFX`-Environments

Beachten Sie bitte, daß bei Verwendung des `GRAFX`-Environments keinerlei Überprüfung der Hardware stattfindet, sondern Sie in eigener Verantwortung handeln. Mit den falschen Parametern können Sie ohne weiteres Ihren Monitor zerstören. Aus diesem Grund müssen Sie eine eventuelle `GRAFX`-Einstellung auch explizit bestätigen! Andererseits können Sie natürlich kleine Unzulänglichkeiten bei der automatischen Grafikkartenerkennung durch Setzen dieser Environmentvariablen ausgleichen. So werden manche `Hercules`-Karten nur bei jedem zweiten `DVI`-Durchlauf als solche erkannt. In diesem Fall sollte man auf `GRAFX` zurückgreifen. Gleiches gilt natürlich für spezielle Grafikmodi exotischer Karten.

2.5 Arbeitsweise von DVI

Nachdem alle Optionen eingegeben wurden, wird die Bearbeitung durch eine „*format*“ bzw. „Start Print“-Anweisung gestartet. Der Speicherbedarf für eine Seite wird berechnet, die Zeichensätze geladen und die erste Seite (evtl. in mehreren Durchgängen) im Speicher aufgebaut und angezeigt/gedruckt.

Bei der Bildschirmanzeige wird evtl. nur ein Ausschnitt der Seite angezeigt. Die nichtsichtbaren Teile können durch Scrollbalken oder der Cursor-Tasten (alle Rechner) in das sichtbare Fenster verschoben werden. Hat man sich an der Bildschirmseite sattgesehen, kann man durch Drücken von N die nächste Seite zur Anzeige bringen (oder den nächsten Durchgang für dieselbe Seite starten) bzw. durch Drücken von Q die Rückkehr in die interaktive Optionseingabe erzwingen. Wird bei der letzten Seite der Formatangabe versucht, eine weitere Seite des Dokumentes zur Anzeige zu bringen, kehrt man trotzdem zur Optionseingabe zurück.

2.6 Besonderheiten

Falls Zeichensätze fehlen, wird eine Datei `missing.fnt` bzw. `MISSING.BAT` zur automatischen Generierung erzeugt, die auf Atari-Rechnern zur TEXshell des Lindner-TEX kompatibel ist und auf `UNIX`-Rechnern sowie AMIGAs ein Shell-Script ist, das nach Eingabe von `./missing.fnt <driver>` auf `UNIX`-Maschinen bzw. `execute missing.fnt <driver>` auf dem AMIGA die Zeichensätze automatisch erzeugt. `<driver>` steht hierbei für den METAFONT-Namen des entsprechenden Ausgabegerätes (`atari_screen`, `OneZeroZero`, `apollo_screen`, etc.). Auf PC-kompatiblen Rechnern werden die Zeichensätze durch den Batchjob über weitere spezielle Batchjobs automatisch erzeugt.

Auf dem Amiga kann alternativ ein in der Konfigurationsvariablen `CALLMF` angegebenes externes `ARexx`-Programm gestartet werden, das dann entweder den Zeichensatz generiert, oder eine entsprechende Batchdatei erzeugt/erweitert. Im Fall von `MakePKFont.rexx` wird versucht, den fehlenden Font direkt zu erzeugen.

Falls Zeichensätze fehlen, erfolgt eine Abfrage, ob die Bearbeitung abgebrochen werden soll, oder ob mit leeren Bitmustern für die fehlenden Buchstaben gearbeitet werden soll.

Falls Sie Zeichensätze mit `missing.fnt` auf `UNIX`-Geräten erzeugt haben, sollten Sie die entstandenen Dateien, die auf `pk` enden, in den entsprechenden Ordner

`/usr/local/lib/tex/pk101` bzw.
`/usr/local/lib/tex/pk120`

kopieren, um diese Zeichensätze permanent zur Verfügung zu haben.

Auf AMIGA-Computern werden die erzeugten Zeichensätze direkt im ersten in der Konfigurationsvariablen `PKPATH` angegebenen Verzeichnis abgelegt.

Die durch METAFONT erzeugten temporären Dateien werden hier automatisch gelöscht.

Auf allen Computern ist es sinnvoll, die Temporärdateien der Zeichensatzerzeugung, nämlich die auf `gf` endenden Dateien und Dateien mit Endung `.log` und `.tfm` zu löschen, falls dies nicht schon, wie auf dem Amiga, geschehen ist.

Natürlich kann jede beliebige Auflösung am Bildschirm dargestellt werden, wenn nur die Zeichensätze vorhanden sind. Wird z.B. `DVI` aufgerufen mit

```
DVI -Ip6h,
```

wird das Programm mit den Standardoptionen für die Druckausgabe an 24-Nadlern mit 360×360 dpi geladen und es erscheint der Prompt `DVI>`. Werden nun als weitere Optionen `out screen`, sowie `hres=360` und `vres=360` angegeben, wird nicht gedruckt, sondern alles in der Auflösung 360×360 dpi am Bildschirm angezeigt. Bei z.B. der `GEM`-Version reicht das Neusetzen des `PK`-Pfades und das Einstellen der Auflösung (H-Resolution/V-Resolution), um dasselbe Ergebnis zu erzielen. Natürlich sind wegen der Größe des Bitmusters wahrscheinlich — abhängig vom Speicherausbau Ihres Computers — mehrere Durchgänge pro Seite erforderlich.

2.7 Installation

Zur Installation müssen lediglich die entsprechenden Dateien — `dvi` bei `UNIX` und Amiga, `dvi.prg` bei Atari ST oder `dvi.exe` bei `MS-DOS` — in einen Ordner kopiert werden, der im Systemsuchpfad enthalten ist und einige Pfade und Optionen richtig eingestellt werden. Will man `DVI` auf dem Amiga nur von der `Workbench` aus benutzen, so muß `DVI` nichteinmal im Pfad stehen. Bei den Gem- und `Motif`-Versionen müssen außerdem alle Optionsdateien `.DVO` gelöscht und neu erzeugt werden, da bei neu hinzugekommenen Optionen die Binärformate der Dateien inkompatibel sind. Als Benutzer merken Sie allerdings von diesen Inkompatibilitäten erst etwas, wenn der Treiber sich seltsam verhält. Bitte Vorsicht! Für die Kommandozeilenersion ist die Klartextdatei `dvi.opt` entsprechend anzupassen.

Für den Amiga wird es in der näheren Zukunft ein Installationsscript für den `Installer` von Commodore geben, mit dem dann die oben beschriebenen Aktionen nicht mehr zu Fuß erledigt werden müssen.

Zum Arbeiten mit reinen Texten ohne Grafik muß der Name und Pfad der Zeichensätze entsprechend der Organisation der Zeichensatzdateien auf der Festplatte eingestellt werden. Bei `UNIX` ist per Voreinstellung

```
./%s.%dpk.:/usr/local/lib/tex/pk%h/%s.%dpk
```

eingebaut, wobei alle Auflösungsstufen eines Zeichensatzes in einem dieser beiden Ordner liegen. Bei Systemen mit 8 Zeichen langen Dateinamen und 3 Zeichen langen Erweiterungen sind die Auflösungsstufen der Zeichensätze naturgemäß nicht im Dateinamen unterzubringen. Man hilft sich dahingehend, daß die Zeichensatznamen nur aus dem Basisnamen bestehen — z.B. `cmr10` — und die Endung `.pk` erhalten. Die Auflösungsstufen der Zeichensätze werden auf eine Ordnerhierarchie abgebildet, wobei die Ordnernamen die Vergrößerungsstufe — z.B. `mag␣␣␣␣1.000` — oder die Auflösung in dpi — z.B. `101` — bezeichnen. Diese Auflösungsstufe errechnet sich zu Gerätegrundauflösung * Vergrößerungsstufe, hier also 101 dpi * `1.000`. Außerdem muß diese Gerätegrundauflösung über die Optionen `hres` und `vres` eingestellt werden.

Sollen schließlich noch Fremdgrafiken eingebunden werden, die sich nicht im aktuellen Arbeitsordner befinden, sind die Pfade `PS` und `IMG` entsprechend zu setzen. Gleiches gilt bei eingebauten Grafikbefehlen, die über externe Dateien mittels `\special{gr input ... }` eingeladen werden.

Kapitel 3. Virtuelle Zeichensätze

Beginnend mit Version 3.0 der Treiber sind `DVI` und `DVILW` in der Lage, virtuelle Zeichensätze zu verarbeiten. Damit steht eine einfach zu bedienende und elegante Möglichkeit zur Verfügung, verschiedene Zeichensatzbeschreibungen mit `DVI`-Treibern zu verwenden, ohne beim Schreiben mit TeX auf spezielle Macros zurückgreifen zu müssen.

Bei virtuellen Zeichensätzen handelt es sich um ein Interface zwischen der Codierung der Zeichen, wie sie TeX handhabt und der Art und Weise, wie das Ausgabegerät dieses Zeichen schließlich darstellt. Im einfachsten Fall handelt es sich also nur um eine Zwischenstufe in der Zeichensatzbearbeitung, die bei Übersetzung der `.dvi`-Datei die Umsetzung des Codes des aktuellen Zeichens in den entsprecheden Code der `.tfm`-Datei vornimmt. In nicht so einfachen Fällen können in der Beschreibung virtueller Zeichensätze auch rekursive Aufrufe von Zeichensatzbeschreibungen, Linienkommandos oder sogar `\special`-Kommandos für den `DVI`-Treiber stehen. Eine Spezifikation der Beschreibungssprache `VPL` kann man in den TeXhax 1990, Issue 11–13 oder in der TUGboat v. 11, no. 1 vom April 1990, Seite 13–23 finden. Die Programme zur Bearbeitung der `VPL`-Sprache — `VFTOVP` und `VPTOVF` — gibt es ja schon seit einiger Zeit mit jeder ordentlichen TeX-Implementierung.

Zur Motivation — nach dem Motto „Brauche ich das denn?“— stellen sie sich folgende Situation vor: Sie haben einen PostScript-Drucker, dazu teuer gekaufte Zeichensätze wie `Garamond` oder `Lucida` und ein TeX-System mit einem Treiber, der *keine* virtuellen Zeichensätze versteht. Im günstigsten Fall hat dann jemand per Hand bzw. mit `PLTOTF` `TFM`-Dateien gestrickt und Sie können Ihren Text mit den kostbaren Zeichensätzen schreiben, solange Sie keine Zeichen wie „ß“ oder „Ä“ verwenden, die von TeX und PostScript unterschiedlich codiert werden. Im Übrigen fehlt Ihnen eine Preview-Möglichkeit, sofern Sie nicht ein Display-PostScript-System zur Verfügung haben. Jeder, der schon einmal mit TeX gearbeitet hat, weiß, wie unbefriedigend und papierfressend eine solche Arbeitsweise wäre ...

Das erste uns bekannte Treiberprogramm, das in diesem Zusammenhang Abhilfe schaffte, war `DVIPS` von Tomas Rokicki. Durch Verwendung virtueller Zeichensätze war jetzt immerhin die Verwendung von Sonderzeichen möglich. Zudem wird mit diesem Treiber ein Programm (`AFM2TFM`) mitgeliefert, das die Zeichensatzmetrikbeschreibung und die Zeichensatzcodierung der PostScript-Zeichensätze in den `.AFM`-Dateien in einen virtuellen Zeichensatz und eine zugehörige `.TFM`-Datei übersetzt. Fehlt also nur noch ein Preview ...

Basierend auf der Arbeit von Rokicki und Knuth wurde für `DVILW` *und* `DVI` die Verwendung virtueller Zeichensätze implementiert. Damit ist ein weiterer Schritt in Richtung Benutzerfreundlichkeit getan, denn jetzt braucht man nur noch entsprechende virtuelle Zeichensätze, die die Umsetzung der PostScript-Zeichensätze auf die Computer Modern Zeichensätze erledigen, um mit Bildschirmpreview und Druckerausgabe mit PostScript-Zeichensätzen zu arbeiten. Es wird übrigens noch ein Freiwilliger für die Anpassung von `AFM2TFM` gesucht!

Da uns momentan die Verwendung von PostScript-Zeichensätzen als sinnvollste Anwendung virtueller Zeichensätze erscheint und Tomas Rokicki auf diesem Gebiet Pionierarbeit geleistet hat, orientiert sich der Rest dieses Abschnittes eng an die Anleitung zu `DVIPS` und `AFM2TFM` bzw. ist sogar eine direkte Übersetzung. Warum das Rad neu erfinden?

3.1 AFM2TFM

Damit `DVI` und `DVILW` mit virtuellen Zeichensätzen arbeiten können, müssen zunächst die zugehörigen `.vf` und ggf. die zugehörigen `.tfm`-Dateien vorhanden sein. Für PostScript-Zeichensätze liegt die benötigte Information in Form von `.afm`-Dateien vor, die mittels `AFM2TFM` in die entsprechenden `.vf` und `.tfm`-Dateien umgesetzt wird. Angenommen, Sie wollen mit dem Palatino-Roman Zeichensatz in TeX arbeiten, dann müssen Sie zunächst die von TeX und `DVI` bzw. `DVILW` benötigten Zeichensatzinformationen mit dem Kommando

```
afm2tfm Palatino-Roman -v pplr rpplr
```

erzeugen. Im folgenden wird übrigens von einem Betriebssystem ausgegangen, das zwischen Klein- und Großschreibung unterscheidet und beliebige Zeichen und Dateinamenslängen verkraftet. Für Systeme mit z.B. 8 + 3-Beschränkungen wie `MS-DOS` lesen Sie bitte Abschnitt 3.2. Die virtuelle Zeichensatzinformation in der `VPL`-Sprache muß dann noch mittels

```
vptovf pplr.vpl pplr.vf pplr.tfm
```

in die für den `DVI`-Treiber verständliche Binärform übersetzt und eine für TeX benötigte `.TFM`-Datei erzeugt werden. Die gerade erzeugten Dateien `pplr.vf`,`pplr.tfm` und `rpplr.tfm` müssen dann noch in die Unterverzeichnisse kopiert werden, wo sie von TeX bzw. `DVI/DVILW` erwartet werden (Je nach Betriebssystem verschieden. Bitte Installationsanleitungen lesen). Jetzt können Sie den Palatino-Roman Zeichensatz in TeX mit

```
\font\myfont=pplr at 11pt
{\myfont Dies ist jetzt die PostScript-Schrift
Palatino-Roman mit Umlauten wie "A, Sonderzeichen
wie "s und Ligaturen!}
```

verwenden und erhalten folgenden Ausdruck:

> Dies ist jetzt die PostScript-Schrift Palatino-Roman
> mit Umlauten wie Ä, Sonderzeichen wie ß und Ligaturen!

Bitte beachten Sie, daß eigentlich zwei Varianten der Palatino-Roman Schrift
→ 3.2 für TEX existieren, nämlich die Eins-zu-Eins-Umsetzung ohne spezielle Satzinformation `rpplr` und der virtuelle Zeichensatz `pplr` mit der Umsetzung für Umlaute, Ligaturen, etc.

Um das Dokument mit diesem Zeichensatz schließlich zu drucken, muß noch in der Datei `DVILW.MAP` ein Eintrag stehen/nachgetragen werden, der die Beziehung zwischen dem PostScript-Namen Palatino-Roman und der Eins-zu-Eins-Font-Metrik `rpplr.tfm` herstellt. Es ist übrigens wichtig, die Eins-zu-Eins-Umsetzung anzugeben, da die Umsetzung der Zeichencodes in der `.dvi`-Datei durch den virtuellen Zeichensatzmechanismus in genau diese Eins-zu-Eins-Umsetzung erfolgt! Die Einträge in `DVILW.MAP` haben folgende Form:

```
rpplr = Palatino-Roman
```

Der erste Eintrag ist der Namen der `.TFM`-Datei der Eins-zu-Eins-Umsetzung ohne Extension, der zweite der Namen des Zeichensatzes, wie er mit dem `findfont`-PostScript-Operator gefunden werden kann (ohne führenden „/“).

