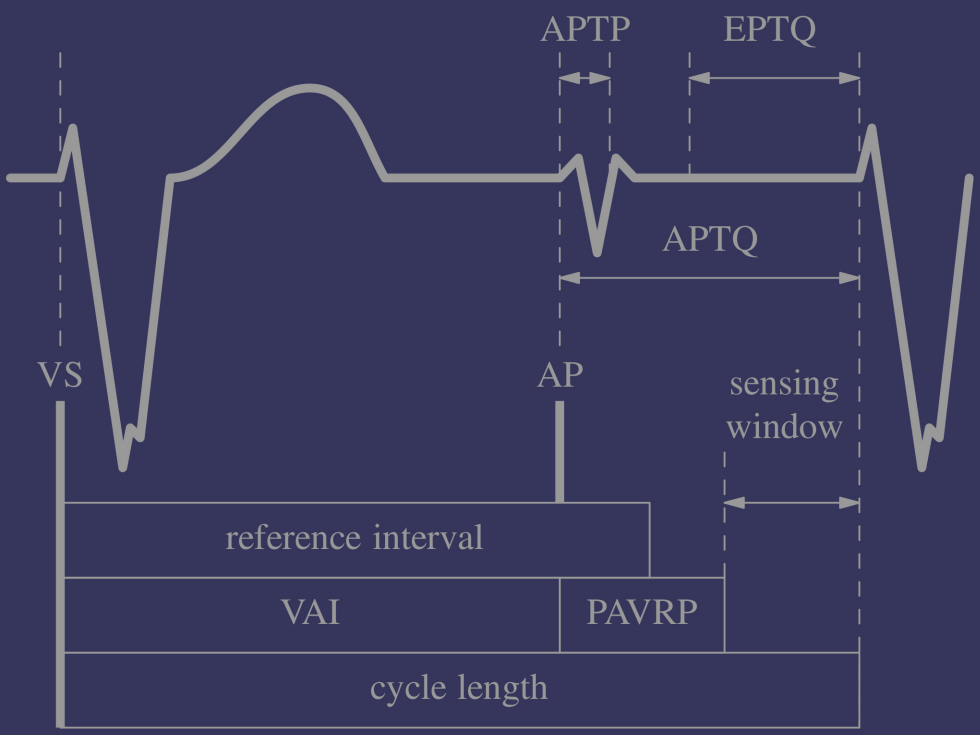
Walter Entenmann

METAPOST

Grafik für TEX und LATEX



lehmanns media

dante

METAPOST

Grafik für TEX und LATEX

Walter Entenmann

München

dante

lehmanns media

© 2017 Walter Entenmann, München
ISBN 978-3-86541-902-6
Umschlag: Herbert Voß
Satz: pdfLaTeX
Verlag: Lehmanns Media GmbH, Berlin (www.lehmanns.de)
Druck: Totem – Inowrocław – Polen
Printed in Poland

Inhaltsverzeichnis

Vorwort

Zeichnungen für Technische Berichte, Reports, Skripten, Vorträge und Veröffentlichungen wurden früher von technischen Zeichnern angefertigt, die eine solide Ausbildung absolviert hatten und alle Regeln und Normen beherrschten. Die meisten dieser großartigen Meister ihres Fachs gibt es nicht mehr, seit der Computer in unser tägliches Leben eindrang. Theo Schreiner, ist einer der letzten, die ich persönlich kennenlernen durfte. Ihm verdanke ich viele gute Ratschläge und Unterstützung.

Heute gibt es viele Zeichenprogramme unterschiedlicher Leistungsfähigkeit und Qualität. Da ich schon einige Erfahrung mit METAFONT von Donald E. Knuth zum Entwurf von Schriftfonts hatte, war der Übergang auf METAPOST von John D. Hobby naheliegend. METAPOST basiert auf METAFONT und unterscheidet sich hinsichtlich der Ausgabe der Bilder im PostScript- statt im Pixel-Format und der zusätzlichen Beschriftung und Verwendung von Farben.

METAPOST ist eine grafische Programmiersprache und ein Compiler/Interpreter zur Erstellung von qualitativ hochwertigen Grafiken im technisch-wissenschaftlichen Anwendungsbereich. Die Sprache beschränkt sich auf die Manipulation der grafischen Grundelemente Punkt, Linie, Fläche und Farbe, aus denen jede Grafik besteht, wie schon Wassily Kandinsky in [13] gezeigt hat. Diese Tatsache ist wichtig, weil sie die Frage, „kann man denn mit METAPOST auch …?“ mit einem simplen „ja“ beantwortet. Aber nicht in dem Sinne, dass mit einem einfachen Klick ein Wunder geschieht, sondern man muss schon alles im Detail selbst programmieren. Daher wird METAPOST oft als schwierig empfunden, wegen der formalen Syntax, der Beschreibung im METAFONTbook und in der Dokumentation.

Der Einstieg war auch für mich etwas schwierig, aber der Lernprozess schreitet rasch voran mit jeder neuen Anwendung und Herausforderung, für die mein Sohn Andreas

laufend sorgte, sodass mit der Zeit ein kleines „METAPOST-Sammelsurium“ entstand aus vielen praktischen Konstrukten und Makrodefinitionen, das den Grundstock für dieses Buch bildete.

Mein Ziel ist daher, aus der Sicht des Anwenders zu zeigen, wie man Zeichnungen anhand praktischer Vorgaben Schritt für Schritt systematisch und mit hoher Qualität entwerfen kann. Vielleicht kann es auch dazu beitragen, ein wenig von dieser Erfahrung weiterzugeben, den einen oder anderen Irrweg und einige Fallstricke zu vermeiden, damit das Arbeiten mit METAPOST von Anfang an wirklich Freude macht. Für Kritik und Verbesserungsvorschläge wäre ich sehr dankbar.

Ein großes Verdienst am Zustandekommen des Buches hat Herbert Voß, der für die Aufnahme in die DANTE-Edition im Verlag Lehmanns Media sorgte und mich in jeder Hinsicht mit Rat und Tat unterstützte und ermunterte.

Danken möchte ich auch den TEXis und Kollegen in nah und fern, die mir in großzügiger Weise erlaubt haben, Ihre Arbeiten im Buch zu beschreiben: Nicky van Foreest, Dag Langmyhr, Berndt E. Schwerdtfeger, Stephan Hennig. Computern ist eine Kunst, in die mich Oliver Riesener einführte und mich mit unendlicher Geduld unterstützt, als begnadeter Guru und Ratgeber.

Meiner Frau Gabriele danke ich für die viele Zeit, die sie mir geschenkt hat, um an dem Buch zu arbeiten.

