Die TEXnische Komödie

DANTE
Deutschsprachige
Anwendervereinigung TEX e.V.

15. Jahrgang Heft 1/2003 Februar 2003

1/2003

Impressum

Beiträge sollten in Standard-LATEX-Quellcode unter Verwendung der Dokumentenklasse `dtk` erstellt und an untenstehende Anschrift geschickt werden (entweder per E-Mail oder auf Diskette). Sind spezielle Makros, LATEX-Pakete oder Schriften dafür nötig, so müssen auch diese mitgeliefert werden. Außerdem müssen sie auf Anfrage Interessierten zugänglich gemacht werden.

Diese Ausgabe wurde mit Hilfe folgender Programme erstellt: `pdfTeX 3.14159-1.00b-pretest-20020211 (Web2C 7.3.7x)`, `LaTeX2e (2001/06/01)`, `Acrobat Reader 4` und `xdvi(k) 22.40k` für die Bildschirmdarstellung. Als Standard-Schriften kamen die Type-1-Fonts CM-Super zum Einsatz.

Erscheinungsweise:	vierteljährlich
Erscheinungsort:	Heidelberg
Auflage:	2700
Herausgeber:	DANTE, Deutschsprachige Anwendervereinigung TEX e.V. Postfach 10 18 40 69008 Heidelberg E-Mail: `dante@dante.de` `dtk-redaktion@dante.de` (Redaktion)
Druck:	Konrad Triltsch Print und digitale Medien GmbH Johannes-Gutenberg-Str. 1–3, 97199 Ochsenfurt-Hohe Stadt
Redaktion:	Gerd Neugebauer (verantwortlicher Redakteur)

Luzia Dietsche, Rudolf Herrmann, Moriz Hoffmann-Axthelm, Gert Ingold, Rolf Niepraschk, Günter Partosch, Bernd Raichle, Volker RW Schaa, Herbert Voß

Redaktionsschluss für Heft 2/2003: 10. April 2003 ISSN 1434-5897

Editorial

Liebe Leserinnen und Leser,

wie so oft, war es auch dieses Mal so, dass der Redaktionsschluss für „Die TEXnische Komödie“ verstrichen war und als Ergebnis einer ersten Runde bei der Zusammenstellung des Heftes gerade einmal magere 42 Seiten zustande kamen. In diesem Fall haben wir die Veröffentlichung etwas hinausgezögert. Das hatte dann zur Folge, dass der Auslieferungstermin sich entsprechend verzögerte – etwas, das jedem Leser sicher sofort aufgefallen sein wird.

Und dann ging es auf einmal Schlag auf Schlag. Es wurden plötzlich einige Beiträge fast gleichzeitig eingeschickt. Damit war es dann auch diesmal wieder möglich, „Die TEXnische Komödie“ mit einigen interessanten Artikeln zu füllen. Hierzu zählt ein Beitrag zu Hüllen für CDs und ein einführender Artikel zur Typographie genauso wie eine tiefergehender Darstellung der Möglichkeit, optische Abbildungen zu erzeugen und eine Anwendung des Pakets `preview` zur Erzeugung von Grafikdateien.

Insbesondere bei dem Beitrag zu `pstricks` ist es wieder einmal schade, dass „Die TEXnische Komödie“ nicht in Farbe gedruckt wird. Einige der Grafiken gewinnen durch die Farbe eine zusätzliche Dimension. Aber im Ganzen gesehen sind die farblastigen Artikel eher selten, sodass die Entscheidung, bei einfarbigem Druck zu bleiben, schließlich doch gerechtfertigt ist. Auf der anderen Seite wäre ich aber auch hoch erfreut, wenn sich einmal genügend Beiträge zum Thema „Farbe“ finden würden und ich den Vorstand – und vor allem den Schatzmeister – davon überzeugen müßte, dass wir zumindest einmal eine mehrfarbige Ausgabe drucken sollten.

Ich habe schon länger den Wunsch, einen Ausblick auf kommende Beiträge zu geben. Durch die bereits vorliegenden Artikel bin ich in der glücklichen Lage, schon etwas darüber schreiben zu können, was in einer der nächsten Ausgaben gedruckt wird. Diese Chance möchte ich dieses Mal nutzen.

Ein bereits vorliegender Beitrag befasst sich mit dem Thema „Briefe“. Hierzu gibt es viele Klassen und das Thema wurde in der Mitgliederzeitschrift auch

schon behandelt. Diesmal geht es um die Briefklasse aus dem Koma-Script-Paket, die von Markus Kohm beschrieben wird. Von Werner Lemberg stammt ein Artikel zu deutschen Trennmustern. Und schließlich kam gerade noch ein Beitrag zu „Blackletter“-Schriften herein.

Alle diese Ankündigungen sollen aber niemanden davon abhalten, einen neuen Beitrag zu schreiben. Im Gegenteil zeigt die verspätete Versendung der Mitgliederzeitschrift, dass es einen großen Bedarf an neuen Beiträgen gibt. In diesem Sinne verbleibe ich mit meinem üblichen Aufruf und

mit TEXnischen Grüßen

Ihr Gerd Neugebauer

Hinter der Bühne

Vereinsinternes

Grußwort

Liebe Mitglieder,

dies ist die erste Ausgabe der Zeitschrift „Die TEXnische Komödie“ für 2003. Wir wünschen Ihnen alles Gute für das neue Jahr und hoffen, dass Sie bei der Verwirklichung aller Vorsätze erfolgreich sein werden (einer der Unterzeichner hat schon seit vier Tagen keine Schokolade mehr gegessen!).