3.2 Namenskonventionen für Zeichensätze

Im vorigen Abschnitt haben Sie schon die Datei `DVILW.MAP` kennengelernt. Die beiden Einträge jeder Zeile wurden als Dateiname einer TEX-`TFM`-Datei und als Name eines PostScript-Zeichensatzes vorgestellt. Vielleicht haben Sie sich inzwischen schon über den etwas kryptisch anmutenden Namen des ersten Eintrags gewundert, doch hinter dieser willkürlich anmutenden Buchstabenkombination steht ein ausgklügeltes Namenssystem, das Benutzern antiquierter Betriebssysteme wie `MS-DOS` erlaubt, mit den Beschränkungen der Dateinamenslänge trotzdem eine Fülle von Informationen zu codieren. Mit anderen Worten: Der erste Eintrag ist ein tatsächlicher Dateiname, der auch als Alias für den tatsächlichen (und längeren) PostScript-Namen dient.

Die Initiatoren dieses Namenssystems waren — wieder einmal — Mittelbach und Schöpf, deren Artikel in der TUGboat, v. 11, no. 2 vom Juni 1990 Karl Berry zu der vorliegenden Namenskonvention inspiriert hat.

Die acht Buchstaben des Basisdateinamens sind folgendermaßen aufgeteilt:

```
FTTWVEDD
```

wobei die einzelnen Buchstaben folgende Information codieren (Liste der Abkürzungen für PostScript folgt):

`F` Foundry. Hersteller des Zeichensatzes. Wird ggf. weggelassen.

`TT` Typeface name. Name des Zeichensatzes.

`W` Weight. Die Dicke der einzelnen Zeichen.

`V` Variant. Verschiedene Ausprägungen desselben Zeichensatzes wie z.B. *Italic* oder SMALL CAPS. Die Ausprägung wird weggelassen, falls sie und der folgende Parameter Expansion „normal“ sind.

`E` Expansion. Die Breite der Zeichen. Wird weggelassen, falls „normal“.

`DD` Design size. Punktgröße der Zeichen. Wird weggelassen, falls alle Zeichensatzgrößen durch Skalieren aus einer einzigen `TFM`-Datei hervorgehen.

3.2.1 Foundry

Die Liste der Hersteller von Zeichensätzen:

a	Autologic
b	Bitstream
c	Agfa-Compugraphic
g	Free Software Foundation (g for GNU)
h	Bigelow & Holmes (with apologies to Chuck)
i	International Typeface Corporation
p	Adobe (p for PostScript)
r	reserved for use with virtual fonts; see below
s	Sun

3.2.2 Typeface Families

Die Liste der Zeichensatznamen:

ad	Adobe Garamond	go	Goudy Oldstyle
ag	Avant Garde	gs	Gill Sans
ao	Antique Olive	jo	Joanna
at	American Typewriter	lc	Lucida
bb	Bembo	lt	Lutetia
bd	Bodoni	nc	New Century Schoolbook
bg	Benguiat	op	Optima
bk	Bookman	pl	Palatino
bl	Balloon	pp	Perpetua
bv	Baskerville	rw	Rockwell
bw	Broadway	st	Stone
cb	Cooper Black	sy	Symbol
cl	Cloister	tm	Times
cr	Courier	un	Univers
cn	Century	uy	University
cs	Century Schoolbook	zc	Zapf Chancery
hv	Helvetica	zd	Zapf Dingbats
gm	Garamond		

3.2.3 Weight

Die Liste der Zeichendicken, grob geordnet nach Dicke (zuerst die feinen Schriftdicken):

a	hairline	d	demi
t	thin	s	semi
i	extra light	b	bold
l	light	x	extra bold
k	book	h	heavy
r	regular	c	black
m	medium	u	ultra

3.2.4 Variants

Die Liste der Ausprägungen:

a	alternate	n	informal
b	bright	o	oblique (z.B. slanted)
c	small caps	r	normal (roman oder sans)
e	engraved	s	sans serif
g	grooved (wie beim `IBM` logo)	t	typewriter
h	shadow	u	unslanted italic
i	(text) italic	x	expert
l	outline		

Es gibt einige Ausnahmen bei der Namensbildung mit Ausprägungen. Wenn die Ausprägung „r" und die Breite der Zeichen (Expansion) „normal" ist, dann werden beide weggelassen. Falls die normale Ausprägung einer Schrift serifenlos ist, wie z.B. bei Helvetica, sollte als Ausprägung „n" an Stelle von „s" benutzt werden. Die „alternate" und „expert"-Ausprägungen werden von einigen PostScript-Zeichensätzen mit speziellen Zeichen und zusätzlichen Ligaturen benutzt.

3.2.5 Expansion

Die Liste der Zeichenbreiten, geordnet von schmalen zu breiteren Versionen:

o	extra condensed	x	extended (per Hand)
c	condensed (by hand)	e	expanded (automatisch)
n	narrow (automatisch)	w	wide
r	regular, normal, medium (normalerweise weggelassen)		

Die Zeichenbreiten können in PostScript automatisch, wie z.B. durch den `scale`-Operator, als auch per Hand für jeden Buchstaben einzeln codiert werden. Deshalb die unterschiedlichen Namen.

3.3 Namenskonventionen für virtuelle Zeichensätze

Wenn Sie sich an den vorvorigen Abschnitt erinnern, können Sie sich das Namensschema für virtuelle (PostScript-)Zeichensätze sicher schon denken, denn der Name des virtuellen Zeichensatzes sollte nach Karl Berry's Schema aufgebaut sein.

Bei PostScript-Zeichensätzen kam noch das Problem der verschiedenen Codierung einzelner Zeichen hinzu, das durch Verwendung zweier `.TFM`-Dateien gehandhabt wurde, wobei die Eins-zu-Eins-Codierung ein „r" als

Prefix erhielt, ansonsten aber der Name dem Schema entsprach. Für reine Anwender ist durch die Verwendung der `.TFM`-Datei des virtuellen Zeichensatzes ohnehin keinerlei Abweichung vom Namensschema vorhanden. Es folgt noch eine Liste der Namen der *virtuellen* Zeichensatznamen der gebräuchlichsten PostScript-Zeichensätze:

pagk	AvantGarde-Book	pncri	NewCenturySchlbk-Italic
pagkc	AvantGarde-Book	pncr	NewCenturySchlbk
pagko	AvantGarde-BookOblique	pncrc	NewCenturySchlbk
pagd	AvantGarde-Demi	pplb	Palatino-Bold
pagdo	AvantGarde-DemiOblique	pplbi	Palatino-BoldItalic
pbkd	Bookman-Demi	pplbu	Palatino-BoldUnslanted
pbkdi	Bookman-DemiItalic	pplrrn	Palatino-Narrow
pbkl	Bookman-Light	pplrre	Palatino-Expanded
pbkli	Bookman-LightItalic	pplri	Palatino-Italic
pbklc	Bookman-Light	pplr	Palatino
pcrb	Courier-Bold	pplro	Palatino-Oblique
pcrbo	Courier-BoldOblique	pplru	Palatino-Unslanted
pcrro	Courier-Oblique	pplrc	Palatino
pcrr	Courier	psyr	Symbol
phvb	Helvetica-Bold	psyro	Symbol-Oblique
phvbo	Helvetica-BoldOblique	ptmb	Times-Bold
phvro	Helvetica-Oblique	ptmbi	Times-BoldItalic
phvr	Helvetica	ptmrrn	Times-Narrow
phvrc	Helvetica	ptmrre	Times-Expanded
phvbrn	Helvetica-Narrow-Bold	ptmri	Times-Italic
phvbon	Helvetica-Narrow-BoldOblique	ptmro	Times-Oblique
phvron	Helvetica-Narrow-Oblique	ptmr	Times-Roman
phvrrn	Helvetica-Narrow	ptmrc	Times-Roman
pncb	NewCenturySchlbk-Bold	pzcmi	ZapfChancery-MediumItalic
pncbi	NewCenturySchlbk-BoldItalic	pzdr	ZapfDingbats

Für Kommentare oder Erweiterungen des Namensschemas können Sie sich direkt an Karl Berry wenden:

`karl@cs.umb.edu`

`135 Center Hill Road`
`Plymouth, MA 02360`

Kapitel 4. Die Grafikbefehle von `DVI` und `DVILW`

Als letzter Aspekt der Treiber bleibt noch die Erklärung der Grafikmöglichkeiten. `DVI` und `DVILW` zeichnen sich hier durch besondere Fähigkeiten aus. Sowohl zwei Fremdgrafikformate, als auch ein Satz von eingebauten Befehlen werden unterstützt. Um die Kompatibilität des TEX-Eingabetextes zu wahren, werden alle Grafikbefehle von TEX mittels `\special`-Befehlen angesteuert, auch die eingebauten Befehle.

4.1 Specials

! → Um unnötige Wiederholungen zu vermeiden, wird im folgenden jeweils nur der `\special`-Text beschrieben, obwohl in der Eingabedatei jeweils

`\special{`⟨*befehl*⟩ ⟨*argumente*⟩`}`

stehen muß, um die korrekte TEX-Syntax zu erhalten.

! → Beachten Sie bitte unbedingt die Klein-/Großschreibung der Befehle, da die Treiber bei unbekannten Specials keine Warnung ausgeben und bei eigenen Befehlen Klein-/Großschreibung beachten! Falls Sie also keine Grafikausgabe erhalten, sollten Sie einmal die genaue Syntax beachten.

4.1.1 Fremdgrafikformate

Zwei Fremdformate werden unterstützt. Zum einen die `IMG`-Grafiken von GEM, zum anderen POSTSCRIPT.

graphic img ⟨*filename*⟩

An der aktuellen Seitenposition wird die in der Datei ⟨*filename*⟩ gespeicherte `IMG`-Grafik eingefügt. Die linke obere Ecke der Grafik kommt dabei am sog. „current point“ zu liegen. Der „current point“ wird nicht verändert. Der
→ 4.2 Benutzer hat selbst von TEX aus für Platz für die Grafik zu sorgen.

ps ⟨*filename*⟩

An der aktuellen Position im POSTSCRIPT-Code wird der Inhalt der Datei ⟨*filename*⟩ eingefügt. Naturgemäß funktioniert dieser Befehl nur mit `DVILW`, da `DVI` (vorerst) keinen eingebauten POSTSCRIPT Interpreter hat. Es liegt
→ 4.2 vollkommen in Ihrer Hand, was Sie in die Datei schreiben. Sie sollten aber auf den Aufbau der TEX-Seite achten, d.h. mit `\hbox`- und `\vbox`-Befehlen genügend Platz für die Grafik reservieren. Der Inhalt der Datei wird übrigens in ein `gsave`/`grestore`-Paar eingebunden und der `showpage`-Operator lahmgelegt, so daß der TEX-Teil der Ausgabeseite nicht manipuliert werden kann.

postscript ⟨*commands*⟩

Die PostScript-Befehle aus ⟨*commands*⟩ werden in das PostScript-Programm eingefügt, das die aktuelle Druckseite beschreibt. Sehr geeignet für Unsinn jeglicher Art! Auch dieser Befehl ist nur unter `DVILW` verfügbar. Ein Anwendungsbeispiel für diesen Befehl ist folgender TEX/LATEX-Code:

```
\vbox to 100bp{\vss % a bp is the same as a PostScript point
\special{postscript newpath 0 0 moveto 100 100 lineto
354 0 lineto closepath gsave 0.8 setgray fill grestore
stroke}}
```

Das Ergebnis ist Abbildung 4.1. Das Beispiel stammt übrigens aus der Anleitung von `DVIPS` von Tomas Rokicki. Die Befehle werden übrigens zur Sicherheit in ein `gsave`/`grestore`-Paar eingebunden.

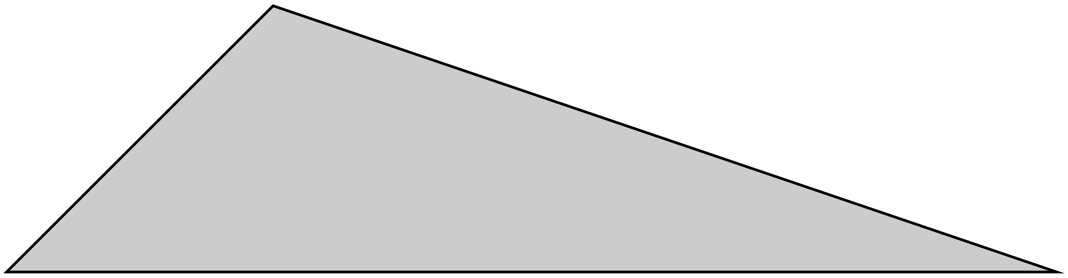


Abbildung 4.1: Anwendungsbeispiel für das *postscript*-Special

¨ ⟨*commands*⟩

Bedeutung wie *postscript* ⟨*commands*⟩ und nur aus Kompatibilitätsgründen zu `DVIPS` von Tomas Rokicki vorhanden.

ps: ⟨*commands*⟩

Die PostScript-Befehle aus ⟨*commands*⟩ werden in das PostScript-Programm eingefügt, das die aktuelle Druckseite beschreibt. Sehr geeignet für Unsinn jeglicher Art! Auch dieser Befehl ist nur unter `DVILW` verfügbar. Die PostScript-Befehle werden übrigens *nicht* zur Sicherheit in `gsave` und `grestore` eingebunden, so daß Sie wirklich Unsinn anstellen können. Die Anzahl der Doppelpunkte im *ps:*-Kommando ist übrigens — wiederum aus Kompatibilitätsgründen — freigestellt.

In Bild 4.2 folgt noch ein Beispiel für dieses Special, das wiederum der Anleitung von `DVIPS` von Tomas Rokicki entnommen wurde. Wie im Original wird auch hier auf jeglichen Kommentar verzichtet!

```
\def\rotninety{\special{ps:currentpoint currentpoint
translate 90 rotate neg exch neg exch translate}}
\font\huge=cmbx10 at 14.4truept
\setbox0=\hbox to0pt{\huge A\hss}\vskip16truept
\centerline{\copy0\special{ps:gsave}\rotninety\copy0
\rotninety\copy0\rotninety\box0\special{ps:grestore}}
\vskip16truept
```

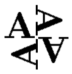


Abbildung 4.2: Spaß mit PostScript

PSFile=⟨filename⟩ llx=⟨llx⟩ lly=⟨lly⟩ urx=⟨urx⟩ ury=⟨ury⟩ rwi=⟨rwi⟩

Ähnlich dem *ps*-Special wird an der aktuellen Position im PostScript-Code der Inhalt der Datei ⟨*filename*⟩ eingefügt. Allerdings muß über eine Reihe weiterer Parameter die Bounding Box des Bildes und die tatsächliche Breite des Bildes in PostScript-Punkten angegeben werden. Aus diesen Angaben wird dann das Bild richtig skaliert und positioniert. Direkte Verwendung dieses Special erscheint nicht sinnvoll, da über die `EPSF.STY`-Macros für TeX und LaTeX die Verwendung automatisiert ist. Lesen Sie hierfür bitte Abschnitt 4.2.3.

Naturgemäß funktioniert das *PSFile*-Special nur mit `DVILW`, da `DVI` (vorerst) keinen eingebauten PostScript Interpreter hat.

4.1.2 Eingebaute Grafikbefehle

Die eingebauten Grafikbefehle werden alle durch `gr` eingeleitet. Danach kann eine Liste von verschiedenen Grafikbefehlen kommen, die jeweils durch $\boxed{;}$ getrennt werden müssen. Natürlich kann man auch jeden Befehl einzeln mit `\special{gr ...}` abschicken.