München, im Herbst 2016 Walter Entenmann

Kapitel 1

Einleitung

METAPOST (MP) ist eine grafische Programmiersprache und ein Compiler/Interpreter mit PostScript-Ausgabe (Vektorgrafik). Die Sprache ist nicht imperativ sondern deklarativ und ermöglicht, die wesentlichen Komponenten von bestimmten benötigten grafischen Formen geeignet anzuordnen, zu zeichnen und mit LaTeX zu beschriften.

Historie von MP

1984 veröffentlichte Donald E. Knuth das Programmsystem METAFONT (MF) zum Entwurf von Schriftfonts, hervorragend dokumentiert im METAFONTbook [26]. MP wurde von John D. Hobby etwa ab 1990 auf der Basis von MF Version 1.9 bei AT&T Bell Laboratories entwickelt, nachdem er zuvor als Doktorand von Professor Knuth an der Stanford University bereits wesentlich zur Entwicklung von MF beigetragen hatte. Nach 1995 wurde MP frei zugänglich. Nachdem sich über lange Zeit zahlreiche Fehler angesammelt hatten, übernahm 2006 Taco Hoekwater die Betreuung und Weiterentwicklung von MP. Neben der Fehlerbeseitigung, erweiterte er MP um eine Library, eine dynamische Speicherverwaltung und um den Übergang von Pascal WEB auf CWEB, endend in Version 1.5. Ab 2009 folgte als weitere Neuerung eine wesentliche Verbesserung der numerischen Genauigkeit. Außer der bisherigen 32-Bit-Festpunktarithmetik steht nun wahlweise auch eine 64-Bit-Gleitpunktarithmetik (IEEE Floating Point Arithmetic) und bei Bedarf eine Arithmetik auf binärer oder dezimaler Basis mit fast beliebig wählbarer Stellenzahl zur Verfügung. Dies führte zu der in Kürze verfügbaren Version 2.0, sodass mit MP ein modernes, sehr stabiles Grafiksystem von hoher Qualität zur Verfügung steht. Die weitere Betreuung von MP hat jetzt Luigi Scarso übernommen.

Die Homepage von MP ist `tug.org/metapost`. Dort findet man auch die Dokumentation [21] und viele interessante Veröffentlichungen, Tutorials [15, 16] und umfangreiche

Beispielsammlungen [35, 43], deren Lektüre ich sehr empfehlen kann. Viele Anregungen enthält auch das MetaFun-Manual von Hans Hagen [12]. Die Dokumentation [21] steht auch lokal mit `texdoc metapost` zur Verfügung.

Viele nützliche Zusatzpakete findet man auf dem CTAN-Server:
`CTAN graphics/metapost/contrib/macros/`

Die derzeitige Entwickler-Homepage von MP ist
`foundry.supelec.fr/projects/metapost/`.

Mailing-List: `tug.org/mailman/listinfo/metapost`

Eigenschaften von MP

MP und MF unterscheiden sich vor allem hinsichtlich des Ausgabeformats: MF gibt das Ergebnis im Raster-Format (GF, *generic font*) aus, MP dagegen als Vektorgrapfik im PostScript-Format. Außerdem wurde MP um eine komfortable Beschriftung (*labeling*) mit TEX/LATEX-Text und die Verwendung von Farben erweitert. Die Zeicheneinheit ist nicht ein Pixel sondern ein *big point* bp (PostScript-Punkt). Ferner gibt es in MP gestrichelte Linien und Pfeile sowie höhere bildverarbeitende Operationen wie `clip`, `image`, `buildcycle`, `thelabel`.

MP kennt wie jede Programmiersprache arithmetisch-logische Ausdrücke und elementare mathematische Funktionen sowie Kontrollstrukturen zur Steuerung des Programmflusses. Die besondere Leistungsfähigkeit als grafische Programmiersprache liegt jedoch in der Bereitstellung von speziellen Variablentypen zur adäquaten Definition und Beschreibung der grafischen Grundelemente Punkt, Linie, Fläche und Farbe und entsprechender Operationen zur Transformation, Platzierung und zum Zeichnen dieser Objekte. Sehr wichtig ist dabei, dass alle schrittweise gezeichneten Bildteile gespeichert bleiben bis zur endgültigen Fertigstellung einer Zeichnung. Dadurch kann man stets auf alle Bildteile zugreifen, z. B. um Schnittpunkte zu berechnen oder Beschriftungen exakt zuzuordnen. Eine weitere Stärke von MP ist die sehr leistungsfähige Definition von Makros (*subroutine*) zur flexiblen Erweiterung der Funktionalität. Von besonders hoher Qualität ist auch das von Donald E. Knuth und John D. Hobby speziell entwickelte Interpolationsverfahren zum Zeichnen gekrümmter Kurvenverläufe.

Einbetten von MP-Code in ein LATEX-Dokument

Zum Einbetten von MP-Code in ein LATEX-Dokument gibt es verschiedene Möglichkeiten. Im folgenden geben wir eine Übersicht über einige Pakete, ohne Anspruch auf Vollständigkeit.

mpgraphics Das Paket stammt von der Persischen TEX User Group
`github.com/persian-tex/mpgraphics`
und ist auch auf CTAN erhältlich.
Es verwendet LATEX mit der Option `-shell-escape` und benötigt das Paket mpgraphics. Die Umgebung `btex etex` zur Formatierung von TEX/LATEX-Text wird nicht ausgewertet.

lualatex, luamplib Das Paket wurde an der University of California Berkeley entwickelt.

`math.berkeley.edu/computing/wiki/index.php/latex_sample_embedded_metapost`.
Das LaTeX-Dokument wird mit `lualatex` bearbeitet. Die Umgebung `btex etex` wird nicht ausgewertet (das Paket enthält einen kleinen Bug, Abhilfe ist angegeben).

emp [34]. Das Paket *EMP – Encapsulated Metapost for LaTeX* stammt von Thorsten Ohl, und ist auf CTAN erhältlich. Dies ist wohl das leistungsfähigste der betrachteten Pakete, es erlaubt die Verwendung von TeX/LaTeX-Text und insbesondere auch des MP-Pakets graph.mp.

ConTeXt Dieses System von Hans Hagen kann MP-Code in das LaTeX-Dokument einbetten.

Insgesamt scheint das Problem noch im Stadium der Entwicklung und nicht vollständig gelöst zu sein. Insbesondere die Beschriftung mit LaTeX-Text und die Verwendung des Pakets graph und anderer Zusatzpakete (epsincl, latexmp) und Tools (mplatex, `gawk`) ist problematisch, da diese Pakete zum Teil die Präambel eigenständig organisieren und eventuell zweimal mit MP bearbeitet und einzelne Bild-Files mit `gawk` nachbearbeitet werden müssen. Für fortgeschrittene Anwendungen ist die Einbettung oft nicht flexibel genug.