Jahreswechsel sind häufig Anlässe für einen Kassensturz. Auch wir merken, dass das Geld bei vielen nicht mehr so locker sitzt. So erreichten uns kurz vor Jahresschluss einige Austritte, bei denen finanzielle Gründe angegeben wurden. Wir möchten daher darauf hinweisen, dass unsere Satzung die Möglichkeit vorsieht, bei wirklicher sozialer Notlage den Mitgliedsbeitrag zu reduzieren oder gar ganz zu erlassen. Bitte scheuen Sie sich nicht, sich in einem solchen Fall an uns zu wenden.

Erfreulich ist eine Spende von 500 Euro, die wir von Stephan Lehmke für unseren Projektfonds erhalten haben und für die wir uns herzlich bedanken. Insgesamt sind wir mit der Entwicklung des Projektfonds sehr zufrieden. Insbesondere die Zusammenarbeit mit der NTG und GUTenberg, die ähnliche Fonds unterhalten, funktioniert gut. So wurden in der Vergangenheit bereits einige Projekte gemeinsam gefördert.

Bei der Annahme der oben erwähnten Spende für den Projektfonds fiel auf, dass nach den Förderrichtlinien jede Erhöhung des Fondskapitals eines Beschlusses der Mitgliederversammlung bedarf – eine Regelung, deren eigentlicher Sinn war, eigenmächtige Erhöhungen durch den Vorstand auszuschließen. Daher ist ein formaler Beschluss der Mitgliederversammlung in Bremen erforderlich. Gleichzeitig werden wir eine Änderung der Förderrichtlinien anregen, die klarstellt, dass Spenden mit dem Verwendungszweck Projektfonds natürlich jederzeit willkommen sind.

Keine guten Nachrichten gibt es vom Verkauf des CTAN-Abzugs durch unseren Kooperationspartner, die Fachbuchhandlung Lehmanns. Dieser verläuft sehr enttäuschend. Offensichtlich wirken sich die jährlich steigenden Verkaufszahlen für die „TEX Live“ negativ auf die des CTAN-Abzugs aus. Lag zu Beginn der Zusammenarbeit mit Lehmanns der Verkauf des CTAN-Abzugs noch deutlich über dem der „TEX Live“, so wurden bis zum Jahresende 2002 nur wenige Exemplare in den Ladengeschäften verkauft. Wir müssen daher damit rechnen, dass das bisherige Modell – DANTE e.V. liefert die Master und erhält dafür von Lehmanns Freiexemplare für alle Mitglieder, die durch den Ladenverkauf finanziert werden – für Lehmanns wirtschaftlich nicht mehr tragbar ist. Nachdem wir bereits während der Mitgliederversammlung in Augsburg über die Zukunft des CTAN-Abzugs diskutiert haben, ergibt sich daher für die Mitgliederversammlung in Bremen erneuter Diskussionsbedarf. Herr Kaeder, unser Ansprechpartner bei Lehmanns Hamburg, will versuchen, auch nach Bremen zu kommen. Da sich mehrere andere Nutzergruppen beteiligt hatten und von Lehmanns zum Materialpreis mit dem CTAN-Abzug beliefert wurden, sind auch Gespräche mit diesen notwendig.

Im Anschluss an das Grußwort finden Sie die Einladung zur TEX-Tagung DANTE 2003 in Bremen. Wir planen für die Tagung einige Vorträge, in denen Projektverantwortliche über den Status ihrer Projekte informieren werden. So hoffen wir, dass die „Macher“ der Latin-Modern-Fonts (`lm`) die Version 1.0 vorstellen werden, wir einen Bericht über die Arbeiten an „TEX Live“ hören und über neue Entwicklungen am WYSIWYG-Frontend „Lyx“ informiert werden. Freundlicherweise hat auch unser Ehrenmitglied Prof. Hermann Zapf sein Erscheinen zugesichert. Vielleicht können wir nochmals bei einem Vortrag wie auf der Tagung in Clausthal-Zellerfeld seine Kalligraphien an der Tafel bewundern.

Wir laden Sie herzlich ein und würden uns freuen, wenn Sie diese Tagung besuchen. Wir denken, dass für jeden, der an der Weiterentwicklung und Benutzung von TEX interessiert ist, interessante Vorträge dabei sein werden. Daneben gibt es wieder die Gelegenheit zum persönlichen Kontakt mit Entwicklern und Gleichgesinnten. Wir sind uns sicher, dass im breiten Angebot auch für Sie etwas dabei sein wird und treffen uns – nach langer Zeit – mal wieder im Norden des deutschsprachigen Raumes.

Mit freundlichem Gruß,

Volker RW Schaa	Klaus Höppner
Vorsitzender	Stellvertretender Vorsitzender

Einladung zur TEX-Tagung DANTE 2003 und 28. Mitgliederversammlung von DANTE e.V.

Volker RW Schaa, Roland Weibezahn

Hiermit laden wir Sie herzlich zur TEX-Tagung DANTE 2003 und 28. Mitgliederversammlung von DANTE e.V. vom 2.–4. April 2003 an der

Universität Bremen
Gebäude NW 1
Otto-Hahn-Allee
28359 Bremen

ein.

Veranstalter sind gemeinsam das Institut für Wissenschaftliche Datenverarbeitung (IWD) der Universität Bremen und DANTE e.V.