! → Die direkte Angabe der `\special`-Befehle, so wie im Tutorial aus didaktischen Gründen geschehen, führt zu Problemen mit dem `\put`-Befehl (siehe dazu Abschnitt 4.2.2). Es wird dringendst empfohlen, die `BILDMAC`-Macros zu verwenden!

! → Nachfolgend erklärte Befehle erwarten meistens Parameter eines bestimmten Typs. Es werden die Parameterbezeichnungen *real* für Gleitpunktzahlen und *dimen* für Längenangaben in TeX-ähnlicher-Syntax (wie z.B. -5.33 *cm* oder 5.1 *pt*) bzw. ebenfalls Gleitpunkzahlen verwendet. Im Unterschied zu TeX benötigen die Treiber zum korrekten Erkennen der Bemaßungseinheiten Leerzeichen zwischen syntaktischen Einheiten. So ist z.B. in TeX die Angabe von `1 truecm` zulässig, muß aber in der Treibersyntax als `1 true cm` geschrieben werden. Andererseits verstehen die Treiber als Zahlenwerte bei Längenangaben Ausdrücke aus den vier Grundrechenarten und Klammern, während TeX wirklich eine Zahl erwartet.

! → Alle Einstellungen der Grafikparameter sind nur auf der Druckseite gültig, auf der sie verändert wurden. Durch einen Seitenumbruch werden alle Parameter wie durch ein `setdefaults`-Kommando zurückgesetzt.

%

Wie in TeX dient das %-Zeichen dazu, den Rest der Eingabezeile als Kommentar zu markieren.

def

Das Schlüsselwort `def` dient zum Definieren von Variablen und Funktionen. Die Syntax einer Variablendefinition/-änderung ist wie folgt:

```
def ⟨identifier⟩=⟨expression⟩;
```

wobei `identifier` ein beliebiger Name ist. Falls der Name schon vorhanden sein sollte, wird der bisherige Wert durch die neue Definition überschrieben.

Eine Funktionsdefinition sieht wie folgt aus:

```
def ⟨identifier⟩(⟨p1⟩,...,⟨pn⟩)=⟨expression⟩;
```

wobei `identifier` ein noch nicht benutzter Name ist. Die Parameter `p1` bis `pn` dürfen schon als Variablen benutzt sein. In diesem Fall wird aber eine lokale Kopie beim definierenden Ausdruck `expression` verwendet, so daß die Variable nicht verändert wird und auch nicht zur Definition herangezogen werden kann.

Der definierende Ausdruck `expression` hat in beiden Fällen folgenden rekursiven Aufbau:

```
<expression>::=<expression>+<expression> |
               <expression>-<expression> |
               <expression>*<expression> |
               <expression>/<expression> |
               <factor>

<factor>::=<real number> |
           <identifier> |
           <function evaluation> |
           (<expression>)

<function evaluation>::=<identifier(<exp1>,...,<expn>)
```

Zahlreiche Variablen und Funktionen sind schon vordefiniert. Bitte informieren Sie sich durch Abbildung A.14.

Tx,Ty

Es gibt zwei vordefinierte Funktionen `Tx` und `Ty`, die bei Angabe von Offsets bei `setpoint` Kommandos dazu dienen, eine Koordinatentransformation nach folgender Formel durchzuführen:

```
setpoint x[a,b] ⟶ setpoint x[Tx(a,b),Ty(a,b)]
```

Diese Funktionen sind als

```
def Tx(a,b) = a;
def Ty(a,b) = b;
```

vordefiniert. Im Normalfall wird also beim Erzeugen eines `setpoint`-Offsets durch die Funktion `Tx` der x-Koordinatenanteil ausgeblendet, durch die Funktion `Ty` der y-Koordinatenanteil. Durch Umdefinition dieser Abbildung lassen sich Effekte wie Transformationen, Skalierungen, Drehungen, etc. auf einfache Weise erzeugen. Konsultieren Sie für diese Möglichkeiten am Besten ein Buch zur grafischen Datenverarbeitung wie *Computer Graphics: Principles and Practice* von Foley/van Dam/Feiner/Hughes.

undef

Mit dem Schlüsselwort `undef` lassen sich mittels `def` vereinbarte Variablen und Funktionen wieder löschen.

setdefaults

Stellt die wichtigsten Parameter auf die Standardwerte zurück, wie in Abbildung 4.3 zu sehen. Außerdem werden alle Variablen- und Funktionsdefinitionen gelöscht!

Liniendicke	0.4 *pt*
Linienmuster	durchgezogen
Pfeilspitze	1.2 *pt* breit, 6.4 *pt* lang
Pfeilart	Geradenstücke, keine Ablenkung
Punktdicke	2.4 *pt*
Linienenden	abgerundet (round)
Punktzähler	0
`\unitlength`	1 *pt*

Abbildung 4.3: Standardwerte der Grafikparameter

setunitlength ⟨*dimen*⟩

Stellt den multiplikativen Faktor auf den Offset des `setpoint`-Befehls ein und bestimmt dadurch die Größe des Bildes. Beachten Sie bitte, daß per Voreinstellung durch `setdefaults` ein Wert von 1 *pt* eingetragen wird und deshalb auch relative Faktoren in der Form von Gleitpunktwerten jederzeit benutzt werden können. Bei Verwendung von expliziten Maßangaben wird dieser Faktor natürlich *nicht* aufmultipliziert!

setdotsize ⟨*dimen*⟩

Stellt den Punktdurchmesser auf ⟨*dimen*⟩.

setlinewidth ⟨*dimen*⟩

Stellt die Liniendicke auf ⟨*dimen*⟩.

setdash ⟨*dimen*⟩ ⟨*dimen*⟩ ...

Stellt das Linienmuster ein. Ohne Parameter wird das Muster zurückgesetzt auf durchgehend gezeichnete Linien. Ansonsten geben die Parameter abwechselnd die Strecken an, die schwarz bzw. weiß gezeichnet werden. Wird nur ein Parameter angegeben, ist das Muster gleichförmig schwarz-weiß, wie z.B. bei `setdash 2pt`.

defpattern ⟨*byte*⟩,⟨*byte*⟩,⟨*byte*⟩,⟨*byte*⟩,⟨*byte*⟩,⟨*byte*⟩,⟨*byte*⟩,⟨*byte*⟩

Definiert ein 8×8-Füllmuster für den `patclip`-Befehl. Die Parameter sind Dezimalzahlen im Bereich 0–255. Jedes gesetzte Bit in der Binärdarstellung der Zahl ergibt ein gesetztes Pixel im Füllmuster. Jede Zahl stellt eine Zeile des Füllmusters dar, wobei die erste Zahl der obersten Zeile entspricht. Es müssen zur Zeit genau acht Zahlen angegeben werden!

setlinecap ⟨*plain*|*round*|*arrow*|*wedge*|*triangle*⟩ ⟨*plain*|*round*|*arrow*|*wedge*|*triangle*⟩

Stellt das Aussehen von Linienenden ein. Der erste Parameter steht für den Linienanfang, der zweite für das Linienende. Die fünf möglichen Werte bedeuten „plain" für eine abgeschnittene Linie, „round" für eine Linie mit abgerundeten Enden, „arrow" für eine Linie mit geschlossener ausgefüllter Pfeilspitze, „wedge" für eine Linie mit offener Pfeilspitze ohne Abschluß und „triangle" für eine Linie mit geschlossener ungefüllter Pfeilspitze.

setarrowsize ⟨*dimen*⟩ ⟨*dimen*⟩

Stellt die Größe von Pfeilspitzen ein. Der erste Parameter gibt an, wie weit sich die Schenkel des Pfeils vom Stamm entfernen, der zweite Parameter gibt die Länge der Pfeilspitze an.

setarrowshape ⟨*real*⟩ ⟨*real*⟩

Stellt das Aussehen von Pfeilspitzen ein. Die Schenkel und der Abschluß der Pfeilspitzen bestehen aus drei Bézierkurven mit doppeltem Kontrollpunkt auf halber Länge der Kurve. Die beiden Parameter beeinflussen die Lage dieser Kontrollpunkte, wobei der erste Parameter die Schenkel, der zweite den Abschluß steuert. Die Parameter sind Gleitpunktzahlen, wobei das Vorzeichen über die Richtung der Ablenkung der Bézierkurven entscheidet. Positive Werte ziehen die Schenkel und den Abschluß nach innen in Richtung Schwerpunkt des Dreiecks, das die Pfeilspitze umschließt. Negative Werte drücken entsprechend nach außen. Bei Aufruf ohne Parameter wird auf die Standardeinstellung zurückgesetzt (Pfeil aus Geradenstücken), bei Aufruf mit einem Parameter wird nur der Schenkelparameter gesetzt.

setarrowline ⟨*dimen*⟩

Bei Linienenden von Typ `triangle` steuert dieser Befehl die Liniendicke der Pfeilspitze.

setpoint ⟨*point number*⟩ ⟨*repeated*⟩ ⟨*position*⟩ ...

Setzt einen oder mehrere der 500–1000 internen Kontrollpunkte (je nach Gerät). Der Parameter *point number* steht dabei für die Nummer des zu definierenden Punktes.

Falls mit einem `setpoint`-Kommando mehrere Punkte definiert werden, können die definierenden Ausdrücke durch Leerzeichen oder, falls wegen Eindeutigkeit notwendig, durch `,` getrennt werden.

Der Parameter *repeated* ist ein Wiederholfaktor, der die Erzeugung mehrerer Punkte mit aufeinanderfolgenden Indizes bewirkt. Ein Wiederholfaktor hat folgenden rekursiven Aufbau:

```
<repeated>::=<> |
             <repeated> |
             <identifier>=<expr1>..<expr2>:<expr3>
```

Der Wiederholfaktor kann also optional (leer) oder verschachtelt angegeben werden. Ein einfacher Wiederholfaktor besteht aus einem noch nicht definierten Bezeichner und drei Ausdrücken, wobei der erste `expr1` und zweite `expr2` den Parameterbereich des Bezeichners definieren und der dritte die Anzahl zu erzeugender Punkte. Der Parameterbereich wird in äquidistante Abschnitte zerlegt und der Bezeichner erhält für jeden erzeugten Punkt einen dieser Werte im Parameterbereich. Durch Verwendung des Bezeichners im folgenden *position*-Parameter können die Punkte auf diese Weise an verschiedene Positionen gesetzt werden.

Optional kann in eckigen Klammern ein Offset zur aktuellen Position angegeben werden, der folgendes Aussehen hat:

```
<position>::=<> |
             [<expr1>,<expr2>] |
             [<expr1>,<expr2>,<rotate>]

<rotate>::=<expression> |
           <expression>[<ex1>,<ex2>]
```

In eckigen Klammern wird durch die Ausdrücke `expr1` und `expr2` ein Verschiebungsfaktor in x- und y-Richtung angegeben. Dieser Wert wird entweder mit dem mittels `setunitlength` eingestellten Wert multipliziert oder absolut durch eine bemaßte Größe angegeben und ergibt den tatsächlichen Offset. Falls ein *repeated* Parameter vorausgegangen ist, kann durch Angabe eines Rotationsfaktors der erzeugte Punkt noch gedreht werden. Der Bezugspunkt für die Drehung ist entweder der letzte eingegebene Punkt oder der durch `ex1` und `ex2` spezifizierte Punkt. Gedreht wird in beiden Fällen um den Winkel `expression`, der im Bogenmaß angegeben werden muß.

Für Beispiele lesen Sie bitte das Grafiktutorial, Anhang A.

⟨*pointlist*⟩

Für die nun folgenden Befehle wird als Parameter jeweils eine Punktliste erwartet. Die Punktliste bezieht sich auf die mit `setpoint` vergebenen Nummern, falls ein direkter Bezug auf diese Punkte gewünscht ist. Einzelne Punkte in der Punktliste können wegen Eindeutigkeit durch $\boxed{,}$ getrennt werden. Bei Eindeutigkeit ist das Trennen durch Leerzeichen statthaft. Sollen mehrere direkt hintereinander definierte Punkte verwendet werden, reicht die Angabe des Bereichs, z.B. 105–113. Bereiche und Einzelpunkte können in der Punktliste beliebig gemischt werden.

Gibt man statt einer Punktnummer den Ausdruck $[p, q]\lambda$ an, so wird damit der Punkt $p(1 - \lambda) + q\lambda$ spezifiziert, also ein Punkt auf der Verbindungsgeraden von p und q. Für λ darf ein beliebiger Ausdruck aus den vier Grundrechenarten und runden Klammern eingesetzt werden. Selbstverständlich können die Punkte p und q wieder von der Form $[r, s]\mu$ sein.

Eine weitere Möglichkeit zur Angabe eines Punktes besteht in einem Ausdruck der Form (p, q). In diesem Fall hat der erzeugte Punkt die x-Koordinate von p und die y-Koordinate von q. Auch diese Notation darf wieder beliebig verschachtelt verwendet werden. Als Kurznotation ist statt der Angabe von p oder q das Zeichen `@` zulässig, das den zuletzt definierten Punkt der Punktliste einsetzt.

Durch Angabe von $\boxed{\texttt{@}}$, direkt gefolgt von einer Zahl ⟨n⟩, bezieht man sich auf die ⟨n⟩ zuletzt eingegebenen Punkte. Ohne Angabe einer Zahl bezieht sich `@` auf den in der Liste vorangehenden Punkt. Ist der Betrag von ⟨n⟩ negativ, wird außerdem der Punktzähler um den Betrag von ⟨n⟩ zurückgesetzt.

Als Besonderheit kann man eine Punktliste auch mit einigen speziellen Parametern versehen, die das Zeichnen von Teilobjekten und das wiederholte Zeichnen von Objekten erlauben. Ein Linienbefehl mit einer solchen modifizierten Punktliste hat dann folgendes Aussehen:

⟨`command`⟩ ⟨`pointlist`⟩ ⟨`pieces`⟩ ⟨`repeated`⟩ ⟨`position`⟩

Das Aussehen der Parameter `repeated` und `position` entspricht den gleichbezeichneten Parametern des `setpoint`-Kommandos. Allerdings ist die Bedeutung dahingehend geändert, daß das übergeordnete Kommando `command` entsprechend dem `repeat`-Faktor wiederholt wird, und zwar an den durch den `position`-Parameter bezeichneten Stellen. Ein `position`-Parameter darf nur angegeben werden, falls ein `repeated`-Parameter vorausgeht!

Der Parameter `pieces` kann verschachtelt angegeben werden und hat folgendes Aussehen:

```
<pieces>::=:<expr1>..<expr2> |
           :<expr1>..<expr2><pieces>
```

Durch die Parameter `expr1` und `expr2` wird nur ein Teil der Punktliste zum Zeichen des mit `command` erzeugten Objektes herangezogen. Die Werte können sich auf die Punktnummern der Punkteliste beziehen und dürfen in diesem Fall bei n Punkten im Bereich $0 \leq expr \leq n$ liegen. Bei bemaßten Werten wird das Zeichnen des Objekts bis zur Position `expr1` unterdrückt und dann nur bis Position `expr2` gezeichnet.

Für Beispiele lesen Sie bitte das Grafiktutorial, Anhang A.

dot ⟨*pointlist*⟩

Die Punkte der Punktliste werden mit dem voreingestellten Durchmesser als gefüllte Kreise dargestellt.

spline ⟨*pointlist*⟩

Zeichnet einen natürlichen Spline, der durch die Kontrollpunkte in ⟨*pointlist*⟩ definiert ist.

hermitespline ⟨*pointlist*⟩

In der Praxis hat man häufig das Problem, Kurven zeichnen zu müssen, die eine bestimmte Anfangs- und/oder Endrichtung haben sollen. Der Grafikbefehl zum Setzen eines solchen Hermiteschen Splines ist ist analog zum `spline`-Befehl, nur daß er eben `hermitespline` heißt. Bitte lesen Sie die Abschlußbemerkung zu Hermiteschen Splines!

lefthermitespline, righthermitespline ⟨*pointlist*⟩

Es ist auch möglich, nur an einem Ende des Splines die Richtung vorzugeben, und am anderen Ende des Splines eine natürliche Randbedingung zu fordern. Dies ist durch Verwendung von `lefthermitespline` und `righthermitespline` möglich.