Meiner Erfahrung nach ist die Integration von MP-Code in ein LaTeX-Dokument jedoch nicht unbedingt erforderlich, weil der Entwurf eines Bildes ein sehr langer und aufwändiger Prozess ist, der besser, wie auch früher schon üblich, separat organisiert werden sollte. Oft wird diese Arbeit von einem Mitarbeiter erledigt, der auch für ein einheitliches Erscheinungsbild aller Zeichnungen sorgt. Die Bilder sollten sorgfältig archiviert werden, unabhängig von ihrer aktuellen Verwendung im Dokument, für eine spätere mehrfache Nutzung, Anpassung und Korrektur. Bei etwas komplizierteren Bildern wird der MP-Code sehr umfangreich sein, sodass man in dem LaTeX-Dokument schnell den Überblick verliert, d. h. bei großem Textumfang mit vielen Bildern wird man Text und Bilder besser getrennt organisieren. Somit kann das Einbetten allenfalls für kleine Dokumente mit wenigen einfachen Bildern in Frage kommen.

Verlage von Fachzeitschriften verlangen die Bilder oft als separate, selbständige (*standalone*) eps- oder pdf-Files.

Man muss jedoch sicherstellen, dass die Bilder und MP-Listings im Dokument mit den entworfenen Grafiken stets konsistent bleiben. Für die Erstellung dieses Buches habe ich die folgende Vorgehensweise und File-Organisation verwendet, die auch auf andere Projekte sinngemäß übertragbar ist.

File-Organisation

Das LaTeX-Hauptdokument mit seinen Kapitelfiles, Stilfiles und der Bibliografie befindet sich im Verzeichnis `buch` Es wird mit `pdflatex`, `bibtex`, `makeindex` bearbeitet und mit einem PDF-Viewer angezeigt und ausgedruckt.

Die Unterverzeichnisse `list`, `fig`, `pdf`, `code` und `defs` dienen zum Entwurf der Bilder in `list` und zur Archivierung in `fig`, `pdf` und `code`. Das Verzeichnis `defs` ent-

Tabelle 1.1: File-Organisation.

		buch/		
		LaTeX-Hauptdokument `book.tex`		
		Stilfiles		
		Kapitel-Files		
		Bibliografie-File		
list/	**fig/**	**pdf/**	**code/**	**defs/**
Arbeitsverz. für Bilder		Bilder		Makros
mp-Files	eps-Files	pdf-Files	MP-Code	
Shell-Skript	`<mpfile>.eps`	`<mpfile>.pdf`	`<mpfile>.mp`	`def_*.mp`
Präambel				
Postambel				
`dummy.tex`				
Datenfiles				
Programme				

hält die Makrosammlung. Der Entwurf der Bilder erfolgt ausschließlich im Arbeitsverzeichnis `list` mit Hilfe eines Shell-Skripts, das die Bearbeitungsschritte und Archivierung übernimmt. Der eigentliche Entwurfsprozess ist ein Wechsel zwischen Editor und diesem Shell-Skript. Alle Hilfsfiles wie `preamble.mp`, `postamble.mp`, `dummy.tex`, Datenfiles der Beispiele etc., die das Shell-Skript auch benötigt, werden im Verzeichnis `list` vorgehalten. Zur Sicherstellung der Konsistenz, werden die Bilder und ihre MP-Listings in das Hauptdokument beim LaTeX-Lauf mit den Befehlen `includegraphics` und `lstinputlisting` aus den Unterverzeichnissen `fig`, `pdf` bzw. `code` automatisch eingebunden.

Gliederung des Buches

Entsprechend meiner Absicht, ein Buch für den praktischen Gebrauch von MP zu schreiben, beginnt es mit einem Schnelleinstieg für den ungeduldigen Leser, um die grundsätzlichen Arbeitsschritte auszuprobieren und die Freude am ersten eigenen Bild zu erleben. Mit diesem Wissen kann man in Kapitel 3 bereits einige interessante Kurven zeichnen. Etwas grundsätzlichere Überlegungen über Technisches Zeichnen, Maße, Normen und Typografie folgen in Kapitel 4. Für das praktische Arbeiten mit MP erweitern wir in Kapitel 5 die eingangs gezeigten Arbeitsschritte zu einer generellen Arbeitsanweisung (Workflow) und stellen den schematischen Aufbau der MP-Quellfiles als Musterdateien (Templates) zusammen. Dies ermöglicht die sichere Verwendung von TeX/LaTeX-Text zur Beschriftung unserer Zeichnungen in der richtigen Schriftgröße und mit dem gewünschten Schriftfont einheitlich für das ganze Bild. Die Einzelheiten werden aber erst in Kapitel 10 und am Ende von Kapitel 13 voll verständlich werden. Bis dahin sind der vorgestellte Workflow und die Templates einfach als Rezepte zu betrachten, um überhaupt erst mal solide arbeitsfähig zu werden. Von besonderer Bedeutung ist die Erzeugung von systemunabhängigen, selbständigen (*stand-alone*) eps- und pdf-Files für die Bilder mit korrekter Boundingbox. Das Shell-Skript fasst alle Bearbeitungsschritte zusammen und erleichtert damit das Arbeiten erheblich. Die folgenden sieben Kapitel

behandeln die Eigenschaften der Programmiersprache MP und sind dem syntaktischen Aufbau der Sprache gewidmet. Schwere Kost, die nicht zum sofortigen Verzehr gedacht ist. Beim ersten Lesen sollte man sich nicht entmutigen lassen und sich einfach mal mit den Begriffen und Befehlen vertraut machen, damit man sich bei späterem Bedarf an das eine oder andere erinnert und dann gezielt das Benötigte nachlesen kann. Vielleicht können die eingestreuten Beispiele das Problem ein wenig mildern. Im einzelnen geht es in Kapitel 6 um die Bildung von Variablennamen, die in MP die gleiche Rolle spielen wie in der Algebra. In Kapitel 7 werden dann die zehn verschiedenen Variablentypen und ihre Verknüpfungen besprochen, die den Kern der Ausdrucksmöglichkeiten von MP darstellen. Den grafischen Grundelementen Punkt, Linie, Teilbild, Farbe entsprechen die Variablentypen `pair`, `path`, `picture`, `color`, `cmykcolor` und zum Zeichnen `pen` als Feder. Zahlenwerte und Abmessungen wie Länge und Breite entsprechen dem Typ `numeric`. Zur Platzierung von Bildteilen durch Skalierung, Verschiebung und Rotation dienen Transformationen vom Typ `transform`. Die restlichen beiden Typen `boolean` und `string` gibt es in jeder Programmiersprache zur Formulierung von Bedingungen und zur Verarbeitung von Zeichenketten.