Das Gebäude NW 1 befindet sich im Süden des Universitäts-Campus. Mit der Straßenbahn oder dem Bus erreichen Sie den Tagungsort über die Haltestelle „Universität/NW1“, mit dem Auto über die Autobahn A 27 (Ausfahrt „Horn/Lehe, Universität“). Parkplätze stehen gegen geringe Gebühr in der Nähe zur Verfügung.

Die Tagesordnung der Mitgliederversammlung am Donnerstag, den 3. April 2003, um 9.30 Uhr lautet:

1. Begrüßung; Vorstellung des Vorstands; Tagesordnung
2. Bericht des Vorstands
3. Finanzbericht
4. Bericht der Kassenprüfer
5. Entlastung des Vorstands
6. Wahl von Kassenprüfern
7. Projektfonds
   (a) Erhöhung der Projektmittel durch eine zweckgebundene Spende

(b) Änderung der Förderrichtlinien bezüglich zweckgebundener Spenden

8. Zukunft CTAN-Abzug

9. Verschiedenes

Wie üblich sind auch Nichtmitglieder als Gäste der Mitgliederversammlung willkommen.

Ihre Stimmunterlagen erhalten Sie direkt vor Ort vor Beginn der Mitgliederversammlung. Bitte melden Sie sich zur Mitgliederversammlung an, Sie erleichtern damit die Vorbereitung der Unterlagen. Ein Anmeldeformular und weitere Informationen finden Sie unter `http://www.dante.de/dante/events/dante2003`. Mit Fragen oder Anregungen wenden Sie sich bitte per E-Mail an

`dante2003@dante.de` (bevorzugt)

oder an

Universität Bremen
Institut für Wissenschaftliche Datenverarbeitung
Dr. Roland Weibezahn
Postfach 33 04 40
28334 Bremen

Wir würden uns freuen, Sie zahlreich auf der Mitgliederversammlung begrüßen zu dürfen.

Mit freundlichem Gruß,

Volker RW Schaa (DANTE e.V.)
Roland Weibezahn (Universität Bremen, IWD)

Bretter, die die Welt bedeuten

Hüllen (nicht nur) für Musik-CDs

Christian Faulhammer

Mit Hilfe der Dokumentenklasse `cd-cover` kann man sehr einfach Einleger für die verschiedensten CD-Hüllen-Typen anlegen. Die Möglichkeiten sind dabei vielfältig, so dass für jede Bauart etwas dabei ist, egal ob Single, Standard-Hülle oder gar Papiertasche.

Einleitung

Angeregt durch den Artikel „Hüllen für Musikkassetten“ in „Die TEXnische Komödie“ 03/2002 von Florian Benischke, habe ich mich auf die Suche nach einem entsprechendem Paket für CDs gemacht. Nur noch eine kleine Minderheit wird heute Musikkassetten einsetzen, da die CD(-ROM) schon seit Jahren ihren festen Platz hat und häufiger genutzt wird, sei es für Backups, eigene Musikkompilationen oder ähnliche Zwecke.

Zur Erstellung von CD-Hüllen kommt kein Paket, sondern eine eigene Klasse namens `cd-cover` zum Einsatz, zu finden auf CTAN: `macros/latex/contrib/other/cd-cover`. Die Klasse stellt für die Gestaltung fünf Umgebungen bereit, mit denen sich vier Arten von Hüllen (plus einer Variante) erstellen lassen:

- `bookletsheets` erstellt ein Booklet (also die Vorderseite) für Standard-CD-Hüllen und ist aufklappbar.
- `backsheet` generiert die Rückseite für denselben Hüllentyp. Die Stern-Form der Umgebung verändert nur die Laufrichtung des Titels auf dem „Rücken“, also auf der schmalen Seite, die man lesen kann, wenn die CD im Regal/Ständer steht.

- `sleevesheet` bietet eine Vorlage für CD-Taschen, in die häufig die CD-Beilagen von Zeitschriften verpackt sind. Diese haben natürlich kein Plastikfenster und müssen auch noch zusammengeklebt werden (Klebekanten werden angelegt).
- `singlesheet` tut genau das, was der Name sagt: Cover für Single-Hüllen, die man auch häufiger bei Rohlingen im Handel findet.

Wichtig: Ebenso wie das Paket `mceinleger.sty` braucht `cd-cover` zusätzlich das Paket `rotating.sty`, das automatisch geladen wird. Die Anzeige im DVI-Betrachter kann daher vom Ausdruck abweichen, das gilt vor allem für die „Rücken"-Texte.

Benutzung des Paketes

Umgebungen

Der Nutzer muss natürlich auswählen, was er erstellen will und sucht danach die benötigte Umgebung aus. Wichtig ist die Angabe des Papierformates in den Optionen zu `\documentclass`, da sonst die Ausgabe verrutscht und Teile über den Rand hinausragen. Die Formate folgen dem Modus der Standard-Dokumentenklasse `article`, von der `cd-cover` abgeleitet wurde.

Für alle Leser von „Die TEXnische Komödie" sollte die Angabe von `a4paper` richtig sein. Von der Benutzung zusätzlicher Pakete zum Setzen des Papierformates wie `a4.sty` oder `vmargin.sty` sollte abgesehen werden, weil dies zu unvorhersehbaren Problemen führen kann.