! → Die Befehle zum Zeichnen Hermitescher Splines haben als Parameter eine Kontrollpunktliste, wobei der erste bzw. letzte bzw. erste und letzte Punkt der Liste eine Richtung vorgeben, die umso stärker ausgeprägt ist, je weiter der Punkt von der Kurve entfernt ist.

closedspline ⟨*pointlist*⟩

Durch diesen Befehl wird ein glatter geschlossener natürlicher Spline gezeichnet, wobei der erste und letzte Punkt der Punktliste miteinander verbunden werden.

poly, closedpoly ⟨*pointlist*⟩

Diese beiden Befehle zeichnen einen offenen bzw. geschlossenen Polygonzug, der durch die jeweilige Punktliste definiert ist.

defclip, clip, whiteclip, patclip, endclip

Diese Funktionen dienen zum Definieren eines Clippfades, bezüglich dessen Begrenzungslinie bis zum `endclip`-Kommando folgende TeX- und Grafik-Ausgaben gekappt werden. Die Begrenzungslinie *muß* geschlossen und in einem durchgehendem Stück definiert sein, unterliegt aber ansonsten keinerlei Beschränkungen. „Durchgehendes Stück“ soll in diesem Zusammenhang bedeuten, daß man nicht nachträglich auseinanderklaffende Stücke mit einem dritten schließt, sondern die Umrißlinie „wie mit einem Pinselstrich“ erzeugt. Durch Angabe von `defclip` wird der Beginn des Clippfades markiert. Alle folgenden Angaben definieren diesen Clippfad, bis ein `clip`-, `whiteclip`- oder `patclip`-Kommando die Eingabe des Clippfades abschließt. Alle folgenden TeX-Ausgaben, *also auch jeder Text*, werden bezüglich dieses Pfades gekappt, bis das `endclip`-Kommando diesen Modus beendet.

Beim Clippen beeinflussen die drei Clipbefehle die Ausgabe wie folgt:

`clip`	bisherige Elemente der Zeichnung werden nicht verdeckt
`whiteclip`	der Clipbereich wird gelöscht
`patclip`	der Clipbereich wird mit dem durch `defpattern` definierten Füllmuster ausgefüllt

input "*filename*"

Der Treiber liest die Grafikkommandos aus der Datei *filename*. Die Angabe des Dateinamens muß den Konventionen des verwendeten Betriebssystems entsprechen. Die Definition eines Pfades für solche externen Grafikdateien ist möglich. Lesen Sie dazu die entsprechenden Kapitel beim verwendeten Treiber!

! → Bitte beachten Sie, das das "-Zeichen bei manchen Macropaketen wie den Anpassungen an die deutsche Sprache `german.sty` eine Spezialfunktion bekommen hat, die die normale Verwendung wie beim `input`-Befehl unmöglich macht! In solchen Fällen muß vor Absetzen des `input` die Spezialbedeutung ausgeschaltet werden. Bei `german.sty` beschieht das z.B. durch eine Klammerung mit `\originalTeX` und `\germanTeX` Befehlen.

4.2 Macros zur Grafikeinbindung

Um Ihnen das Kopfzerbrechen zu ersparen, alle `\special`-Befehle zu lernen bzw. Ihre Grafiken auszumessen und mit `\hbox` und `\vbox` Befehlen zu hantieren, um Platz für Grafiken zu schaffen, gibt es eine ganze Anzahl vorgefertigter TeX-Macros, die Ihnen diese Arbeit abnehmen. Diese Macropakete sollten im Standard-Suchpfad von TeX liegen und können mit `\input` ⟨`name`⟩ geladen werden, wobei `name` für `graphic`, `bildmac`, `epsf`, etc. steht.

4.2.1 Die GRAPHIC-Macros

Die Datei `GRAPHIC.TEX` enthält die Macros zur Verwendung von Grafiken im Fremdformat.

IMG-Grafiken werden auf dem Atari ST zunächst mit IMGTOTEX.TTP bearbeitet. Dabei wird die gewünschte Auflösung der Grafik in dpi erfragt und in der Datei eingetragen und eine Zusatzdatei erstellt, die die Größe des Bildes in TEX-Einheiten enthält. Als Richtwert zur Auflösung kann bei Laserdruckern ein Wert von 150 dpi für Grafiken, die den ganzen Atari Bildschirm enthalten, angegeben werden. Die erzielte Größe im Ausdruck liegt in der Breite in etwa richtig für ein LATEX-Dokument. Als Beispiel siehe Abbildung B.1. In der TEX-Eingabe wird die bearbeitete Grafik dann mittels \includegraphic{filename} eingebunden. Der Platz für das Bild wird automatisch freigehalten. Zudem kann das Bild behandelt werden, wie ein einzelner Buchstabe, d.h. zentriert werden, etc.

An der Universität Augsburg wurde im Rahmen einer Diplomarbeit das objektorientierte Zeichenprogramm PCDRAW erstellt, das PostScript-Code als Ausgabe erzeugen kann. Zudem ist ein ähnlicher Mechanismus, wie für IMG-Grafiken auf dem Atari eingebaut. Mittels des Menüpunktes „Picture to TEX“ wird interaktiv die beabsichtigte Bemaßung abgefragt, der PostScript-Code des Bildes passend abgespeichert und eine Zusatzdatei für TEX erstellt. Das Bild kann dann mittels \psdraw{filename} in den Text eingebunden werden. Auch hier kann das Bild wie ein einzelner Buchstabe behandelt werden. Genaueres entnehmen Sie bitte der Beschreibung von PCDRAW.

Falls man selbst in PostScript programmiert, kann man sein fertiges Bild per Hand ausmessen und in eine zweite Datei die Größeninformation folgendermaßen ablegen:

```
\grwd = 108.160 mm
\grht = 67.600 mm
```

Dabei muß \grwd die Breite des Bildes und \grht die Höhe des Bildes zugewiesen bekommen. Wenn z.B. die erste Datei mit dem PostScript-Bild unter ESCHER.PS und die zweite Datei mit der Größeninformation unter ESCHER.TEX abgespeichert sind, so kann durch den Aufruf

```
\includeps{ESCHER}
```

dieses Bild in einen TEX-Text eingebunden werden, wobei automatisch genügend Platz freigehalten wird. Ein Beispiel für diese Technik ist Bild 4.4 von Peter Henderson und Peter Bumbulis frei nach Maurits C. Escher.

Bitte beachten Sie, daß dieser Mechanismus zum Einbinden von Bildern, die mittels PostScript erstellt wurden, inzwischen veraltet ist und nur aus Kompatibilitätsgründen unterstützt wird. Neue Projekte sollten gleich auf die EPSF-Macros zurückgreifen (siehe Abschnitt 4.2.3), die übrigens von Tomas Rokicki und Donald E. Knuth für DVIPS, einen anderen PostScript-DVI-Treiber, entwickelt wurden.

4.2.2 Die BILDMAC-Macros

Die Datei BILDMAC.TEX enthält einige Macros zur Verwendung der eingebauten Grafikbefehle. Die Entwicklung an diesen Macros dauert noch an. Die

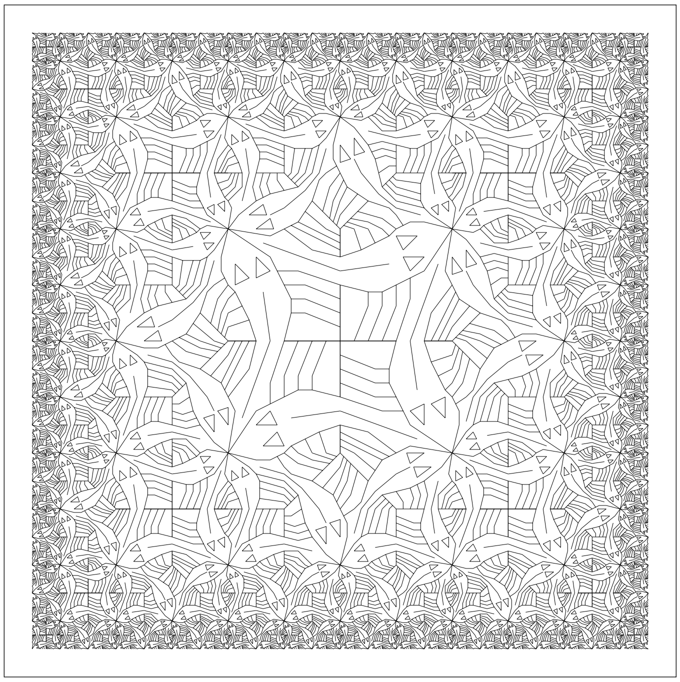


Abbildung 4.4: Eine flächenfüllende Grafik nach Escher

Macros können jedoch als eine Anregung verstanden werden, eigene Macros zu definieren.

Trotz der noch andauernden Entwicklung empfehlen die Autoren dringend die Verwendung dieser Macros anstelle der ausführlichen `\special`-Befehle, da durch jeden `\special`-Befehl mit anschließenden Zeilentrenner ein Leerzeichen erzeugt wird, das das ordnungsgemäße Funktionieren des `\put`-Befehls beeinträchtigt.

Die Kompatibilität von neuen `BILDMAC`-Dateien zu vorherigen Versionen wird ab Treiberversion 2.2 weitestgehend bewahrt. Falls dies aus irgendeinem Grund einmal nicht möglich sein sollte, ist im Handbuch eine Möglichkeit angegeben, dieselben Ausgaben mit anderen Konstruktionen zu erreichen.

Das prinzipielle Schema der `BILDMAC`-Befehle ist immer gleich. Die jeden Grafikbefehl einleitende `\special{gr`-Sequenz und etwaiger Leerraum, der den Befehlen folgt, wird herausgefiltert. Auf diese Weise wird

```
\special{gr setpoint 0}
```

zu

```
\setpoint{0},
```

usw. Bitte drucken Sie sich die Datei `BILDMAC.TEX` aus und sehen selbst, welche Befehle zur Verfügung stehen. Macros, die über Textersatz hinausgehen, sind jeweils (hoffentlich) kommentiert.

4.2.3 Die EPSF-Macros

In den Dateien `EPSF.TEX` bzw. `EPSF.STY` liegen Macros vor, die zur Einbindung von Bildern in Encapsulated PostScript dienen. Wenn man eine PostScript-Datei vorliegen hat, sollte man zunächst feststellen, ob sie in Encapsulated PostScript-Form vorliegt. Falls dies nicht der Fall ist, muß man die Datei in diese Form bringen. Laden Sie also die Datei mit der Grafik in einen Editor. Sollte es dabei Probleme mit der Zeilenlänge geben, hat das erzeugende Programm den PostScript-Standardzeilentrenner LF verwendet und Sie müssen die Datei mit `UNIX2DOS` oder `CRLF`[1] bearbeiten, so daß eine Standard-ASCII-Datei entsteht, die Ihr Drucker übrigens problemlos verdaut. Haben Sie die Datei erfolgreich geladen, sollte unter den ersten paar Dutzend Zeilen ein PostScript-Kommentar

```
%%BoundingBox: 1 2 3 4
```

oder zumindest

```
%%BoundingBox: (atend)
```

zu finden sein. Im zweiten Fall kopieren Sie den richtigen `Bounding Box` Kommentar vom Dateiende an den Dateianfang und löschen die jetzt obsolete `atend`-Zeile. Die Datei ist jetzt problemlos durch TEX und `DVILW` zu verarbeiten.

Haben Sie keinen `%%Bounding Box`-Kommentar gefunden, müssen Sie selbst einen einfügen. Dazu steht Ihnen die Hilfsdatei `BB.PS` von Bernie Cosell zur Verfügung. Kopieren Sie den Inhalt von `BB.PS` und Ihres Bildes in eine neue Datei und drucken dieses Bild aus. Dazu müssen Sie evtl. am Ende der Datei ein `showpage`-Kommando einfügen. Das neue Bild hat jetzt einen Rahmen und die notwendigen Parameter für die `Bounding Box` stehen am linken unteren Ende der Grafik. Laden Sie nun erneut das ursprüngliche Bild und fügen irgendwo am Anfang der Datei eine Zeile

```
%%BoundingBox: 1 2 3 4
```

ein, wobei Sie anstelle von `1`, `2`, `3` und `4` die richtigen Werte von Ihrem Testausdruck übernehmen. Falls Ihre Datei eine `showpage`-Anweisung enthielt, brauchen Sie diese nicht entfernen, da das Ausdrucken der Grafik jetzt im Zusammenhang mit dem TEX-Text geschieht und die entsprechenden Befehle von `DVILW` erzeugt werden, d.h. das `showpage` wird lokal von `DVILW` umdefiniert.

Zum Setzen des Bildes im laufenden Text benötigt TEX einige Macros, die — wie schon vorher erwähnt — in der Datei `EPSF.STY` gespeichert sind. Laden Sie diese Datei durch

```
\input epsf.sty
```

(TEX) oder durch

[1] Quellcode wird mitgeliefert

```
\documentstyle[...,epsf,...]{...}
```

(LaTeX). Sie können natürlich in TeX auch die Datei `EPSF.TEX` werwenden und sich die Endung beim `\input`-Befehl sparen. Der Inhalt der Dateien ist gleich! Jetzt können Sie Ihr Bild durch

```
\epsfbox{bild.ps}
```

setzen lassen, wobei durch TeX bzw. LaTeX automatisch durch die `Bounding Box`-Anweisung genügend freier Platz gelassen wird. Außerdem haben Sie die Möglichkeit, das PostScript-Bild in der Größe dem Text anzupassen. Dazu müssen Sie lediglich *vor* dem `\epsfbox`-Kommando[2] eines der Kommandos

```
\epsfxsize=<dimen>
\epsfysize=<dimen>
```

absetzen, um das Bild horizontal bzw. vertikal zu skalieren. Der Parameter `<dimen>` ist eine gültige TeX-Maßeinheit wie 150pt oder 5.7cm. Falls Sie beide Kommandos absetzen, hat das `\epsfxsize`-Kommando Vorrang. Dadurch können keine Verzerrungen entstehen! Ein günstiger Wert für die horizontale Skalierung ist z.B. für TeX `\hsize` bzw. für LaTeX `\textwidth`. Dadurch ist das Bild genauso breit, wie der umgebende Text.

! → Bei Verwendung von LaTeX kann es bei manchen Environments notwendig sein, unmittelbar vor das `\epsfbox`-Kommando ein `\leavevmode`-Kommando zu stellen. Sie werden die entsprechende Fehlermeldung bekommen und können ganz gelassen reagieren!

Durch Absetzen des Befehls `\epsfverbosetrue` *vor* den `\epsfbox`-Befehlen können Sie sich die Größe der eingebundenen PostScript-Grafiken am Bildschirm anzeigen lassen.

[2] Achtung: in älteren Versionen der `EPSF`-Macros heißt `\epsfbox` `\epsffile`.

Anhang A. Grafiktutorial

Grundlagen: Punkte, Linien, Einheiten und Macropakete

`DVI` und `DVILW` erzeugen ihre Grafiken dadurch, daß man einige Punkte durch ihre Positionen definiert und diese dann durch verschiedene Linien miteinander verbindet oder durch verschieden dicke Punkte sichtbar macht. Dazu folgendes Plain-TEX-Beispiel:

```
\leftline{\hskip 3cm\special{gr setpoint 0}}
\rightline{\special{gr setpoint 1; poly 0,1}
  \hskip 3cm}
```

Dadurch entsteht eine ähnliche Abbildung wie links.