MP kennt wie jede Programmiersprache Kontrollstrukturen für Wiederholungen und Verzweigungen zur Steuerung des Programmflusses (Kapitel 8). Die wichtigsten mathematischen Rechenoperationen und elementaren Funktionen, die Rundungs- und Wandeloperationen sind in Kapitel 9 tabellarisch zusammengestellt und erläutert. Kapitel 10 behandelt die Beschriftung unserer Zeichnungen mit dem Label-Befehl, einschließlich der Besonderheit der dynamischen Beschriftung. Die Thematik wird am Ende von Kapitel 13 nochmals zusammenfassend aufgegriffen. Der Datenaustausch mit der Außenwelt, also das Lesen und Schreiben von Daten von und auf externe Files, Bildschirm und Tastatur wird in Kapitel 11 beschrieben. Den Abschluss dieses Buchteils bildet Kapitel 12, das die Definition und Verwendung von Makros behandelt, die eine sehr leistungsfähige Erweiterung der Funktionalität von MP darstellen. Mit Kapitel 13 beginnt der praktische Teil des Buches. Da die meisten Zeichnungen, die wir benötigen werden, Diagramme sind, welche Funktionsverläufe darstellen, spielt das Paket `graph.mp` von John Hobby eine zentrale Rolle. Es ist in der Lage, mit einer einzigen Anweisung die Daten aus einem externen Datenfile einzulesen und daraus völlig selbständig den Kurvenverlauf in ein komplettes Diagramm mit Skalierung, Achsen, Ticks, Bezifferung und Beschriftung zu zeichnen. Wir werden daher dieses leistungsfähige Paket auf zahlreiche Beispiele anwenden, um die Anpassung an unterschiedliche Forderungen zu demonstrieren. Das Paket graph lädt und verwendet auch das Paket format, das die Formatierung von Zahlenwerten in Fest- und Gleitpunktdarstellung mit gewünschter Stellenzahl ermöglicht. Der entsprechende Befehl `format` beeinflusst also auch die Beschriftung unserer Zeichnungen, deshalb erfolgt erst hier die vollständige Beschreibung des Beschriftungsproblems. In Kapitel 14 lernen wir von den großen Meistern ihres Fachs, wie man qualitativ hochwertige Diagramme entwirft und welche goldenen Regeln dabei beachtet werden sollten. Im folgenden Kapitel konzentrieren wir uns auf weitere häufig verwendete Formen von Diagrammen wie Block-, Struktur-, Fluss- und Balkendiagramme, die Darstellung statistischer Daten durch Box-Plots, Scatter-Plots, Regressionsgeraden und ROC-Kurven. Weiterhin wird gezeigt, wie man externe Bilder im eps-Format in MP einfügt, einen Text entlang eines Pfades schreibt und Pfeilspitzen auf Pfade setzt

sowie weitere praktische Konstrukte wie Superellipse, geschweifte Klammern und das Lot auf eine Gerade. Das letzte Kapitel zeigt an einer etwas umfangreicheren Problemstellung, der perspektivischen Darstellung von dreidimensionalen Objekten, die besondere Leistungsfähigkeit von MP, auch mathematisch anspruchsvolle Zusammenhänge zu formulieren und durch Zusammenfassung der einzelnen Schritte in Makros ein komplexes Problem praxisgerecht und kompakt zu lösen. Die Anwendung der Orthogonalprojektion auf die Erdkugel folgt im wesentlichen der exzellenten Arbeit von Berndt E. Schwerdtfeger. Das letzte Beispiel zeigt im Rahmen der konformen Abbildung die dreidimensionale Darstellung des Betrags einer komplexwertigen Funktion als Relief.

Dem Leser verbleibt die Aufgabe, die im Verlaufe des Buches entwickelten Makros im Unterverzeichnis `defs` und die Hilfsfiles wie `preamble.mp` im Unterverzeichnis `list` zu sammeln, um die Beispiele selbst ausprobieren zu können.

Kapitel 2

Schnelleinstieg in MP

Dieses Kapitel soll dem ungeduldigen Leser, der noch nie mit MP gearbeitet hat, aber Erfahrung mit dem TEX-System hat und mit der Eingabe von Befehlen auf der Kommandozeile vertraut ist, rasch eine Vorstellung verschaffen, wie das Grafikprogramm MP funktioniert. Wir gehen davon aus, dass auf dem Rechner die entsprechende TEX Live-Distribution installiert ist.

Als erstes erstellt man mit einem Editor (z. B. `emacs`) ein frei benennbares MP-Quellfile (hier `quick.mp`)

```
emacs quick.mp
```

und trägt folgende Zeilen ein

```
beginfig(1) % Mein erstes Bild mit MP
  draw (0,0)--(30,20);
endfig;
end
```

Dieses File bearbeiten wir mit MP

```
mpost quick
```

und erhalten als Ergebnis ein File namens `quick.1`. Zur Anzeige der Grafik am Bildschirm fügen wir das File `quick.1` in ein LATEX-Dokument ein, z. B. `quick.tex`, das wir ebenfalls mit dem Editor erstellen

```
emacs quick.tex
```

mit folgendem Inhalt

```
\documentclass{article}
\usepackage{graphicx}
\begin{document}
  \includegraphics{quick.1}
\end{document}
```

Dieses bearbeiten wir wie üblich mit

```
latex quick
dvips quick
```

und betrachten das Ergebnis mit einem PostScript-Previewer (z. B. GhostView, `gv`). Unter Mac OSX muss man `gv` durch `open` ersetzen, weil GhostView dort nicht existiert und stattdessen 'Vorschau' verwendet wird.

```
gv quick.ps
```

Unsere erste mit MP erstellte Zeichnung erscheint am Bildschirm wie in Bild 2.1 gezeigt.

Bild 2.1: Anzeige der Zeichnung am Bildschirm.

Kapitel 3

Kurven

Da wir jetzt schon mal wissen, wie man MP benutzt, wollen wir gleich mal zur Übung ein paar Linienzüge (Kurven) zeichnen.

3.1 Quellfiles

Die prinzipielle Struktur des MP-Quellfiles `pic.mp` besteht aus einer Folge von numerierten Umgebungen `beginfig` `endfig` und dem Abschluss mit dem Befehl `end`.

```
beginfig(1) % Erstes Bild
  <MP-Zeichenbefehle>
endfig;
beginfig(2) % Zweites Bild
  <MP-Zeichenbefehle>
endfig;
beginfig(<n>) % n-tes Bild
  <MP-Zeichenbefehle>
endfig;
end
```

Entsprechend benötigen wir dazu ein kleines LaTeX-File `pic.tex` der Form

```
\documentclass{article}
\usepackage{graphicx}
\parindent0pt
\pagestyle{empty}
\begin{document}
  \includegraphics{pic.1}
  \newpage
  \includegraphics{pic.2}
% ...
  \newpage
  \includegraphics{pic.<n>}
\end{document}
```

Erläuterung der Symbole und Bezeichnungen

< > Angaben in spitzen Klammern sind **Platzhalter**, symbolische Variable, die durch konkrete Zahlen, Texte oder Anweisungen zu ersetzen sind.

% **Kommentare** werden mit einem Prozentzeichen eingeleitet. Die danach folgenden Zeichen bis zum Zeilenende werden von MP ignoriert.