Die Syntax ist sehr einfach, dennoch stelle ich alle Umgebungen einzeln vor.

```
\begin{bookletsheets}
  <irgendetwas muss hier stehen>
\end{bookletsheets}
```

`cd-cover` erstellt einen Rahmen und schreibt beliebigen Text einfach hinein, man kann natürlich alle Elemente und Befehle benutzen, die LaTeX so kennt, beispielsweise Tabellen, Aufzählungen, aber auch Grafiken. Es stehen dabei zwei Seiten zur Verfügung, die bei Fließtext voll geschrieben werden (die Seiten sind natürlich recht klein), mit einem `\newpage` kann man auf die zweite Seite wechseln und dort eine Liedliste unterbringen.

```
\begin{backsheet}{<Titel>}
  <irgendetwas muss hier stehen>
\end{backsheet}
```

Hierbei wird der Einleger für die CD-Hüllen-Rückseite auf *einem* Blatt erstellt, deswegen auch der Singular im Umgebungsnamen. Dieser Umstand wird nur erwähnt, weil es auf Nutzerseite zu Komplikationen führen kann, wenn LaTeX die Umgebung nicht kennt, obwohl man meint, alles richtig gemacht zu haben. Leider ist der Autor nicht ganz konsequent in seiner Benennung, oder mir ist das Schema schlicht nicht zugänglich.

Die Rückseite enthält in den meisten Fällen die Inhaltsangabe (in Form einer Liedliste oder Ähnlichem), Text wird deshalb auch hier am häufigsten zum Einsatz kommen. Das heißt nicht, dass irgendetwas verboten wäre, man hat wieder volle Handlungsfreiheit.

Der Parameter der Umgebung nimmt den Titel auf, der auf den beiden schon erwähnten „CD-Hüllen-Rücken“ erscheint und die sich rechts und links gegenüber stehen: Der linke Rücken beginnt unten, der rechte oben und man neigt den Kopf nach links, um den linken lesen zu können. Die Sternform vertauscht die Laufrichtung der beiden Rücken.

```
\begin{sleevesheet}
  <irgendetwas muss hier stehen>
\end{sleevesheet}
```

Diese Umgebung funktioniert wie `bookletsheets`, nur dass zuerst die rechte Seite (Vorderseite der Hülle) beschrieben wird.

```
\begin{singlesheet}{<Laschentext>}{<Titel>}
  <irgendetwas muss hier stehen>
\end{singlesheet}
```

Hiermit wird ein Cover für eine Single-Hülle erstellt, die Titelangabe sollte klar sein, der Text muss aber noch schmaler sein als bei einer normalen CD-Hülle. Der Laschentext ist quasi die Rückseite und enthält bei Musik-Singles meistens die Liedliste.

Längen

Es gibt einige Längen, die den Abstand des Textes bzw. des Bildes vom gedruckten Rand festlegen. Tabelle 1 zeigt die komplette Liste; das Format

Tabelle 1: Längen der Klasse `cd-cover`

`\CDbookletTopMargin`	`\CDbookletMargin`
`\CDbackTopMargin`	`\CDbackMargin`
`\CDsleeveTopMargin`	`\CDsleeveMargin`
`\CDsingleTopMargin`	`\CDsingleMargin`

ist immer dasselbe: Alle, die auf `TopMargin` enden, geben einen vertikalen Abstand an, `Margin` bezieht sich auf einen horizontalen Abstand. Der Text zwischen `\CD` und der Endung bezeichnet den Hüllen-Typus, auf den sich die Länge bezieht, und sollte selbsterklärend sein. Standardmässig ist überall 5 mm eingestellt.

Mit einer Abstandsangabe von 0 (egal in welcher Einheit) können Bilder ein gesamtes Blatt ohne hässliche weiße Ränder ausfüllen; die Höhenangabe für das Bild sollte aber immer kleiner sein als die zwölf Zentimeter, damit LaTeX nicht zufälligerweise eine neue Seite erzeugt und das Bild dorthin mitnimmt.

Probleme

Es gibt ein sehr eigenartiges Verhalten, wenn man mehrere Einleger in einem Dokument anlegt. Das Standardverhalten am Ende einer Seite ist entweder auf die nächste Seite zu springen (soweit noch möglich) oder auf der ersten Seite eines neuen Einlegers desselben Typs zu beginnen. Leider geht ein zu langer Text manchmal gnadenlos über die Randmarkierungen hinaus. Dies tritt vor allem beim ersten Einleger auf, alle weiteren erscheinen meistens korrekt.

Zudem ist die Ausgabe des Laschentextes in der Umgebung `singlesheet` nicht ganz optimal positioniert, mit Hilfe der Anweisungen `\vspace` und `\hspace` lässt sie sich aber korrigieren (siehe Beispiele). Diese Probleme sind bereits an den Autor gemeldet, er hat versprochen, diese zu beheben, ebenso einige Fehler in der Dokumentation, die aber für die Funktion nicht relevant sind.