Verwendet wurden in diesem Beispiel zwei `\special`-Befehle des Treibers, nämlich `setpoint`, womit die Positionen der Punkte 0 und 1 festgelegt wurden und `poly`, der die Verbindung der beiden Punkte 0 und 1 durch eine Gerade zeichnete. Um die Positionierung etwas zu erleichtern, werden folgende TEX-Macros und Variablen für Plain-TEX definiert:

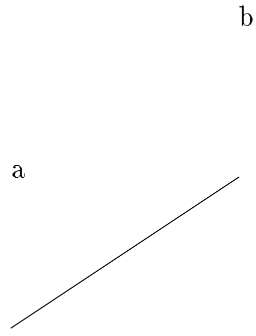
```
\newdimen\unitlength \unitlength = 1pt
\def\setunitlength#1{%
  \unitlength=#1\special{gr setunitlength #1}}

\def\picture(#1,#2)#3{\vbox to #2\unitlength{\vss\hbox
  to #1\unitlength{#3\hss}}}

\def\put(#1,#2)#3{\unskip\raise#2\unitlength\hbox
  to 0pt{\kern#1\unitlength#3\hss}\ignorespaces}
```

Für LATEX ist natürlich die Definition der Macros `\picture` und `\put` unnötig, da das `picture`-Environment genau dasselbe leistet. Der nächste Absatz muß deshalb von LATEX-Anwendern übersetzt werden, d.h. mit den notwendigen `\begin{}` und `\end{}`-Zusätzen versehen werden.

Die neue Variable `\unitlength` gibt die Maßeinheit an, in der alle Punktekoordinaten angegeben werden können. Das Macro `\setunitlength` dient der Veränderung dieser Maßeinheit, zusätzlich wird den Treibern diese Veränderung über das Grafik-Kommando `setunitlength` mitgeteilt. Es ist wichtig, sich darüber klarzuwerden, daß die TEX-Variable `\unitlength` für die Treiber unbekannt ist, und nur für die beiden Macros `\picture` und `\put` von Bedeutung ist. Alternativ ist auch die Verwendung von absoluten Längenangaben, wie sie TEX versteht (z.B. 1.1 *cm*, 5 *pt*), erlaubt.

Mit `\picture(w,h)` wird Platz für ein Bild der Breite `w` und der Höhe `h` geschaffen. Mit `\put(x,y)` können Bildelemente innerhalb von `\picture` positioniert werden. Beispiel: Die Befehle

```
\setunitlength{1cm}
\centerline{\picture(3,2){\put(0,0){a}\put(3,2){b}}}
```



erzeugen das Bild links.

Anstatt der Buchstaben a und b sollen nun die Endpunkte einer Geraden an dieselben Stellen gesetzt werden. Dies geschieht mit folgenden Befehlen, die das Bild links liefern:

```
\setunitlength{1cm}
\centerline{\picture(3,2){
  \put(0,0){\special{gr setpoint 0}}
  \put(3,2){\special{gr setpoint 1; poly 0,1}}}}
```

Punktdefinitionen für fortgeschrittene Aufgaben

Es ist jedoch nicht nötig, jede einzelne Punkteposition durch einen neuen `\special`-Befehl festzulegen, vielmehr ist das Kommando `setpoint` selbst in der Lage, Punktpositionen durch Angabe von Koordinaten zu definieren. Das obige Bild hätte auch so erzeugt werden können:

```
\setunitlength{1cm}
\centerline{\picture(5,2){\special{gr setpoint 0 1[4,2];
  poly 0,1}}}
```

Dieses Beispiel soll zeigen, daß es möglich ist, beliebig viele Punktpositionen durch ein einziges `setpoint`-Kommando festzulegen. Zum anderen sieht man, daß es möglich ist, zur Punktenummer zusätzlich eine x- und eine y-Koordinate anzugeben. Die Maßeinheit dieser Koordinatenangaben wird durch das Kommando `setunitlength` festgelegt, falls die Angaben Gleitpunktwerte sind. Wahlweise könnte auch eine andere gültige TEX-Längenangabe wie `0.3 true cm` benutzt werden. In diesem Fall wurden die Offsets durch das TEX-Macro `\setunitlength` festgelegt.

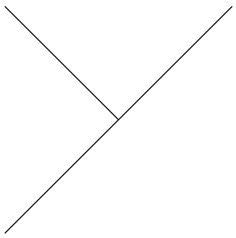
Eine weitere Möglichkeit, das obige Bild zu erzeugen, ist:

```
\setunitlength{1cm}
\centerline{\picture(5,2){\special{gr setpoint 0, [4,2];
  poly 0,1}}}
```

Wie man sieht, ist die Punktenummer des zweiten gesetzten Punktes weggelassen, allerdings steht nun ein Komma zwischen beiden Angaben, da dies sonst wie `0[4,2]` gelesen würde. Werden Punktenummern nicht angegeben, so numerieren `DVI` und `DVILW` die Punkte automatisch in aufsteigender Reihenfolge. Dies ermöglicht auch folgende Eingabe:

```
\setunitlength{1cm}
\centerline{\picture(5,2){
  \special{gr setpoint [0,0] [4,2]; poly @2}}}
```

Hier sind überhaupt keine Punktenummern mehr angegeben. Natürlich ist diesem Zusammenhang nicht mehr unbedingt klar, welche Punktenummern der Treiber verwendet hat, denn dies hängt ja davon ab, welche Punktenummern in vorangegangenen Bildern verwendet wurden. Deshalb wurde im `poly`-Kommando anstatt einer Liste von Punkten der Ausdruck `@2` verwendet, der besagt, daß die Punkteliste aus den beiden zuletzt gesetzten Punkten besteht. Auch der Ausdruck `@-2` wäre an dieser Stelle möglich gewesen. In diesem Fall wäre der interne Punktezähler des Treibers wieder um den Wert 2 zurückgesetzt worden.

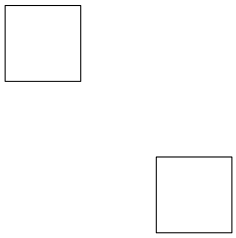


Die nächste Aufgabe besteht darin, Punkte auf einer Linie als Ausgangspunkte anderer Linien zu verwenden, wie links im Bild zu sehen ist.

Der Code, der dieses Bild erzeugte, hat folgendes Aussehen:

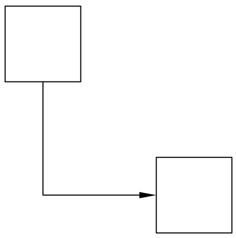
```
\setunitlength{1cm}
\centerline{\picture(3,3){
  \special{gr setpoint 0, [3,3], [0,3];
    poly 0,1; poly [0,1]1/2, 2;}}}
```

Setzt man statt einer Punktnummer den Ausdruck $[p,q]\lambda$ ein, so wird damit der Punkt $p(1-\lambda)+q\lambda$ spezifiziert. `[0,1]1/2` ist also der Punkt genau in der Mitte zwischen den Punkten 0 und 1. Für λ kann übrigens ein beliebiger Ausdruck[1] stehen, der z.B. aus reellen Zahlen, den vier Grundrechenarten und runden Klammern gebildet wird. Anstatt der Punktenummern p und q kann auch wieder ein Ausdruck der Form $[p',q']\lambda'$ stehen.



Jetzt soll folgendes Problem gelöst werden: Zwei Quadrate sind wie in der Abbildung links gegeben, die übrigens folgendermaßen erzeugt wurde:

```
\setunitlength{1cm}
\centerline{\picture(3,3){
  \special{gr setpoint 0[0,2] [1,2] [1,3] [0,3]
                     [2,0] [3,0] [3,1] [2,1];
                closedpoly 0-3; closedpoly 4-7}}}
```



Das linke obere Quadrat hat die Eckpunkte 0,1,2,3 das rechte untere die Eckpunkte 4,5,6,7. Nun soll ein Pfeil von der Mitte der Unterkante des linken Quadrats zur Mitte der linken Kante des rechten Quadrats gezeichnet werden. Dies geschieht durch folgenden zusätzlichen Befehl im `\special`-Kommando mit dem Ergebnis, das links zu sehen ist.

```
setlinecap round arrow;
poly [0,1]1/2, ([0,1]1/2,[4,7]1/2), [4,7]1/2
```

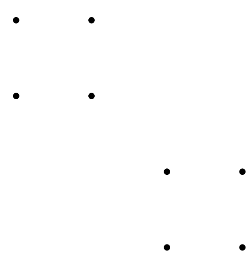
Die Schreibweise (p,q) spezifiziert einen Punkt, der die x-Koordinate von p und die y-Koordinate von q besitzt. Selbstverständlich kann auch diese Schreibweise wieder beliebig geschachtelt werden. Das Kommando `setlinecap round arrow` dient lediglich der Darstellung von Pfeilspitzen am hinteren Ende einer Linie. Die verschiedenen Möglichkeiten, Linienenden zu gestalten, werden später gesondert besprochen. Eine andere Möglichkeit, den Pfeil zu erhalten, stellt folgendes `DVI/DVILW`-Kommando dar:

[1] Lesen Sie dazu auch den Abschnitt über Ausdrücke

```
poly [0,1]1/2, (@,[4,7]1/2), (4,@)
```

Das Symbol `@` steht bei einem Linienkommando wie `poly` für den in der Punkteliste zuletzt spezifizierten Punkt, im ersten Fall also für den Punkt `[0,1]1/2` und im zweiten Fall für den Punkt `([0,1]1/2,[4,7]1/2)`.

Das Zeichnen von Punkten



Bisher wurden die verwendeten Punkte als mathematisches Objekt ohne räumliche Ausdehnung betrachtet und durch Linien verbunden. Es ist jedoch auch möglich, an Punktepositionen gefüllte Kreise mit beliebigen Durchmesser zu zeichnen. Hierzu werden die Befehle `dot` und `setdotsize` verwendet. Wenn dieselben Punkte, die schon im Bild mit den Quadraten verwendet wurden, als Grafikpunkte der Dicke 2.4 *pt* dargestellt werden, ergibt sich das Bild links. Im Code wurden lediglich die beiden `closedpoly`-Anweisungen durch `dot 0-7` ersetzt.

Das Zeichnen von Linien

Die nun folgenden Beispiele sollen zeigen, welche Möglichkeiten bestehen, verschiedene Kurven durch definierte Punkte zu ziehen. Zunächst seien die folgenden 5 Punkte definiert:

```
setpoint 0[1,0] [2,0] [2,1] [0,1] [0,0]
```

Dann ergeben sich die Möglichkeiten von Abbildung A.1.

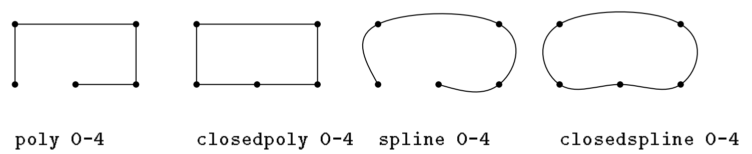


Abbildung A.1: Kurven durch Punkte

Spline-Kurven

Zunächst betrachten wir die dritte Kurve näher, den sog. **Spline**. Tja, und was ist nun ein Spline? Grob gesagt, eine Kurve, die irgendwie durch eine Anzahl von Kontrollpunkten, die die Kurve definieren, verläuft. Abbildung A.2 zeigt einen Spline, der durch zwei Endpunkte gegeben ist.

Solche Splines werden, wenn sie nicht nur durch zwei Punkte bestimmt sind, mit Hilfe von Bézierkurven gezeichnet. Eine solche Bézierkurve ist durch vier Punkte bestimmt, wie in Abbildung A.3 zu sehen ist. Die Bézierkurve läuft durch die beiden Endpunkte, jeweils in Richtung der beiden Kontrollpunkte, die in Abhängigkeit von ihrer Entfernung zur Kurve globale Kontrolle unterschiedlicher Stärke ausüben. Man kann sich die Kontrollpunkte als fest verankerte Magneten vorstellen, die die bewegliche Kurve anziehen, bis sich ein Gleichgewicht eingestellt hat.

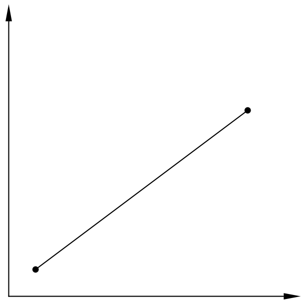


Abbildung A.2: Spline mit zwei Endpunkten

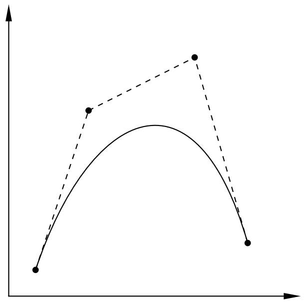


Abbildung A.3: Bézierkurve

Soll ein Spline nun durch eine beliebige Anzahl von Punkten laufen, wird er aus vielen Bézierkurven zusammengesetzt. Die Lage der Kontrollpunkte wird dabei so bestimmt, daß die Übergänge immer glatt aussehen. Lediglich der erste Kontrollpunkt der ersten Bézierkurve und der zweite Kontrollpunkt der letzten Bézierkurve sind dadurch nicht bestimmt. Bei einem *natürlichen Spline* werden diese Kontrollpunkte so gewählt, daß der Spline in den Endpunkten keine Krümmung mehr hat. Abbildung A.4 zeigt einen natürlichen Spline durch drei Punkte. Die Kontrollpunkte der Bézierkurven sind hier nicht mehr zu sehen, diese werden ohnehin automatisch berechnet.

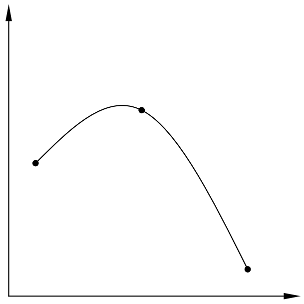


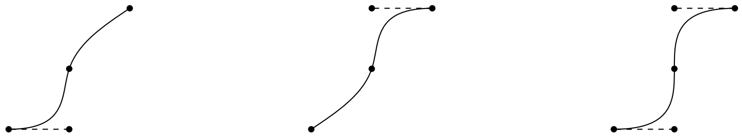
Abbildung A.4: Natürlicher Spline

Splines für fortgeschrittene Aufgaben

Die nächsten Beispiele sollen zeigen, daß es auch möglich ist, Splines mit vorgegebenen Ableitungen an den Enden zu zeichnen. Dazu seien die folgenden Punkte gegeben:

```
setpoint 0[0,0] [1,1] [2,2] [1,0] [1,2]
```

Die Punkte 3 und 4 sollen in diesem Beispiel die Ableitungen definieren. In der Praxis sieht dies aus, wie Abbildung A.5.



lefthermitespline 3,0-2   righthermitespline 0-2,4   hermitespline 3,0-2,4

Abbildung A.5: Hermitesche Splines

Die Praxisvorgabe ist die Aufgabe, Kurven zeichnen zu müssen, die eine bestimmte Anfangs- und/oder Endrichtung haben sollen. Nehmen wir zum Beispiel an, der Spline in Abbildung A.4 soll im linken Endpunkt senkrecht nach oben hinauslaufen und im rechten Endpunkt senkrecht nach unten einmünden. Dies kann durch die zusätzliche Vorgabe der beiden noch freien Kontrollpunkte am Anfang und Ende geschehen. Ein solchen Spline wollen wir hier *Hermiteschen Spline* nennen. Abbildung A.6 zeigt das Ergebnis.