; Jede MP-**Anweisung** wird mit einem Strichpunkt abgeschlossen.

3.2 Federn

Zum Zeichnen benötigt man ein Zeichengerät, einen Stift, eine Feder (`pen`). MP verwendet als Voreinstellung `defaultpen`, eine Feder mit einer runden Federspitze vom Durchmesser 0.5 bp, die wir auch selbst mit der Anweisung

```
pickup pencircle scaled 0.5bp;
```

hätten erzeugen können. Nach jedem `beginfig` wird die Feder auf die Voreinstellung `defaultpen` gesetzt. Die eingestellte Feder bleibt solange in Gebrauch, bis sie durch einen erneuten Befehl `pickup` geändert wird. Um wieder auf die Voreinstellung zurückzusetzen, schreibt man

```
pickup defaultpen;
```

Beispiel:

```
beginfig(1)
  pickup pencircle scaled 4mm;
  draw origin;
  draw (0,7mm)--(0,17mm);
endfig;
```

Der Punkt des Rufzeichens in Bild 3.1a entspricht dem Querschnitt der Federspitze.

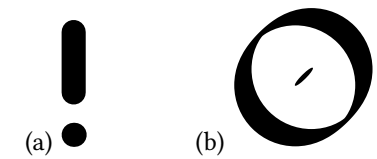


Bild 3.1: (a) Beispiel Rufzeichen, (b) Beispiel kalligrafisches 'O'.

Mit der Option `withpen` kann man bei einem Zeichenbefehl eine individuelle Feder angeben, ohne die eingestellte Feder zu verändern, z. B.

```
draw (0,0)--(30,20) withpen pencircle scaled 1bp;
```

Spezielle **kalligrafische Effekte** kann man erzielen, wenn man eine Feder mit elliptischem Querschnitt und geneigter Hauptachse wählt

```
beginfig(1)
  pickup pencircle xscaled 4mm yscaled 0.8mm rotated 45;
  draw origin;
  draw fullcircle scaled 2cm;
endfig;
```

Der „Punkt" im Innern des Buchstabens 'O' in Bild 3.1b zeigt den schräggestellten elliptischen Querschnitt der Federspitze.

3.3 Punkte und Linien

Zur Festlegung von **Punkten**, als einfachstem grafischem Grundelement, verwendet man ein kartesisches **Koordinatensystem** mit einer waagrechten x-Achse und einer dazu senkrechten y-Achse wie in Bild 3.2 gezeigt. Der Schnittpunkt ist der Nullpunkt. Der Punkt (x, y) = (30,20) liegt also vom Nullpunkt 30 Einheiten nach rechts und 20 Einheiten nach oben.

Ohne weitere Angaben bei den Zahlenwerten handelt es sich um Vielfache der **internen MP-Zeicheneinheit** *big point* (PostScript-Punkt) mit 1 bp = 1/72 in = 25.4 mm/72 = 0.3527 mm.
Selbstverständlich kann man den Zahlenwerten auch eine **Einheit**, z. B. cm hinzufügen gemäß (x, y) = (1.5cm,2cm), was im allgemeinen der Fall sein wird.

Es ist sehr empfehlenswert, für jedes Bild eine zweckmäßige **Zeicheneinheit** festzulegen, z. B. ut als Abkürzung für unit

```
ut := 1mm;
```

und alle Maße als Vielfache dieser Einheit auszudrücken gemäß (x, y) = (15ut,20ut). Dann lässt sich die gesamte Zeichnung durch Änderung eines einzigen Zahlenwerts bei der Variablen `ut` nach Bedarf **skalieren**.

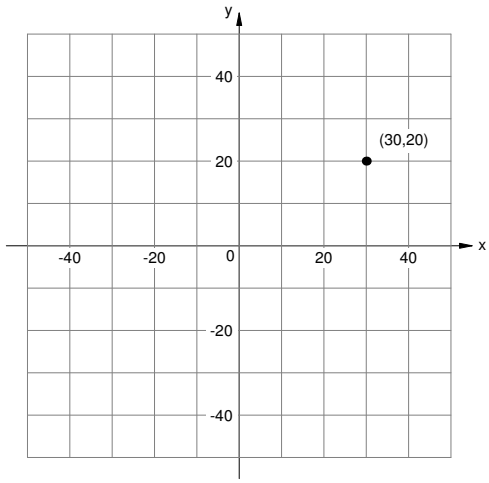


Bild 3.2: Koordinatensystem.

Für Punkte benützen wir zunächst am einfachsten die in MP vordefinierten **z-Variablen** $z_i = (x_i, y_i)$, $i = 0, 1, \ldots$. Diese dürfen nur mit dem Gleichheitszeichen (=) verwendet werden, eine Wertzuweisung mit (:=) ist nicht erlaubt. Nach jedem `beginfig` werden alle z-Variablen zurückgesetzt. Vorsicht: z=(1,1); ...z=(0,5); ergibt einen Fehler, da in einer Gleichung die linke und rechte Seite übereinstimmen müssen.

Zum Zeichnen von **Linien** dient der Zeichenbefehl `draw` gefolgt von einer Anzahl von Punkten (x_i, y_i), die mit `--` geradlinig oder mit `..` gekrümmt als glatte Kurve verbunden werden

```
draw (x0,y0)..(x1,y1)    ..(x[n],y[n]);
```

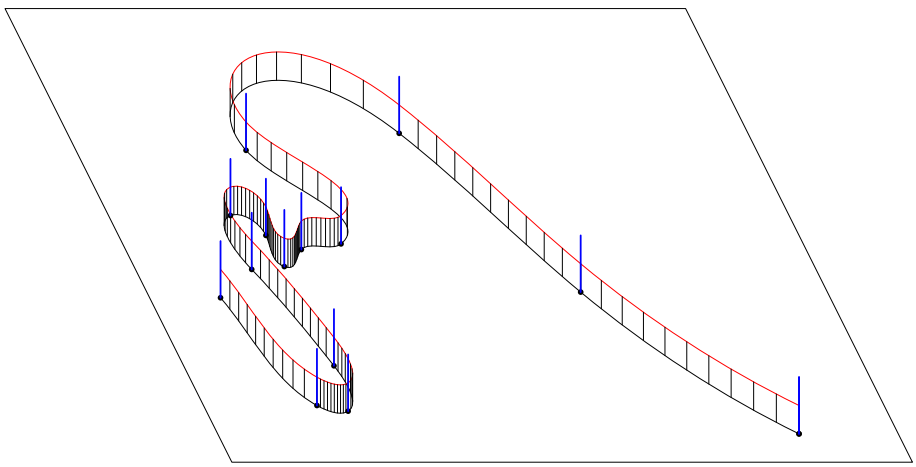


Bild 3.3: Federstahlband als Modell der Interpolation.