Beispiele

```
\documentclass[a4paper]{cd-cover}
```

```
\begin{document}
\begin{singlesheet}{%
  \vspace*{-1.5cm}\begin{enumerate}
   \item erstes Lied \item zweites Lied
   \end{enumerate}}
  {\vspace*{-1.5ex}{\tiny meine erste Singlehuelle}}
  Dieser Text dient nur dazu, dass die Seite ueberhaupt
  ausgegeben wird. Steht hier nichts, wird auch
  keine Huelle erzeugt.
\end{singlesheet}
\end{document}
```

```
\documentclass[a4paper]{cd-cover}
\begin{document}
\begin{booklets heets}
  \vspace*{0.4\textheight}
  {\huge Wieder mal Text, damit wir was zeigen koennen...}
  \newpage
  Und jetzt sind wir auf der zweiten Seite des Booklets.
\end{booklets heets}
\end{document}
```

Zusammenfassung

`cd-cover` erlaubt die schnelle Erstellung von CD-Hüllen für jeden Verwendungszweck. Ob man eine Backup-CD beschriften möchte oder für eine eigene Musikkompilation ein ansprechendes Cover braucht, diese Klasse bietet das alles, auch wenn sie sich noch im Beta-Stadium befindet. Bis auf ein paar kleine Fehler (falsche Beschriftung der Klebemarken, teilweise schlechte Textpositionierung) entspricht die Ausgabe (fast) den Erwartungen.

Nach dem Ausdruck muss man nur noch entlang der fetten Linien ausschneiden, die gestrichelten Faltmarken benutzen, und man hält sein fertiges Booklet in den Händen. Die CD-Taschen haben praktischerweise Klebelaschen und sollten auch ausreichend sein für Konzept-CDs, die man nur kurzzeitig braucht, aber sicher und platzsparend transportieren will.

Wer automatisch für ein Album mit Hilfe von CDDB ein Cover erstellen will, sollte einmal einen Blick auf `http://home.wanadoo.nl/~jano/disc-cover.html` werfen. Es ist ein in Perl geschriebenes Projekt, das nach

Eingabe von wenigen Infos eine sehr ansprechende Ausgabe erzeugt. Dabei hat man die Wahl zwischen PS, PDF, LaTeX und einigen anderen Formaten, alles über eine Web-Oberfläche (natürlich auch lokal ausführbar, wenn ein Interpreter vorhanden ist).

Eine weitere Übersicht, die Programme auflistet, mit deren Hilfe CD-Cover erstellt werden können, findet sich auf `http://mp3cover.sourceforge.net/news-article.html`. Diese Liste ist als Übersicht ganz nützlich, wenn man mit dem oben erwähnten Perl-Programm oder `cd-cover` nicht zufrieden ist, was ich für sehr unwahrscheinlich halte.

Digitaler Textsatz, digitale Typographie. Ein Überblick.

Tim Doll

Ziel dieses Artikels ist es, einen Überblick über typographische Entwicklungen im Arbeitsprozess der digitalen Textproduktion zu verschaffen. Nach einer knappen historischen Übersicht werde ich kurz die Entwicklung von Satz- und Druckverfahren im Zeitungswesen darstellen, dies insbesondere in Hinsicht auf die Standardausgabeformate PostScript und Portable Document Format (PDF) von Adobe Systems. Weiterhin werden mit dem Satzsystem TeX, dem dazu gehörigen Zeichensatzprogramm METAFONT und dem sogenannten *Desktop Publishing* (DTP) wichtige Entwicklungen im typographischen Bereich dargestellt. Die Diskussion erfolgt vor allem mit Blick auf die Frage, inwiefern sich DTP-Autoren und Mainstream-DTP-Software als „gute Typographen“ eignen. Als Schlussfolgerung ergibt sich, dass weder professionelle Methoden noch DTP zu einem befriedigenden Ergebnis führen, wenn vorausgesetzt wird, dass zu Gunsten des effizienten digitalen Arbeitsprozesses das menschliche Beurteilungsvermögen typographischer Kriterien vernachlässigt werden kann.

Historischer Überblick

Die Frage nach der bedeutendsten historischen Entwicklung in der Geschichte der Schrift und des Schreibens wird gern mit dem Verweis auf Johannes

Gutenberg (†1468) und seiner Buchdruckmaschine beantwortet, mit der erstmals mittels beweglicher Lettern Seiten von Dokumenten und Büchern auf verlässliche Weise wiederholt werden konnten. War das Verfassen von Manuskripten zuvor ein auf die Manipulation eines Schreibwerkzeuges beschränkter Prozess, so wurde nun die Kunst des Schreibers um die handwerklichen Fähigkeiten des Setzers und des Druckers ergänzt. Der kostbarste Rohstoff für den Schreiber beispielsweise eines universitären oder klösterlichen Skriptoriums war zunächst der Beschreibstoff, entweder Pergament aus Tierhäuten oder Papier, sodass eine Manuskriptseite möglichst ökonomisch ausgenutzt werden musste, indem man sie nicht nur eng beschrieb, sondern sich beim Schreiben auch auf Konventionen von Abkürzungen und Kontraktionen verließ. War der Auftraggeber sehr wohlhabend, so findet man gelegentlich auch Manuskripte, die etwas freizügiger und weniger eng beschrieben sind.

Diese Tatsachen und auch die sich daraus ergebenden zeitgenössischen Lesegewohnheiten führten dazu, dass es vorerst Aufgabe des Druckhandwerks war, die handschriftlich gezeichneten Buchstaben der Manuskriptschreiber möglichst getreu nachzuahmen und auf mechanische Art und Weise ein Produkt anzufertigen, das eine Manuskriptvorlage zwar nicht unbedingt in ihrer Originalität, aber in Perfektion und vermehrt auch in seiner Ästhetik bei weitem übertraf.