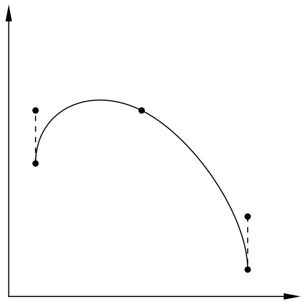


Abbildung A.6: Hermitescher Spline

Bei einem `lefthermitespline` dient der erste Punkt in der Liste dazu, die Richtung der Kurve zu definieren, d.h. die Kurve startet im *zweiten* Punkt der Liste in Richtung des *ersten* Punktes. Je weiter dieser erste Punkt vom Startpunkt entfernt ist desto ausgeprägter ist die Richtungsvorgabe. Ein `righthermitespline` ist das Analogon zum `lefthermitespline`, nur ist hier die Richtung im letzten Punkt der Liste vorgegeben. Die Kurve endet im *vorletzten* Punkt in die Richtung des *letzten* Punktes zeigend. Zwei solcher Kurven sind in Abbildung A.7 zu sehen. Die untere der beiden Kurven hat im linken Punkt eine vorgegebene Richtung, die obere Kurve im rechten Randpunkt.

Beim `hermitespline` sind die Richtungen in beiden Endpunkten der Kurve, also im *zweiten* und *vorletzten* Punkt der Liste vorgegeben. Die Kurve startet in Richtung des *ersten* angegebenen Punktes und endet in die Richtung des *letzten* angegebenen Punktes zeigend.

Eine weitere Sorte von Splines sind geschlossene Splines, wie einer in Abbildung A.8 zu sehen ist. Der Grafikbefehl ist — wie nicht anders zu erwarten

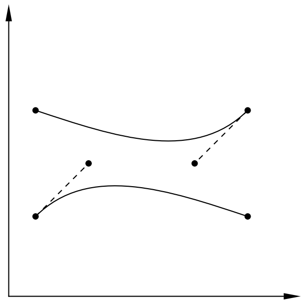


Abbildung A.7: Halbe Hermitesche Splines

— analog zum `spline`-Befehl `closedspline` und hat als Parameter eine Kontrollpunktliste. Der erste und letzte Punkt werden so miteinander verbunden, daß eine glatte Kurve entsteht.

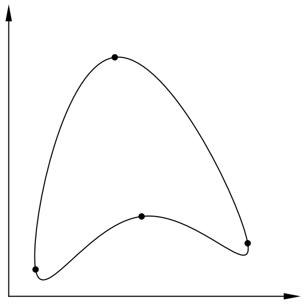


Abbildung A.8: Geschlossener Spline

Das Zeichnen von Geraden

Bei all diesen bisher gezeichneten Splines handelt es sich um sogenannte *kubische* Splines. Es gibt allerdings auch noch die Möglichkeit *lineare* Splines zu zeichnen. Diese linearen Splines sind Polygone. Abbildung A.9 zeigt einen einfachen Polygonzug, Abbildung A.10 einen geschlossenen Polygonzug.

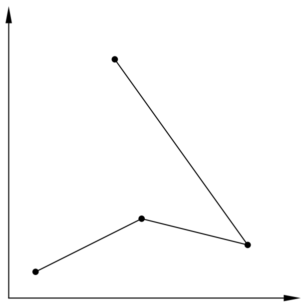


Abbildung A.9: Polygonzug

Die Befehle zum Zeichnen dieser Polygonzüge sind `poly` und `closedpoly` und erwarten als Parameter jeweils eine Punktliste.

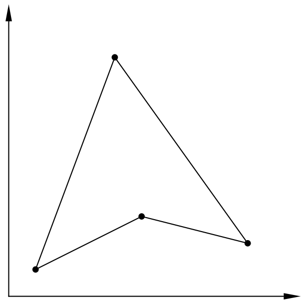


Abbildung A.10: Geschlossener Polygonzug

Das Zeichnen von gefüllten Flächen

Es ist möglich, beliebige TEX-Ausgaben an einem geschlossenen Kurvenzug, unabhängig davon, aus welchen Teilen er besteht, zu kappen. Eine Demonstration dieser Möglichkeit stellt Abbildung A.11 dar, die eine Fläche enthält, die von einem Spline und einem Polygonzug begrenzt ist und mittels Kappen an der Begrenzungslinie mit einer `\hrule` gefüllt wurde.

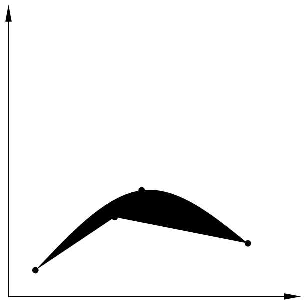


Abbildung A.11: Gefüllte Fläche

Das Füllen geschieht durch Angabe eines Clippfades und beliebigen TEX-Ausgaben, die bezüglich dieses Pfades gekappt werden.

Die Definition des Clippfades geschieht mit dem Befehl `defclip`, der den Beginn eines Clippfades markiert und mit dem Befehl `clip`, der den Clippfad abschließt und gleichzeitig das Kappen einschaltet. Wird statt `clip` der Befehl `whiteclip` verwendet, wird das Innere des Clippfades gelöscht. Alle folgenden Ausgaben werden bezüglich dieses Pfades gekappt, bis der `endclip`-Befehl das Kappen ausschaltet.

Zur einfacheren Handhabung von Füllmustern betrachten Sie bitte die zwei Macros `fill(x,y){}` und `sfill(x,y){}`, die folgendermaßen definiert sind:

```
\newbox\fillbox
\newdimen\fillshift
\newdimen\fillwidth

\def\fill(#1,#2)#3{\setbox0\hbox to #1\unitlength{
  \cleaders\hbox{#3}\hfil}\loop
  \ifdim\ht\fillbox<#2\unitlength\setbox\fillbox
```

```
  \vbox{\unvbox\fillbox\copy0}\repeat\hbox to 0pt{
  \vbox to 0pt{\vss\box\fillbox\vglue0pt}\hss}}

\def\sfill(#1,#2)#3{\setbox0\hbox{#3}%
  \fillshift=0.5\wd0\fillwidth=#1\unitlength\loop
  \ifdim\ht\fillbox<#2\unitlength\setbox\fillbox\vbox{
  \unvbox\fillbox\hbox to\fillwidth{\cleaders\hbox{#3}
  \hfil}}\advance\fillwidth by\fillshift\fillshift=
  -\fillshift\repeat\hbox to 0pt{\vbox to 0pt{\vss\box
  \fillbox\vglue0pt}\hss}}
```

Ausgehend von der aktuellen Position wird ein rechteckiger Bereich der Breite `x` und Höhe `y` mit dem jeweiligen beliebigen Parameterinhalt überdeckt. Bei Verwendung von `fill` wird der Parameter in aufeinanderfolgenden Zeilen direkt untereinander gedruckt, bei `sfill` dagegen etwas seitlich verschoben.

Als Anwendung der Füllbefehle betrachten Sie bitte folgenden LaTeX-Code, mit dem Abbildung A.11 erzeugt wurde:

```
\begin{figure}[hbp]\centering
\begin{picture}(110,110)(0,0)
  \special{gr setlinecap round arrow}
  \put(0,0){\special{gr setpoint 0}}
  \put(110,0){\special{gr setpoint 1}
  \special{gr spline 0-1}}
  \put(0,110){\special{gr setpoint 1}
  \special{gr spline 0-1}}

  \special{gr setlinecap arrow arrow}
  \put(10,10){\special{gr setpoint 0; dot 0}}
  \put(50,40){\special{gr setpoint 1; dot 1}}
  \put(90,20){\special{gr setpoint 2; dot 2}}
  \put(40,30){\special{gr setpoint 3; dot 3}}
  \special{gr defclip; spline 0-2;
    poly 2,3,0; clip}
  \put(0,0){}\vbox{\hrule width 110pt
    height 110 pt depth 110pt}\special{gr endclip}
\end{picture}
\caption{Gef"ullte Fl"ache}
\end{figure}
```

Natürlich ist es manchmal lästig, sich ein Füllmuster als TeX/LaTeX-Code auszudenken, weil sich Füllmuster oft als Bitmap leichter definieren lassen. Wenn Sie anstelle der `clip`- bzw. `whiteclip`-Befehle den `patclip`-Befehl verwenden, wird der Clipbereich mit einem durch `defpattern` angegebenen Füllmuster gefüllt. Die Parameter dieses Befehls sind *genau acht Bytes*, die das Füllmuster spezifizieren. Im Moment sind also auch nur Füllmuster mit 8×8-Pixeln möglich. Die Bytes selbst werden als Dezimalzahl angegeben und jedes gesetzte Bit entspricht einem gesetzten Pixel im Füllmuster. Erinnern Sie sich noch an Abbildung 4.1? Die Realisierung dieses Bildchens mit Treiber-Befehlen sieht wie folgt aus:

```
\setdefaults\setunitlength{1bp}
\begin{picture}(354,100)(0,0)
  \put(0,0){\setpoint{0}}
  \put(100,100){\setpoint{1}}
  \put(354,0){\setpoint{2}}
  \defpattern{170,85,170,85,170,85,170,85}
  \defclip\closedpoly{0-2}\patclip\endclip
  \setlinewidth{1 pt}\closedpoly{0-2}
\end{picture}
```

Das Ergebnis können Sie in Abbildung A.12 bewundern.

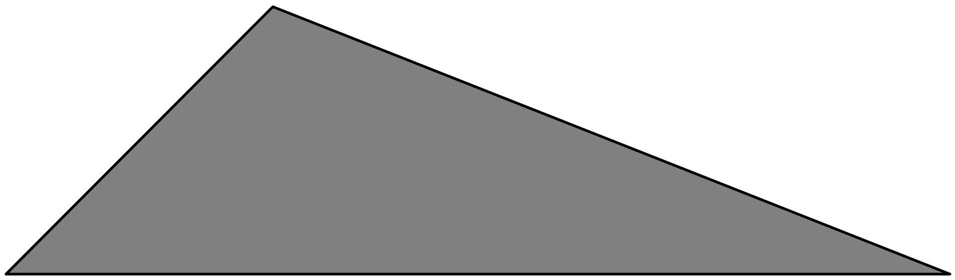


Abbildung A.12: Anwendungsbeispiel für `defpattern`, `patclip`

Als kleines Bonbon folgt noch die etwas umfangreichere Abbildung A.13, damit Sie sehen, was Sie jetzt beispielsweise zeichnen können, falls Ihnen nicht gerade eine TeX-Implementation mit Speicherbeschränkungen den Spaß verdirbt (TeX capacity exceeded ...).

Platzhalter, Funktionen und Ausdrücke

Speziell bei Aufgabenstellungen aus dem naturwissenschaftlichen Bereich sind nicht nur einzelne Punkte von Grafiken bekannt, die durch Geraden bzw. interpolierende Kurven miteinander verbunden werden, sondern die mathematischen Formeln, die die Grafik vollständig beschreiben. Für solche Anwendungen bieten die Treiber die Möglichkeit, mit Werten, Variablen, Funktionen und Ausdrücken, die aus den Grundrechenarten und diesen Konstrukten — also wiederum Werten, Variablen und Auswertungen von Funktionen — aufgebaut sind, zu arbeiten.

Das Schlüsselwort `def` dient zur Definition von Variablen und Funktionen. Dazu gibt man einen (bei Funktionen noch undefinierten) Namen als Platzhalter an, bei Funktionen gefolgt von in runden Klammern eingeschlossenen Platzhaltern der Parameter, und läßt nach einem =-Zeichen den definierenden Ausdruck folgen. Die Definition wird durch einen Strichpunkt abgeschlossen, wenn noch weitere Treiberkommandos im selben `\special`-block folgen. So definieren z.B.

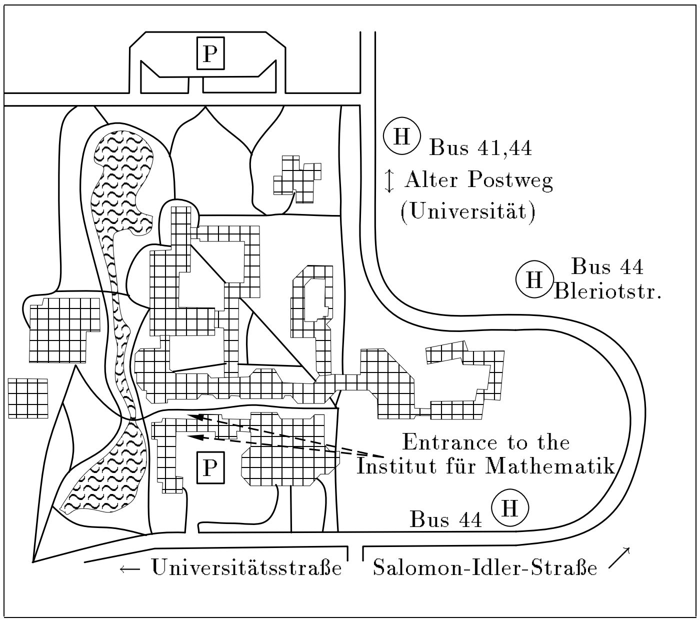


Abbildung A.13: Augsburg University

```
def factor=365.3/0.33;
```

und

```
def test(x,n)=sin(x/n);
```

eine Variable `factor` mit aktuellem Wert 1106.97 bzw. eine Funktion `test` in Abhängigkeit von den Parametern `x` und `n`. Die Variablen und Funktionen aus Abbildung A.14 sind bereits vordefiniert:

`pi`	Kreiszahl $\pi = 3.1415\ldots$
`e`	Eulerkonstante $e = 2.71828\ldots$
`sin(x)`	Sinus
`cos(x)`	Cosinus
`tan(x)`	Tangens
`asin(x)`	Arcus Sinus
`acos(x)`	Arcus Cosinus
`atan(x)`	Arcus Tangens
`exp(x)`	Exponentialfunktion
`ln(x)`	natürlicher Logarithmus
`log(x)`	10er Logarithmus
`pow(x,y)`	Potenzierung x^y
`jump(y0,y1,x)`	`if (x<0) jump=y0; else jump=y1;`

Abbildung A.14: Vordefinierte Variablen und Funktionen

Kurven im $\mathbb{R}^2$

Durch die gerade beschriebenen Möglichkeiten kann man z.B. fast beliebige Kurven im $\mathbb{R}^2$ zeichnen, indem man selbstdefinierte oder vordefinierte

Funktionen benutzt, um die Offsets des `setpoint`-Kommandos zu berechnen. Abbildung A.15 zeigt ein Beispiel.

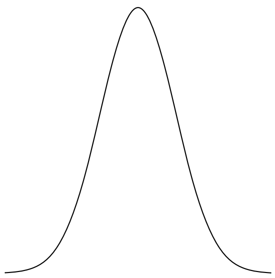


Abbildung A.15: Gauss-Glockenkurve

Der LaTeX-Code zum Erzeugen von Abbildung A.15 hat folgendes Aussehen:

```
\setdefaults
\begin{picture}(100,100)(0,0)
\special{gr def gauss(x)=exp(-x*x);}
\put(0,0){\special{gr setpoint 0[0,100*gauss(-2.5)]
  1[10,100*gauss(-2)] 2[20,100*gauss(-1.5)]
  3[30,100*gauss(-1)] 4[40,100*gauss(-0.5)]
  5[50,100*gauss(0)] 6[60,100*gauss(0.5)]
  7[70,100*gauss(1)] 8[80,100*gauss(1.5)]
  9[90,100*gauss(2)] 10[100,100*gauss(2.5)];
  spline 0-10}}
\end{picture}
```

Bitte beachten Sie, daß dieses Beispiel nur zur Demonstration des eben Besprochenen dient. Es gibt wesentlich bequemere Methoden, Bild A.15 zu erzeugen, indem man die Wiederholmöglichkeiten der Grafikbefehle ausnutzt. Dazu mehr im nächsten Abschnitt!