Gekrümmte Kurven sind von einer anderen Qualität als Polygonzüge. Um durch vorgegebene Punkte eine glatte, formschöne, harmonische Kurve zu zeichnen, entwickelten Donald E. Knuth und John D. Hobby ein sehr leistungsfähiges mathematisches **Interpolationsverfahren**, das u. a. die besondere Qualität des Grafiksystems MP/MF auszeichnet. Die Form der durch Interpolation gewonnenen gekrümmten Linien hat große Ähnlichkeit mit einem durch die Stützpunkte gezogenen Federstahlband, wie in Bild 3.3 gezeigt, d. h. gekrümmte Kurven haben eine innere Spannung und durch Zwangskräfte vorgegebene Richtungen. Beide Eigenschaften können in MP wie folgt formuliert werden.

Die **Spannung** des Kurvenverlaufs zwischen zwei Punkten wird verändert, indem man statt `..` den Befehl

```
..tension <t>..
```

einfügt. Der Spannungsparameter t ist ein Zahlenwert ≥ 0.75, der Wert 1 entspricht der Voreinstellung. Je größer der Wert, desto größer wird die Spannung.

Jedem Punkt kann man in geschweiften Klammern eine **Ein- und Ausfallsrichtung** voran- bzw. nachstellen

```
..{<Einfallsrichtung>}z{<Ausfallsrichtung>}..
```

Fehlt eine der beiden Angaben, sind die beiden Richtungen gleich der angegebenen. Diese Richtungen haben verschiedene Darstellungsformen

```
{right} {left} {up} {down} {z} {x,y} {1,m} {z2-z1} {dir alpha}
```

wie wir in den folgenden Beispielen sehen werden.

(a) (b) (c)

Bild 3.4: (a) Spannung, (b) Richtung, (c) Sinus.

Bild 3.4a zeigt den Kurvenverlauf

```
z1=origin; z2=(5cm,0);
draw z1{up}..tension <t>..{down}z2;
```

für verschiedene Spannungswerte t = 0.75, 1, 1.5, 2.5, 5 von oben nach unten.

Die Linie in Bild 3.4b beginnt und endet jeweils in Richtung 60°

```
draw z1{dir 60}..{dir 60}z2;
```

Bild 3.4c wurde erzeugt mit

```
pi:=3.1415;  a:=2.5cm; z1=origin; z2=(pi/2,1);
draw (z1{1,1}..{right}z2) scaled a;
```

Der Verlauf ist in guter Näherung eine Viertelperiode der Sinusfunktion. Zum Vergleich ist auch der exakte Wert für $\sin 60°$ eingetragen.

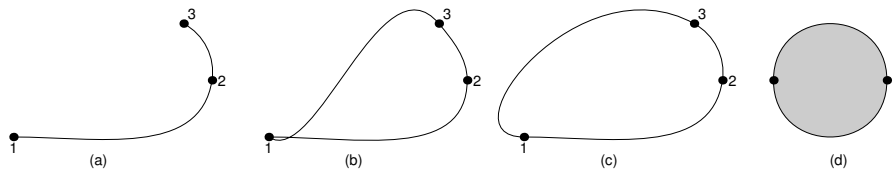


Bild 3.5: (a) Offener Pfad, (b) offener Pfad mit Endpunkt gleich Startpunkt, (c) geschlossener Pfad, (d) farbige Fläche.

Ein Linienzug ist im allgemeinen **offen**, auch wenn Anfangs- und Endpunkt übereinstimmen. Um ihn zu schließen, muss man explizit den Befehl `cycle` anfügen. Zur Demonstration zeichnen wir drei getrennte Bilder mit den gleichen Werten

```
ut:=1mm; z1=origin; z2=(35ut,10ut); z3=(30ut,20ut);

% (a)
draw z1{right}..z2..z3;

% (b)
draw z1{right}..z2..z3..tension 1.5..z1;

% (c)
draw z1{right}..z2..z3..tension 1.5..cycle;
```

Das Ergebnis zeigt Bild 3.5a–c. Man beachte in Teilbild (b) die Ecke im Punkt 1. Hier ist zwar der Endpunkt gleich dem Startpunkt, aber trotzdem handelt es sich um einen offenen Pfad, weil er nicht mit dem Befehl `cycle` abgeschlossen ist. In Teilbild (c) ist der Verlauf an dieser Stelle glatt und ohne Knick.

Geschlossene Pfade begrenzen eine **Fläche**, die auch mit einer **Farbe** gefüllt werden kann, indem man statt `draw` den Befehl `fill` verwendet und die Option `withcolor` für die Farbe anfügt, z. B.

```
z1=(10mm,0); z2=(-10mm,0);
fill z1..z2..cycle withcolor 0.8white;
draw z1..z2..cycle;
```

Teilbild 3.5d zeigt das Ergebnis. Man beachte die erstaunliche Leistungsfähigkeit des Interpolationsverfahrens, aus der Vorgabe von nur zwei Punkten einen fast idealen Kreis zu zeichnen! Der zusätzliche Befehl `draw` zeichnet die Randlinie. Wenn man die Reihenfolge von `fill` und `draw` vertauscht, wird die Randlinie zur Hälfte überdeckt und erscheint nur halb so dick.

Um dies besser zu verstehen, ist es hilfreich, sich Punkte und Linien als *mathematische Objekte der Geometrie* ohne Ausdehnung und damit unsichtbar vorzustellen. Sehen kön-

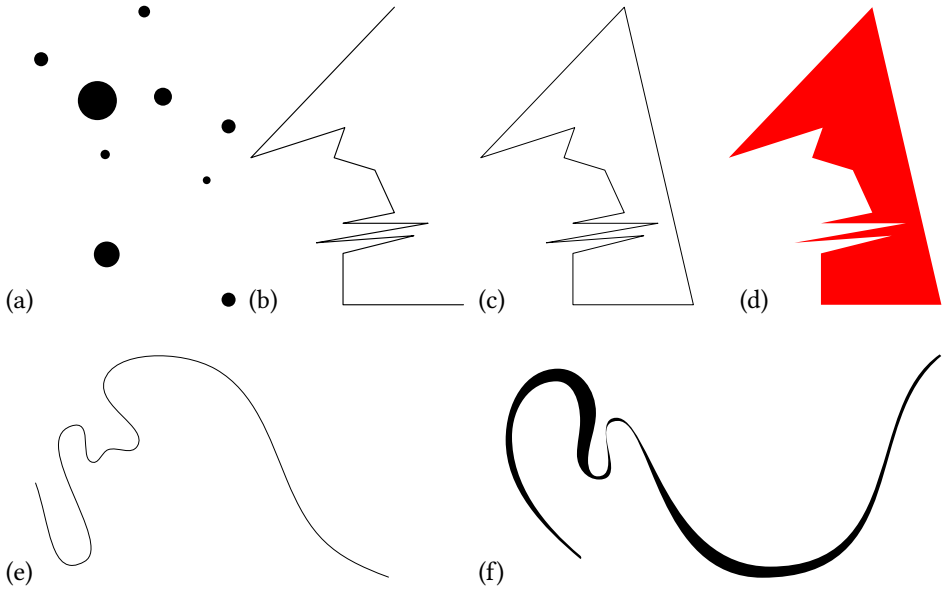


Bild 3.6: (a) 9 Punkte im Aufstieg, (b) freie vieleckige Linie, offen, (c) geschlossen (d) mit Farbe gefüllt, (e) gebogene Freiwellenartige, (f) freie Wellenartige mit Nachdruck – horizontale Lage, nach Kandinsky.