Nachdem an Gutenbergs Errungenschaft knappe 500 Jahre keine *prinzipiellen* Veränderungen vorgenommen wurden, war es die Idee, dass nicht nur das Drucken, sondern auch das Setzen von Zeichen, Zeilen und Seiten automatisiert werden sollte, die die nächste Revolution in der Geschichte des Schreibens ankündigte. Bereits im Jahre 1784 plante J. H. Müller, seine von ihm entworfene Rechenmaschine durch einen Apparat zu ergänzen, der die ausgeführten Rechenoperationen automatisch auf ein Papier drucken und den Druckprozess mit dem voll beschriebenen Blatt enden lassen sollte [3].

Knappe 50 Jahre später war es der Engländer Charles Babbage (1791–1871), der an den ebenso ehrgeizigen Plänen scheiterte, das Konzept seiner *Analytical Engine* zu verwirklichen, die als Vorläufer des modernen digitalen Computers angesehen wird. Mit dem Ziel, die Ergebnisse seiner früher konstruierten *Difference Engine* mechanisch ausgeben zu lassen, experimentierte Babbage mit beweglichen Lettern, Stereotypen und gestanzten Kupferkarten. Sein Entwurf zur *Analytical Engine* sah eine Maschine vor, die bereits gespeicherte Operanden als Grundlagen zur Addition, Subtraktion, Multiplikation und Division abrufen sollte. Zur Ausführung komplexerer Rechenoperationen soll-

te Babbages Maschine Eingaben von gestanzten Kupferkarten einlesen und die Ergebnisse auch in eben dieser (und später in gedruckter) Form wieder ausgeben können. Unglücklicherweise war jedoch sein Vorhaben nicht nur auf Grund finanzieller Schwierigkeiten zum Scheitern verurteilt – Babbage konnte die für die Verwirklichung nötigen feinmechanischen Teile erst gar nicht mit der entsprechenden Präzision herstellen, sodass er das Ergebnis seiner Arbeit nicht mehr erleben konnte. Nachdem die Details von Babbages Plänen in Vergessenheit geraten waren, gelang es britischen Wissenschaftlern erst im Jahre 1991 anhand von Babbages Spezifikationen, die *Difference Engine No. 2* zu konstruieren, die auf bis zu 31 Ziffern exakt arbeitet.

Erfolg versprechender waren hingegen die Pläne des Schweden Georg Scheutz, der zusammen mit seinem Sohn Edvard mehrere Rechenmaschinen konstruierte, die nicht nur komplexere Rechenoperationen erfolgreich ausführen konnten. Bereits die erste Scheutz-Maschine (1849) produzierte Gussformen, wie sie auch im Stereotyp-Verfahren verwendet wurden, und war somit in der Lage, eine Tabelle auszudrucken, die beispielsweise die Werte von $x^3 - 3x^2 + 90037x$ für $1 \leq x \leq 25$ enthielt. Spätere Versionen dieser Rechenmaschine konnten schließlich über mehrere Seiten hinausgehende Tabellen mit einer Rate von ca. 120 Ziffern pro Stunde ausdrucken, und dies bei einer durchweg akzeptablen Qualität [3].

Fast 100 Jahre später erst wurde die Kapazität der damals modernen Computer ausgenutzt, um Operationen auszuführen, die über „einfache“ arithmetische Berechnungen hinausgehen. Sieht man den 1945 in Harvard entwickelten ENIAC als den Grundstein der modernen Computertechnologie an, da er als erster Computer das Binärsystem ausnutzte, so war der LINASEC von Compugraphic der erste Computer, der von einer Endlos-Wortkette einen Zeilenumbruch berechnen konnte. LINASEC gab seine Ergebnisse an eine Linotype-Maschine weiter, die dann die entsprechenden Zeilen in Metall goss und setzte, eine Entwicklung, die den Typographen John Dreyfus zu der Aussage veranlasst, dass die Typographie erst zu diesem Zeitpunkt ihren Einzug ins Computerzeitalter gehalten habe [8].

Es ist durchaus bemerkenswert, dass selbst zum Zeitpunkt der Markteinführung von LINASEC in den 60er Jahren, und auch bis sehr viel später, weiterhin „traditionelle“ Satz- und Druckverfahren zum Einsatz kamen. Grundlegende Veränderungen von Gutenbergs Druckpresse brachten weder die Monotype noch die Linotype, die Gutenbergs Prinzip des Hochdrucks lediglich mit effizienteren Arbeitsmethoden weiter verfolgten. Beiden Metho-

den gemeinsam ist die Tatsache, dass mit Hilfe von Matrizen Gussformen angefertigt werden, die dem Setzen von individuellen Seiten dienen. Wenngleich dies über verschiedene Wege und mit unterschiedlicher Arbeitsleistung geschieht, so bleibt festzuhalten, dass doch erst mit dem vom Offsetdruck verwendeten Flachdruckverfahren ein gänzlich neues Prinzip Einzug in die Drucktechnologie gehalten hat.

Man ist hierbei versucht, auch die Einführung der Photokomposition als vollständige Neuerung im Bereich des Satzes anzusehen, doch es bleibt hier festzuhalten, dass das Prinzip der Photomaschinen der ersten Generation im Großen und Ganzen als eine Weiterführung der traditionellen Methoden mit anderen Mitteln anzusehen ist. Im Aufbau war beispielsweise der MONOPHOTO von Monotype ähnlich konzipiert wie seine mechanischen Vorgänger, nur war das Gussverfahren durch einen photographischen Apparat ersetzt und die Matrizen bestanden nicht mehr aus Messing, sondern aus photographischem Material. Obgleich diese Verfahren bereits computergesteuert waren, ist zu bemerken, dass die Methoden im Grunde wenig mehr boten, als lediglich die traditionellen, beispielsweise von Monotype und Linotype verwendeten Tastaturen zu ersetzen [10].