Das wiederholte Zeichnen gleicher Objekte

Eine immer wiederkehrende Aufgabe liegt im wiederholten Zeichnen von bestimmten Objekten an verschiedenen Positionen. Beim `setpoint`-Kommando, allen Linientypen und dem `dot`-Befehl ist diese Wiederholungsmöglichkeit durch Angabe eines Wiederholungsfaktors mit zugehörigen Offsets gegeben. Die Syntax der entsprechenden Befehle lautet nun

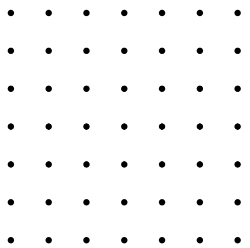
`setpoint` ⟨`number`⟩ ⟨`repeated`⟩ ⟨`position`⟩ und
`command` ⟨`pointlist`⟩ ⟨`repeated`⟩ ⟨`position`⟩,

wobei `command` einer der Befehle `dot`, `spline`, etc. ist. Der Wiederholfaktor `repeated` hat folgenden Aufbau:

⟨`identifier`⟩`=`⟨`expression1`⟩`..`⟨`expression2`⟩`:`⟨`expression3`⟩

Der Bezeichner `identifier` darf noch nicht definiert sein! Durch die Ausdrücke `expression1` und `expression2` wird der Parameterbereich des Bezeichners bestimmt und durch `expression3` schließlich die Anzahl Wiederholungen angegeben. Dabei geschehen zwei Dinge: Zum einen wird das gesamte `setpoint` bzw. Linienkommando sooft wiederholt, wie durch `expression3` angegeben, zum anderen werden genausoviele äquidistante Punkte, die den Paramterbereich vollständig überdecken, erzeugt. Im Falle des `setpoint`-Kommandos werden beginnend mit dem Startpunktindex so viele weitere Punkte erzeugt, wie durch den Wiederholfaktor angegeben. Der Bezeichner kann in der folgenden Offsetangabe `position` verwendet werden und enthält in der aktuellen Wiederholung jeweils einen Parameterwert im Parameterbereich. Als Besonderheit kann bei der Offsetangabe bei Wiederholungen zusätzlich ein Rotationsfaktor angegeben werden, der eine Drehung bei jeder Wiederholung bewirkt. Der Rotationsfaktor ist ein Ausdruck, der den absoluten Drehwinkel bezüglich der Position des durch `setpoint` definierten Punktes angibt, falls nicht in eckigen Klammern ein anderer Bezugspunkt folgt. Ein Beispiel für diese Anwendung folgt beim Zeichnen von Kreisen und Ellipsen.

Die Wiederholung läßt sich — je nach Speicherausbau des verwendeten Computers — bis zu zehnfach verschachteln. Eine Anwendung dieser Wiederholung/Verschachtelung ist das Koordinatengitter links, das durch folgende LaTeX-Anweisungen erzeugt wurde:



```
\setunitlength{1cm}
\begin{picture}(3,3)
\put(0,0){\setpoint{0[0,0]}
\special{gr dot 0 x=0..3:7 y=0..3:7 [x,y]}}
\end{picture}
```

Bitte beachten Sie, daß der erste Punkt in der Punktliste bei Verwendung von Wiederholfaktoren mit Linienkommandos schon definiert sein muß, damit die Offsets an einer definierten Stelle ansetzen! Im vorherigen Beipiel wurde dies durch das vorangehende `setpoint`-Kommando erreicht!

Die Verwendung von Wiederholfaktoren mit Linienkommandos demonstriert Abbildung A.16, bei der mehrere Sinuskurven entlang einer Parabelbahn erzeugt werden. Der zugehörige LaTeX-Code sieht wie folgt aus:

```
\setunitlength{2cm}
\begin{picture}(4.15,3)
\put(0,1){\special{gr setpoint 0 t=0..2*pi:50[t/2,sin(t)];
  spline 0-49 x=0..1:25[x,x*x]}}
\end{picture}
```

Das Zeichnen von Teilobjekten

Manchmal benötigt man in einer Zeichnung nur Teilstücke einer bekannten Gesamtkurve. Für solche Fälle kann man Punktlisten mit einem oder mehreren zusätzlichen Auswahlparametern versehen. Die Syntax ist wie folgt beschrieben:

```
command ⟨pointlist⟩:⟨expr1⟩..⟨expr2⟩
```

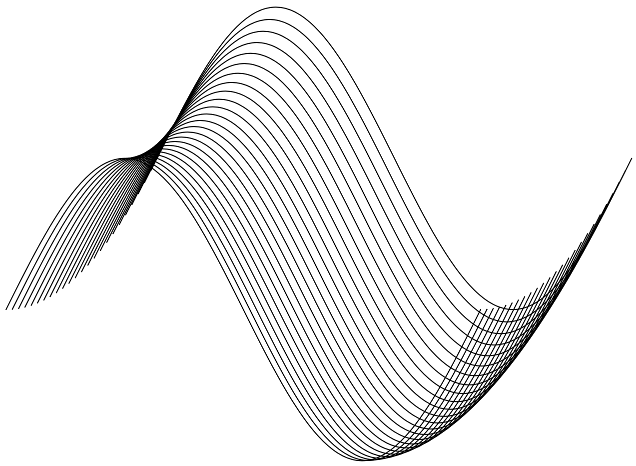
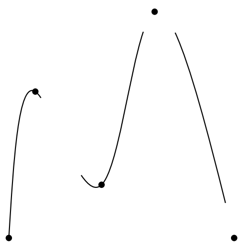


Abbildung A.16: Wiederholfaktoren mit Linienkommandos



Dabei ist `command` einer der Linienbefehle `poly`, `spline`, etc. Die Punktliste ⟨`pointlist`⟩ besteht aus zwei oder mehr Punkten. Das Zeichen „:“ ist das syntaktische Erkennungszeichen für einen Auswahlparameter und die beiden Parameter `expr` sind zwei Gleitpunktzahlen, die die Auswahl des Kurvenstücks vornehmen. Der Wert dieser Zahlen muß im Bereich $0 \leq expr \leq n-1$ liegen, wobei n die Anzahl der Punkte in der Punktliste ist. Ansonsten sollten die Längenangaben die Länge der erzeugten Kurve nicht überschreiten. Die wiederholte Angabe von Auswahlparametern ist zulässig. Folgender LaTeX-Code liefert das Bild links:

```
\setdefaults
\setunitlength{1pt}
\begin{picture}(95,95)
\put(0,0){\special{gr setpoint 0[5,5] 1[15,60] 2[40,25]
  3[60,90] 4[90,5]}}
\special{gr spline 0-4:0..1.1:1.7..2.8:3.3..3.9; dot 0-4}
\end{picture}
```

Falls Auswahl- und Wiederholungsparameter gleichzeitig benutzt werden, muß die Angabe des Auswahlparameters zuerst erfolgen!

Das Zeichnen von Kreisen und Ellipsen

Eine weitere Aufgabe besteht darin, einen Kreis oder eine Ellipse zu zeichnen, ohne dabei selbst die Koordinaten einiger Stützpunkte auf der Kreislinie angeben zu müssen. Dies geschieht durch Berechnung der Parameterdarstellung der Kreis- bzw. Ellipsengleichung in der Ebene. Dazu folgende Aufgabe: Gegeben seien ein Startwinkel ϕ_0 und ein Endwinkel ϕ_1 in Grad, ein Mittelpunkt x, y, horizontale und vertikale Halbachse r_x, r_y, die Anzahl approximierender Punkte n sowie ein Drehwinkel bezüglich des Mittelpunktes α. Gesucht sind die Grafikbefehle der Treiber zum Zeichnen des zugehörigen Ellipsenbogens.

Als Lösung werden die n Punkte mit Hilfe eines Wiederholfaktors und der Parameterdarstellung der Ellipse erzeugt, um α gedreht und schließlich mit einem Spline verbunden.

Der Code zum Zeichnen sieht dann wie folgt aus:

```
\special{gr
  def t0=phi0*pi/180;
  def t1=phi1*pi/180;
  def rot=alpha*pi/180;
  setpoint 0 t=t0..t1:n[rx*cos(t)+x,ry*sin(t)+y:rot[x,y]];
  spline 0-n-1}
```

Mit den Parametern $x = 50$, $y = 50$, $r_x = 50$, $r_y = 20$, $\phi_0 = 0$, $\phi_1 = 270$, $\alpha = 15$ und $n = 12$ ergibt das die Punkte von Abbildung A.17 links, die verbunden durch einen Spline den Ellipsenbogen in Abbildung A.17 rechts zeichnen.

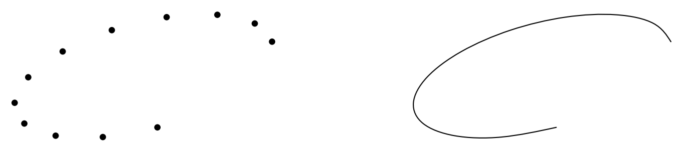


Abbildung A.17: Ellipsenbogen

Das Zeichnen von Polynomen und Polygonen

Zwei Arten von Kurven in der Ebene werden häufig zur Darstellung von Grafiken verwendet und sollen deshalb eigens erwähnt werden, nämlich Polynome und Polygone. Ein Polynom in der Ebene ist gegeben durch

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_0 + a_1 t + a_2 t^2 + \ldots \\ b_0 + b_1 t + b_2 t^2 + \ldots \end{pmatrix},$$

wobei n Punkte bei den äquidistanten Parameterwerten $t_s + (i \cdot ((t_e - t_s)/(n-1)))\quad 0 \le i \le n-1$ berechnet werden. Durch Verwendung eines Wiederholfaktors läßt sich ein solches Polynom sehr einfach darstellen. Der Parameterbereich wird durch den Bezeichner im Wiederholfaktor abgedeckt und die Polynomgleichung wird einfach zur Berechnung der Offsets herangezogen. Als Beispiel ist das Polynom

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -4 * t + t^3 \\ -4 + t^2 \end{pmatrix},$$

in Abbildung A.18 zu sehen. Der zugehörige LaTeX-Code ist:

```
\setdefaults\setunitlength{1cm}
\begin{picture}(12,7)
\special{gr setdotsize 2pt}
\put(6,4){
 \special{gr setpoint 0 t=-2.5..2.5:16[-4*t+t*t*t,-4+t*t];
             dot 0-15; spline 0-15; }}
\end{picture}
```

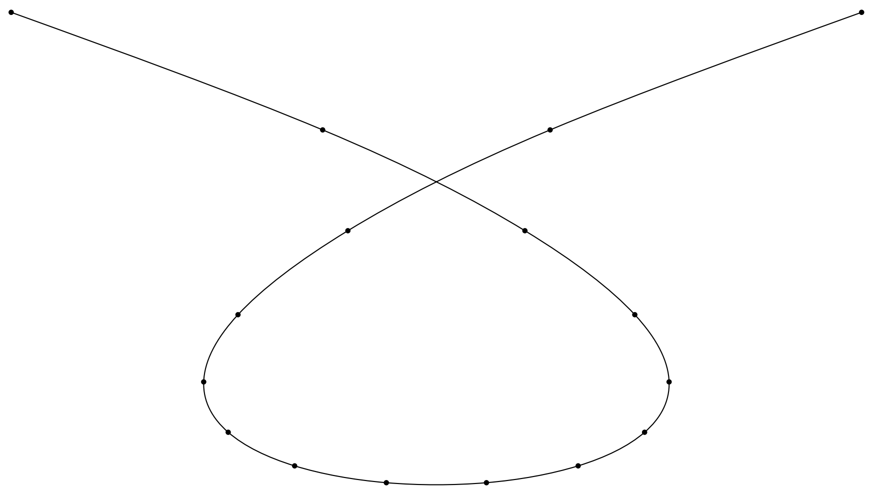
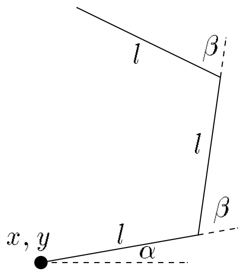


Abbildung A.18: Ein `setpoint` Polynom

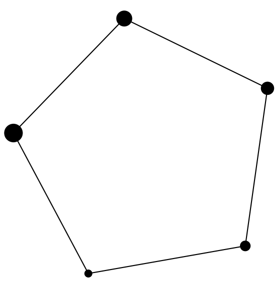
Ein (evtl. offenes) Polygon wird dadurch erzeugt, daß man zu einem gegebenen Startpunkt x, y die Seitenlänge l und einen Startwinkel α sowie einen Winkel β, der bei jedem erzeugten Punkt auf den bisherigen Winkel aufaddiert wird, angibt. Links sehen Sie ein Beispiel, in dem die entsprechenden Parameter in einer Zeichnung verdeutlicht werden.



Als eine Art Kochrezept zum Zeichnen von Polygonen kann folgendes Vorgehen dienen. Für ein Polygon, das durch n Punkte definiert werden soll, erzeugt man sich n Punkte $z_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$ durch die Iterationsvorschrift

$$z_i = \begin{pmatrix} x \\ y \end{pmatrix} + \frac{l \cdot \sin(i \cdot \gamma)}{\sin(\gamma)} \cdot \begin{pmatrix} \cos((1-i) \cdot \gamma - \alpha) \\ -\sin((1-i) \cdot \gamma - \alpha) \end{pmatrix} \quad ,$$

wobei der Parameter i im Bereich $0 \leq i \leq n-1$ an den n äquidistanten Stellen $0, 1, \ldots n-1$ eingesetzt wird und der Winkel γ sich berechnet zu $\gamma = \frac{\beta}{2}$.



Mit den Parametern $\alpha = 10°$, $\beta = 360/5°$, $l = 60$, $n = 5$ und dem Startpunkt $x = 15$, $y = 0$ ergibt sich das regelmäßige Fünfeck links. Die Eckpunkte sind in der Reihenfolge ihrer Erzeugung jeweils etwas größer gezeichnet. Der zugehörige LaTeX-Code ist folgender. Bitte beachten Sie, daß die Winkelangaben in Grad für die Treiber ins Bogenmaß umgerechnet werden müssen. Deshalb das merkwürdige Aussehen der Formeln.

```
\setdefaults\setunitlength{1pt}
\begin{picture}(90,90)
\put(0,0){\special{gr def p1=360/5*pi/180/2;
  def p0=10*pi/180;
  def x0=15; def y0=0; def l=60;
  setpoint 0 t=0..4:5[
    x0+l*sin(t*p1)/sin(p1)*cos((1-t)*p1-p0),
    y0-l*sin(t*p1)/sin(p1)*sin((1-t)*p1-p0)];
  closedpoly 0-4;
  setdotsize 3 pt; dot 0; setdotsize 4 pt; dot 1;
  setdotsize 5 pt; dot 2; setdotsize 6 pt; dot 3;
  setdotsize 7 pt; dot 4}}
\end{picture}
```

Das Aussehen von Linien

Als Besonderheit kann man bei allen Linien das Aussehen in weiten Bereichen bestimmen. So kann man Liniendicke und Linienmuster frei definieren und hat in der Gestaltung der Linienenden große Auswahl. Dicke, Muster, Art des Linienendes und Pfeilgröße sind mittels der Befehle `setlinewidth`, `setdash`, `setlinecap` und `setarrowsize` leicht zu steuern, wie folgender LaTeX-Code beweist, mit dem Abbildung A.19 erzeugt wurde.