nen wir nur Flächen, die mit Farbe (meist schwarz) gefüllt sind. Sichtbar werden Punkte und Linien daher erst, wenn wir sie mit einer 'dicken' Feder nachziehen. Eine Fläche hat stets eine unsichtbare Randlinie. Die Befehle `fill` und `unfill` füllen oder leeren das Innere dieser Randlinie. Ziehen wir zuerst mit einer dicken Feder die Randlinie nach und leeren anschließend die Fläche, bleibt nur die äußere Hälfte des Federzugs übrig (siehe Bild 3.7b).

Das Nachziehen mit einer dicken runden Feder hat auch zur Folge, dass die äußeren Ecken eines Quadrats abgerundet sind. Spitze Ecken erhält man jedoch mit

```
linejoin:=mitered
```

wie das Beispiel in Bild 3.7a zeigt.

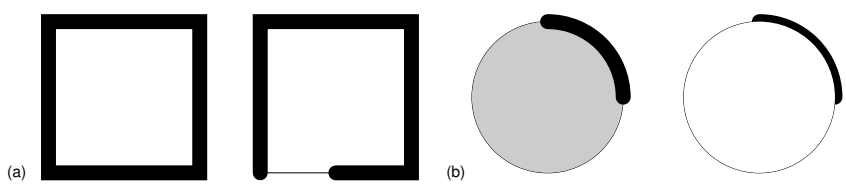


Bild 3.7: (a) Spitze Ecken, (b) Federstriche und geometrische Linien.

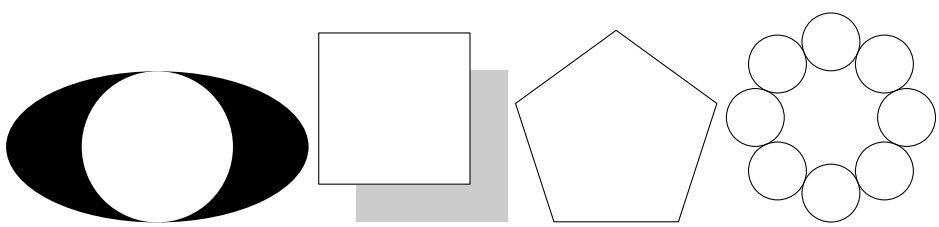
```
beginfig(1)
  path p[]; ut:=1mm; a:=25ut; d:=10ut;
  linejoin:=mitered;
  p1:=unitsquare scaled a;
  p2:=fullcircle scaled a;
  draw p1 withpen pencircle scaled (0.1a);
  draw subpath(0.5,4) of p1 withpen pencircle scaled (0.1a)
    shifted(a+d,0);
  draw (origin--point 0.5 of p1) shifted(a+d,0);
  label.lft(btex (a)\quad  etex,origin);
  draw p2 shifted(2.5a+2d,a/2);
  label.lft(btex (b) etex,origin) shifted (2a+2d,0);
  fill p2 shifted(2.5a+2d,a/2) withcolor 0.8;
  draw quartercircle scaled a withpen pencircle scaled (0.1a)
    shifted(2.5a+2d,a/2);
  draw p2 shifted(3.5a+3d,a/2);
  draw quartercircle scaled a withpen pencircle scaled (0.1a)
    shifted(3.5a+3d,a/2);
  unfill p2 shifted(3.5a+3d,a/2);
endfig;
```

Zusammenfassend kann man feststellen:

> **„Punkt – Linie – Fläche – Farbe sind die notwendigen und hinreichenden grafischen Grundelemente eines jeden Bildes."**

Diese Erkenntnis verdanken wir nicht zuletzt Wassily Kandinsky (1866–1945), der dies in seinem bis heute sehr lesenswerten Buch an vielen wunderbaren Beispielen dargestellt hat [23]. Bild 3.6 zeigt ein paar Grafiken aus seinem Buch. Kopieren Sie in vergrößertem Maßstab die Teilbilder (a)–(e) auf kariertes Papier, lesen Sie die Koordinatenwerte signifikanter Punkte ab und schreiben Sie entsprechende MP-Programme zum Zeichnen der Grafiken. Wie würden Sie im Fall des Teilbildes (f) vorgehen? (Lösung: Obere und untere Randlinie an den Enden zu einem geschlossenen Pfad verbinden und füllen.)

3.3.1 Weitere Beispiele



(a) (b) (c) (d)

Bild 3.8: (a) Auge, (b) Schattenbox, (c) n-Eck, (d) Kreiskette.

Die Zeichnungen in Bild 3.8 veranschaulichen den `fill/unfill` Mechanismus, die Überdeckungseigenschaften und die Kontrollstruktur `for: endfor` zur Wiederholung von Anweisungen in einer Schleife.

(a) „Auge“

```
beginfig(1)
  ut:=2mm;
  a:=10ut; b:=5ut;
  fill fullcircle xscaled 2a yscaled 2b;
  unfill fullcircle scaled 2b;
endfig;
```

(b) Schattenbox

```
beginfig(1)
  ut:=1mm;
  a:=20ut; d:=5ut;
  fill unitsquare scaled a shifted (d,-d) withcolor 0.8white;
  unfill unitsquare scaled a;
  draw unitsquare scaled a;
endfig;
```

(c) n-Eck

```
beginfig(1)
  ut:=0.7mm;
  a:=20ut; n:=5; alpha:=360/n;
  draw(
    for i=0 upto (n-1):
      a*dir(i*alpha)--
    endfor
    cycle) rotated (90-alpha);
endfig;
```

Die Kontrollstruktur `for: endfor` ist keine Anweisung, sie enthält keinerlei Strichpunkte und wird zwischen die Anweisungen eingeschoben.

(d) Kreiskette

```
beginfig(1)
  ut:=0.5mm;
  a:=20ut; n:=8;
  b:=a*sind(180/n);
  for i=0 upto (n-1):
    draw (fullcircle scaled 2b shifted (a,0)) rotated (i*360/n);
  endfor
endfig;
```

Mehrere Konstrukte wurden in dieser kurzen Einführung nicht erläutert. Dies ist durchaus beabsichtigt, da wir in späteren Kapiteln nochmals systematisch darauf zurückkommen werden. Die Beispiele sollten Ihnen lediglich einen ersten Eindruck vom Zeichnen

mit MP vermitteln.