Dies führt aber endlich zu der Frage, wodurch digitale Typographie und digitaler Textsatz definiert sind. Hierbei schließe ich mich Patterson an, der in Anlehnung an Rubenstein lediglich solche Methoden der Typographie und des Textsatzes als digital definiert, die eine digitale Bearbeitung von sowohl Form als auch Inhalt beinhalten. Somit bestehen die Hauptunterschiede zwischen traditionellem und digitalem Textsatz in den Eingabemöglichkeiten („blind keyboarding giving way to display terminals“ [10]) und der digitalen Ausgabe, die dazu in der Lage ist, die verschiedensten Seitenelemente in einem Arbeitsschritt zu positionieren, gleichzeitig über verschiedene Zeichensätze verfügt usw. Das bedeutet, dass beispielsweise frühe computergesteuerte Photomethoden sowie solche, die ein manuell-mechanisches Eingreifen in den Arbeitsprozess beinhalten, wie beispielsweise das nachträgliche Einfügen von Bildelementen, nicht als digital definiert werden. Dieser Definition folgend ergeben sich somit zwei wichtige Kennzeichen des digitalen Arbeitsprozesses. Erstens ist das Verfahren durchweg digital gesteuert, vom Seitenlayout im Eingabegerät bis hin zu den Anweisungen an das Ausgabegerät, an welcher Stelle der Seite ein Bildpunkt zu setzen ist, d. h. allgemein formuliert, wie Text- und Graphikelemente auf einer Seite arrangiert werden. Hieraus ergibt sich bereits das zweite Merkmal des als digital definierten Prozesses: Das zu beschreibende Grundprodukt ist die Seite und nicht mehr die Fahne, die zwar

bereits den zu setzenden Text enthält, aber keine Informationen darüber, wie und wo der Text anzuordnen ist, einschließlich Spalten- und Seitenumbruch. Letzteres war auch beim Photoverfahren eine Aufgabe geblieben, die letztendlich nachträglich durchgeführt werden musste.

Als eines der wichtigsten Medien des digitalen Arbeitsprozesses in der Dokumentgestaltung gilt bis heute PostScript von Adobe, das sich als Programmiersprache und Ausgabeformat als Standard durchgesetzt hat. PostScript entspricht der obigen Definition nicht zuletzt auf Grund der Tatsache, dass seine grundlegende strukturelle Beschreibungseinheit die Seite eines Dokuments ist und somit die Fahne als Grundeinheit ersetzt hat. Im Folgenden werde ich demnach zunächst den Übergang vom traditionellen hin zum digitalen Arbeitsverfahren im professionellen Bereich am Beispiel der Zeitungsindustrie, insbesondere in Hinblick auf PostScript und PDF, und weiterhin im „nicht-professionellen“ Bereich des Desktop-Publishing, darstellen. Dies wird schließlich auch zu einer tiefer gehenden Auseinandersetzung mit typographischen Merkmalen eines Dokuments führen, wenngleich auch dies in den Hintergrund der bisherigen Darstellung gerückt war. Dies hängt nicht zuletzt damit zusammen, dass mit den bisherigen Verfahren Produktionsmethoden vorgestellt wurden, die dem Typographen im eigentlichen Arbeitsverfahren nicht zugänglich sind, da sie im Grunde Bestandteil des Druckprozesses sind.

Digitales Drucken und Setzen

Traditionelle und digitale Prozesse im Zeitungswesen

Bis heute hat sich der Arbeitsprozess des Setzens und Druckens fast vollständig zu einem computerisierten, aber wohlgemerkt nicht „verunmenschlichten“ Vorgang entwickelt. Es ist bezeichnend, dass diese rationalisierte Arbeitsweise besonders durch die Bedürfnisse der expandierenden Zeitungsindustrie bedingt war und noch immer ist, während der Buchdruck über die Jahrhunderte ein sehr traditionelles Geschäft geblieben war. Gerade die Zeitungen waren es, die von den neuen und mechanisierten Entwicklungen profitierten, wohingegen der Buchdruck stets eine gewisse Zeit brauchte, um die neuen Methoden für eine höhere Qualität und ein verbessertes Buchdesign zu übernehmen.

Noch in den 80er Jahren des vergangenen Jahrhunderts war es für viele Zeitungshäuser üblich, Artikel und Anzeigen nach der traditionellen Methode zu arrangieren, indem Ausdrucke angefertigt wurden, um diese erst dann dem

Satzvorgang zu übergeben. Anzeigen werden hierbei von den einzelnen Werbeagenturen individuell und mit Hilfe von Software-Anwendungen entworfen, ausgedruckt oder zu Film gebracht und schließlich per Post oder Express an den Zeitungsverlag geschickt. Bei diesem Verfahren ergibt sich die Schwierigkeit, dass der Verlag nur sehr eingeschränkte Möglichkeiten hat, die Anzeige weiter zu bearbeiten. Die Bearbeitung erfolgt hierbei lediglich in Hinsicht auf ein Überprüfen der Größe und Farbeigenschaften der Anzeige. Letztlich aber muss sich das Verlagshaus darauf beschränken, den Ausdruck direkt in die Produktion weiterzuleiten, wo er für eine eventuelle spätere Weiterverwendung gescannt und schließlich in die entsprechende Seite eingefügt wird. Die offensichtliche Problematik liegt nun nicht nur in der Tatsache begründet, dass der Verlag grundsätzlich keine Möglichkeiten hat, in das Design der Anzeige einzugreifen um somit Einfluss auf ihre letztendliche Qualität nehmen zu können, sondern auch darin, dass die Anzeige den oftmals unzuverlässigen Postweg nehmen muss, wobei Lieferkosten und Termine eine nicht zu vernachlässigende Rolle spielen.