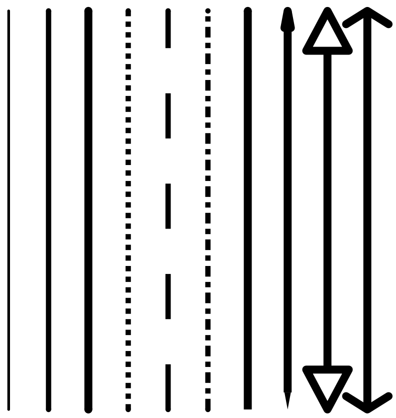


Abbildung A.19: Verschiedene Linientypen

```
\begin{figure}[hbtp]\centering
\setdefaults\setunitlength{1pt}
\begin{picture}(150,150)
\put(0,0){\special{gr setpoint 0[10,0] 1[10,150] 2[25,0]
3[25,150] 4[40,0] 5[40,150] 6[55,0] 7[55,150] 8[70,0]
9[70,150] 10[85,0] 11[85,150] 12[100,0] 13[100,150]
14[115,0] 15[115,150] 16[130,0] 17[130,150] 18[145,0]
19[145,150]}}%
\special{gr setlinewidth 1pt; poly 0-1}
\special{gr setlinewidth 2pt; poly 2-3}
\special{gr setlinewidth 3pt; poly 4-5}
\special{gr setlinewidth 2pt; setdash 2pt; poly 6-7}
```

```
\special{gr setdash 17pt; poly 8-9}
\special{gr setdash 4pt 2pt 2pt 2pt;
  poly 10-11; setdash}
\special{gr setlinewidth 3pt; setlinecap plain round;
  poly 12-13}
\special{gr setlinecap arrow wedge; poly 14-15}
\special{gr setarrowline 3pt; setarrowsize 8pt 15 pt;
  setlinecap triangle triangle; poly 16-17}
\special{gr setlinecap wedge wedge; setarrowsize 8pt 5pt;
  poly 18-19}
\end{picture}
\caption{Verschiedene Linientypen}\label{llines}
\end{figure}
\begin{figure}[hbtp]\centering
```

Bei den Linienenden sind die Möglichkeiten, Pfeilspitzen zu erzeugen, eine besondere Betrachtung wert. Eine Pfeilspitze ist prinzipiell aus drei Bézierkurven aufgebaut, wovon zwei spiegelbildlich die Schenkel des Pfeils bilden und die dritte die Verbindung der Schenkelenden. Bei diesen Bézierkurven liegt jeweils ein doppelter Kontrollpunkt auf halber Länge, dessen Lage sich in Richtung Schwerpunkt des Pfeildreiecks verschieben läßt. Diese Verschiebungsfaktoren werden mittels `setarrowshape` eingegeben. In Bild A.20 sind sechs Linien zu sehen, bei denen die beiden Parameter von `setarrowshape` wie unter dem jeweiligen Pfeil zu sehen gesetzt sind. Die Linienenden sind oben von der Art `triangle` und unten `wedge`.

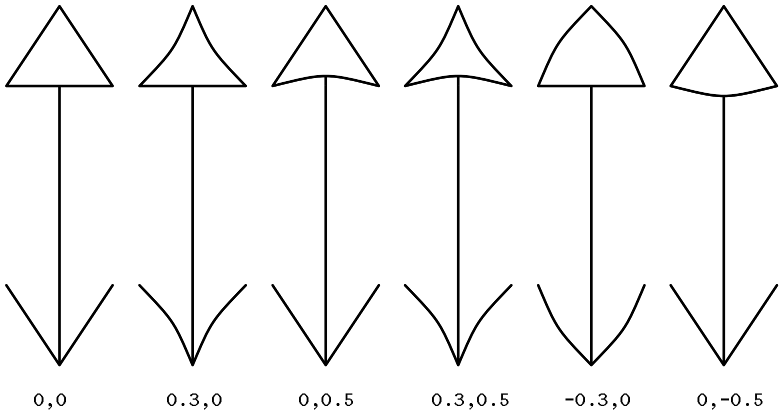


Abbildung A.20: Arten von Pfeilspitzen

Neue Bilder ohne TEX

Üblicherweise gelingen Bilder nicht immer auf Anhieb, sondern man muß im Gegenteil immer wieder kleine und kleinste Änderungen verwirklichen, um ein perfektes Ergebnis zu erreichen. Bei Verwendung der Grafikbefehle der Treiber wie bisher mußte jeweils das Dokument mit der Grafik von TEX neu übersetzt werden, um die Änderungen in der Zeichnung wirksam werden zu lassen. Dieser erhebliche Aufwand läßt sich durch Verwendung des `input`-Befehls umgehen. Durch diesen Befehl werden die Grafikbefehle nicht mehr der DVI-Datei entnommen, sondern aus einer externen Textdatei. Wenn also die Größe und Beschriftung einer Zeichnung feststeht, kann man die übrigen Grafikbefehle für den Treiber in eine Datei auslagern, die über

```
\special{gr input "dateiname"}
```

eingeladen wird. Dadurch sind Änderungen an der Zeichnung durch Ändern der Datei `dateiname` möglich, ohne daß TeX erneut übersetzen muß. Der Dateiname muß natürlich den Konventionen des Betriebssystems Ihres Rechners entsprechen. Die doppelten Anführungszeichen sind Bestandteile des `input`-Befehls und dürfen nicht fehlen.

Ein kleiner Rahmen als Beispiel ist in Abbildung A.21 zu sehen.

```
\setdefaults\setunitlength{1cm}
\begin{picture}(10,5)(0,0)
\special{gr input "bildchen.gr"}
\end{picture}
```

Abbildung A.21: Grafikrahmen im TeX-Dokument

Als kleines Beispiel für eine Grafik könnte Abbildung A.22 dienen. Die Grafikbefehle stehen in einer externen Datei.

```
setpoint 0[0.5,2.5] 1[2.5,2.5] [2.5,3] [0.5,3];
closedpoly 0-3;
setpoint t=pi/8..7/8*pi:5[5+0.5*cos(t),2+0.5*sin(t)];
setpoint t=pi/8..7/8*pi:5[5+cos(t),2+sin(t)];
spline 4-8; poly 8,13; spline 13-9; poly 9,4;
setpoint 14[7.5,0.5] [9.5,0.5] [9.5,1] [7.5,1];
closedpoly 14-17;
setlinecap round arrow; setdash 4pt 2pt;
poly [14,17]1/2, ([0,1]1/2,@), [0,1]1/2 : 0.1..1.8;
defclip; defpattern 170,85,170,85,170,85,170,85;
spline 4-8; poly 8,13; spline 13-9; poly 9,4;
patclip; endclip
```

Abbildung A.22: Grafikbefehle in Datei `bildchen.gr`

Testen Sie doch einmal selbst und probieren Sie eigene Änderungen in der Datei `bildchen.gr` aus!

Anhang B. DISPLAY – Multi-Window-Accessory, Atari

Das `DISPLAY`-Accessory zeigt TEX-Druckseiten an, die mit `DVI` mit Ausgabegerät *File* erstellt wurden. Auf diese Weise ist es bei Editoren mit Menüleiste möglich, Fehler direkt auf dem Bildschirm zu korrigieren, ohne einen Ausdruck zu machen. Man holt sich die Seite mit Fehler in das Fenster, verkleinert das Fenster und zeigt die TEX-Eingabedatei im Editorfenster gleichzeitig an. So kann man ohne `*.LOG` Dateien direkt Fehler korrigieren. Abbildung B.1 zeigt `DISPLAY` und Tempus auf dem Atari ST in voller Aktion.

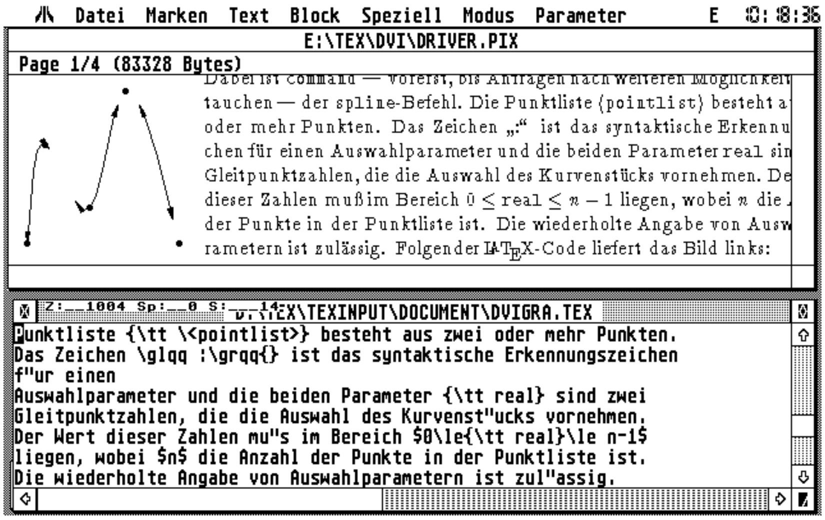


Abbildung B.1: `DISPLAY` und *Tempus* in Aktion

Man beachte allerdings, daß genug Speicherplatz für `DISPLAY` vorhanden sein muß. Bei Editoren wie Tempus, die sich gnadenlos den kompletten zur Verfügung stehenden Speicher reservieren, ist für `DISPLAY` nicht genügend Speicher vorhanden. In diesem Fall muß man den Editor dazu bringen, dem System etwas mehr Speicher zu überlassen. Bei Tempus geschieht dies im Menü *Parameter* durch „Tempus-Parameter“, wo dann der Mindestsystemspeicher auf z.B. 100 000 Byte gesetzt wird. Nicht vergessen, mit „Param. & sichern“ diese Einstellung permanent zu machen!

Nun zur Bedienung von `DISPLAY`. Die Druckseite wird in einem GEM-Fenster dargestellt. Alle üblichen Fensteroperationen bringen auch das gewünschte Resultat, z.B. wird mittels der Slider der gewünschte Teil der Druckseite zur Anzeige gebracht. Alternativ kann man den Fensterinhalt auch mittels der Cursortasten bzw. Shift-Cursortasten verschieben. Mittels Home wird wieder der Seitenanfang angezeigt. Mit + und - kann man seitenweise durch den Text blättern. Mit Return bzw. Enter wird man zur Eingabe einer Seitennummer aufgefordert, die dann direkt angesprungen wird.

Wichtig: `DISPLAY` *muß unbedingt ordnungsgemäß verlassen werden, um „Speicherleichen“ zu vermeiden. Das Programm wird durch Drücken von* [UNDO] *bzw. Anklicken des Closers (oben links in der Fensterkopfzeile) verlassen.*

Bei der Beschreibung haben Sie hoffentlich gemerkt, daß dieses Programm nur auf dem Atari ST implementiert ist. Der IBM PC hat für eine solche Anwendung zu wenig Speicher, besonders unter GEM! Bei den anderen Rechnern entfällt durch die Mehrfenstertechnik der Betriebssysteme die Notwendigkeit für dieses Programm.

Zur Installation muß lediglich die Datei `DISPLAY.ACC` in das Wurzelverzeichnis des Bootlaufwerkes kopiert werden, also üblicherweise nach `C:\` oder `A:\`.

Anhang C. Weitergabe von DVI und DVILW

C.1 Die Programme

Beide DVI-Treiber sind geschützte Programme mit Copyright bei Ingo Eichenseher. Die ausführbaren, *nicht veränderten* Programme dürfen und sollen allerdings für private Nutzung uneingeschränkt *kostenfrei* weitergegeben werden. Wer sich die Quellcodes besorgt hat und veränderte Versionen der Treiber weitergibt, muß das unmißverständlich kennzeichnen, indem z.B. die Treiber umbenannt werden. Trotzdem darf auch in solchen Fällen ein Vermerk auf die ursprüngliche Herkunft und das Copyright der Treiber nicht fehlen. Da es sich jedoch meistens um Verbesserungen bzw. ausgemerzte Fehler handeln dürfte, ist es vorzuziehen, den Autoren die Änderungen zukommen zu lassen, so daß alle davon profitieren können (Adressen siehe unten).

Als Gegenleistung für die Benutzung der Treiber erwarten wir *keine* Sharewaregebühr, sondern lediglich kurze Mitteilungen bei ungewöhnlichen Verhalten der Programme bzw. bei Fehlern. Verbesserungsvorschläge sind natürlich ebenfalls willkommen.

Obwohl sich die Autoren größte Mühe beim Erstellen der Programme und der Anleitung gegeben haben, kann nicht für die Fehlerfreiheit garantiert werden. Die Benutzung der Programme erfolgt deshalb auf eigene Gefahr. Für etwaige Schäden, die durch den Gebrauch der Programme verursacht werden, wird keinerlei Haftung übernommen. Zur Beruhigung ist anzumerken, daß beide Programme auf unseren Systemen seit 3 Jahren problemlos laufen. Es trat zu keiner Zeit ein Datenverlust auf und kein Gerät wurde beschädigt!

Die Quellcodes der DVI-Treiber sind nun frei erhältlich, am Besten per anonymen Filetransfer von der Universität Augsburg:

```
  Adresse: ftp.RZ.Uni-Augsburg.DE (137.250.113.20)
    login: anonymous oder ftp
 password: ID
Directory: /pub/tex/iegw_driver
```

Dort sind auch die TEX-Quellen dieses Handbuchs und die ausführbaren Programme für alle Systeme, auf die wir Zugriff haben, erhältlich. Für Leute ohne Internet-Zugang sind die Bezugsadressen für die Programme unverändert:

Ingo Eichenseher
Am Leiterle 9
8901 Stadtbergen

Gerhard Wilhelms
Josef-Priller-Str. 35
8900 Augsburg 1

Freiumschlag und drei oder vier (Sourcen) formatierte Disketten (HD oder 720KB) und ein Hinweis auf das gewünschte System sind immer eine gute Idee! Da wir allerdings kein Kopierstudio haben, kann sich je nach Andrang die Bearbeitung etwas verzögern.

Als besonderer Service wird testweise eine Hotline angeboten:

0821/43 22 20 (Mo – Fr, 17.00 – 19.00 Uhr)
0821/598 2116 (Mo – Fr, 8.00 – 11.00, versuchweise Wochenende)

Bitte nicht mit dem Anrufen übertreiben und immer daran denken, daß meistens ein Bedienungsfehler vorliegt, bei dem sorgfältiges Lesen der Anleitung die Lösung bringt!

Für die Glücklichen, die Zugang zu einem e-mail Anschluß haben, gibt es die Möglichkeit, Fragen an folgende Adressen zu richten (Internet), allerdings wegen hausgemachter e-mail Probleme in Augsburg nicht ganz zuverlässig:

`wilhelms@Uni-Augsburg.DE`
`ieichens@Appl-Math.Tu-Muenchen.DE`

PS: Die Aufnahme der Programme in einen kommerziellen Public-Domain-Vertrieb bedarf der ausdrücklichen schriftlichen Genehmigung von Ingo Eichenseher.

C.2 Die Anleitung

Diese Anleitung selbst ist `FREEWARE`, d.h. darf jederzeit in unveränderter Form zusammen mit den Treibern `DVI` und `DVILW` weitergegeben werden. Der Text dient gleichzeitig als Tutorial für die Grafikfähigkeiten der Treiber. Falls sich irgendein selbstloser Mensch findet, der bereit ist, diesen Text zu verlegen und verbreiten, ist ihm dies hiermit unter folgenden Auflagen gestattet:

- Der Text bleibt unverändert, wird insbesondere nicht gekürzt.
- Alle Bilder bleiben erhalten, also auch `IMG` und PostScript.
- Die Weitergabe in gedruckter, evtl. gebundener, Form erfolgt zum Selbstkostenpreis. Gewinne, die durch Ansetzen eines Pauschalpreises erzielt werden, müssen gemeinnützigen oder wohltätigen Zwecken zugeführt werden.

Fehlermeldungen und Wünsche in Bezug auf Klarheit der Darstellung oder sinnvolle Ergänzungen sind jederzeit willkommen und können an die Adressen des vorgehenden Abschnitts geschickt werden.