3.4 Beschriftung

Bislang haben wir nur Bilder gezeichnet, aber nicht beschriftet. MP bietet jedoch als Erweiterung gegenüber MF die Möglichkeit, Bilder mit Texten und Formeln komfortabel zu beschriften. Insbesondere besteht eine enge Verbindung zu TEX und LATEX, sodass alle diesbezüglichen Formatierungsmöglichkeiten und Fonts hierfür genutzt werden können.

MP verwendet als Voreinstellung den Font Computer Modern, cmr10. Dies ist nicht ganz unproblematisch, da dieser Font im Vergleich zu PostScript kein Leerzeichen hat und die ASCII-Zeichen " < > \ _ { | } fehlen und mit anderen Glyphen besetzt sind. Der Text in Strings wird von MP als ASCII-Zeichenfolge interpretiert, gedruckt werden aber die den Codeplätzen entsprechenden Zeichen des Fonts cmr10. Abhilfe: Man wähle einen PostScript Font, z. B. Times Roman mit der Anweisung `defaultfont:="ptmr8r"`. Hier stimmen die Zeichen des ASCII-Codes und des PostScript-Fonts überein.

Zur Beschriftung einer Zeichnung dient der Label-Befehl mit der Syntax

⟨ label command ⟩ ⟶ `label`⟨ label suffix ⟩`(`⟨ label expr ⟩`,`⟨ pair expr ⟩`);`
⟨ label suffix ⟩ ⟶ ⟨ empty ⟩ | `top` | `bot` | `lft` | `rt` | `ulft` | `urt` | `llft` | `lrt`

Der ⟨ label suffix ⟩ gibt an, wie der Text relativ zur Position ⟨ pair expr ⟩ gesetzt werden soll. Ohne Suffix wird der Text bezüglich dieses Punktes zentriert gesetzt.

⟨ label expr ⟩ ⟶ ⟨ string expr ⟩ | ⟨ picture expr ⟩

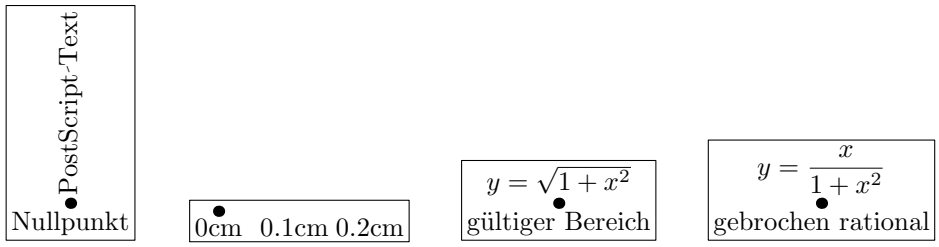
Der senkrechte Strich | bedeutet in der Syntax 'oder'. ⟨ label expr ⟩ ist also entweder ein String (Zeichenkette) oder ein Picture (Bildelement).

Beispiele.
1) String-Text
(a)

```
beginfig(1)
  draw origin withpen pencircle scaled 1.5mm;
  label.bot("Nullpunkt",origin);
  label.rt("PostScript Text",origin) rotated 90;
  draw bbox currentpicture;
endfig;
```

⟨ label expr ⟩ ist hier ein String, der aus einer in Doppelapostroph eingeschlossenen Folge von ASCII-Zeichen besteht, welche auf einer Zeile geschrieben werden muss und die keinen Doppelapostroph enthalten darf. Der Befehl `rotated` dreht den Label um 90°.

Man beachte, dass das „Leerzeichen“ in 'PostScript Text' falsch dargestellt wird, weil die ASCII-Zeichen in dem durch `defaultfont` eingestellten Schriftfont, hier `cmr10` (Vor-



(a) (b) (c) (d)

Bild 3.9: (a) String-Text, (b) String aus Text und MP-Daten, (c) TEX-Text, (d) LATEX-Text.

einstellung), gesetzt werden. Das Leerzeichen steht in der ASCII-Zeichentabelle auf Position 32, diese ist jedoch bei Computer Modern mit einem anderen Zeichen (˙) besetzt.

(b)

```
beginfig(1)
  draw origin withpen pencircle scaled 1.5mm;
  for i=0 upto 2:
    label.bot(decimal(i/10)&"cm",(i*1cm,0));
  endfor
  draw bbox currentpicture;
endfig;
```

Strings können mit & verkettet werden. `decimal` ist ein MP-Befehl, der einen Zahlenwert in einen String wandelt. Die Label-Anweisung ist in eine Schleife eingefügt und eine Funktion der Laufvariablen i.

2) TEX-Text

```
beginfig(1)
  draw origin withpen pencircle scaled 1.5mm;
  label.top(btex  $y=\sqrt{1+x^2}$ etex,origin);
  label.bot(btex g\"ultiger Bereich etex,origin);
  draw bbox currentpicture;
endfig;
```

Texte innerhalb von `btex etex` werden mit TEX gesetzt und in eine Variable vom Typ `picture` gewandelt. Man beachte, dass nach `btex` und vor `etex` mindestens ein Leerzeichen stehen und die gesamte Umgebung `btex etex` in eine einzige Zeile geschrieben werden muss. Es sind keine Zeilenumbrüche erlaubt. Wortzwischenräume, Umlaute, etc. (z. B. 'gültiger Bereich') werden jetzt korrekt formatiert. Es können alle Formatierungsmöglichkeiten von TEX (nicht LATEX!) verwendet werden.

3) LATEX-Text

```
beginfig(1)
  draw origin withpen pencircle scaled 1.5mm;
```

```
    label.top(btex  $\displaystyle y=\frac{x}{1+x^2}$ etex,origin);
    label.bot(btex gebrochen rational etex,origin);
    draw bbox currentpicture;
endfig;
```

Dieses Beispiel ist in dieser Form nicht lauffähig! Es funktioniert nur, wenn man zu Beginn und am Ende des MP-Programms eine LaTeX-Präambel bzw. LaTeX-Postambel einfügt, die MP anweist, wie die Angaben innerhalb von `btex etex` mit LaTeX formatiert werden sollen. Das MP-Quellfile `pic.mp` sieht nun folgendermaßen aus:

```
% Praeambel
verbatimtex
%&latex
\documentclass{article}
\begin{document}
etex;
beginfig(1)
<MP-Zeichenbefehle>
endfig;
% Postambel
verbatimtex
\end{document}
etex;
end
```

Sollte die Direktive `%&latex` in der Präambel vom System nicht ausgewertet werden, muss man diese Anweisung entfernen und alternativ die Umgebungsvariable `TEX` auf `latex` setzen gemäß

```
export TEX=latex
```

oder MP mit der Option `-tex=latex` aufrufen

```
mpost -tex=latex pic
```