Ähnlich verhält es sich mit den von Redakteuren der Zeitung verfassten Artikeln, wobei der Satz des Artikels jedoch häufig innerhalb eines Layout-Programms vorgenommen wird und Anzeigen als letztes Element der anzufertigenden Seite nach Cut-and-Paste-Methoden eingefügt werden. Insbesondere sind es Farbphotos und kolorierte Graphiken, die nur in einem recht arbeitsintensiven Verfahren eingebunden werden können und somit weitere Probleme bei der Erstellung des Dokuments aufwerfen. Dies geht so weit, dass qualitativ hochwertige Farbillustrationen oftmals lediglich in Sonderberichten, nicht aber innerhalb des üblichen Zeitungsformats erscheinen.

Der digitale Arbeitsprozess bietet den kommerziellen Verlagen nun eine erhebliche Vereinfachung bei der Erstellung ihrer Dokumente. Die computerisierte Bearbeitung von Druckdokumenten ist u. a. gekennzeichnet durch die ausschließliche Verwendung von digitalen Daten als Druckvorlage und die direkte Steuerung des Druckvorgangs aus dem Datenbestand. Für das Seitenlayout stehen Software-Applikationen zur Verfügung, die auch ein nachträgliches Einfügen von Elementen, insbesondere von Anzeigen, die auf elektronischem Wege an den Verlag geschickt werden, auf bequeme Art und Weise ermöglichen. Der Inhalt des Dokuments wird dabei einer Datenbank entnommen und automatisch auf der dafür vorgesehene Seite platziert, und es werden Überschriften und Unterschriften für Illustrationen und Photos hinzugefügt. Es ist offensichtlich, dass hierbei verschiedene traditionelle Arbeitsstufen ent-

fallen, wie beispielsweise die Verwendung von Druckplatten oder Filmen, da die Daten direkt der entsprechenden Datei entnommen werden können.

Als Folge ergibt sich nun, dass bei einer Steigerung der Qualität des Druckdokuments bei einer ausreichend hohen Auflösung sowohl die Produktionskosten als auch die Fehlerquote innerhalb des Arbeitsprozesses sinken, dies nicht zuletzt durch die Möglichkeit von Redakteuren und Werbeagenturen, ihre Artikel bzw. Anzeigen per Fax oder E-Mail weiterzuleiten. Eine Qualitätssteigerung ergibt sich nicht nur durch eine größere Flexibilität im Arbeitsvorgang und hinsichtlich der Ausgabegeräte, sondern auch durch eine größere Genauigkeit im Platzieren von Seitenelementen, die der Datenbank entnommen werden. Darüber hinaus entfallen bedeutende Schwierigkeiten, die mit der Einbindung von Farbelementen auf der Seite verbunden waren, da nun die Berechnung der Farbpunkte während der Seitenbearbeitung geschieht und der Prozess der mechanischen Farbgebung der Vergangenheit angehört.

Nachdem das Seitenlayout fertig gestellt ist, wird eine PostScript-Datei erstellt, die an ein Ausgabegerät, sei es ein Drucker, Plotter oder Monitor, gesendet werden kann. Die PostScript-Datei wird schließlich an einen *Raster Image Processor* (RIP) gesendet, der die Datei weiterverarbeitet und zum Druck weitergibt, entweder an die hauseigene oder per Fax an eine ausgelagerte Druckerei. Im nächsten Abschnitt werde ich etwas näher zu beschreiben versuchen, wie das Erstellen und Verarbeiten einer PostScript-Datei vonstatten geht.

PostScript

PostScript selbst vereint in diesem Prozess verschiedene wichtige Eigenschaften. Im Grunde ist PostScript eine eigene Programmiersprache, die als Vermittler zwischen einer Software-Anwendung und einem (Druck-)Ausgabegerät fungiert. Als Programmiersprache dient PostScript somit einerseits dazu, eine abstrakte Beschreibung dessen zu erzeugen, was beispielsweise von einem Drucker ausgegeben werden soll, und andererseits dazu, diese Beschreibung in einer für den Drucker verständlichen Weise als Druckanweisung weiterzuleiten. Es ist hierbei interessant festzustellen, dass eine Datei in der PostScript-Sprache nur in den seltensten Fällen von einem menschlichen Benutzer, sondern üblicherweise von einer anderen Software-Anwendung, beispielsweise von einem Layout-Programm wie QuarkXPress, erzeugt wird. Damit bleibt PostScript selbst im Grunde für den Benutzer unsichtbar im Hintergrund. Seine Fähigkeiten zur Seitenbeschreibung leitet PostScript aus

einer Menge von graphischen *low-level*-Befehlen ab, die einfache Linien und Kurven zeichnen. Diese sogenannten *primitives* werden von PostScript derart miteinander kombiniert, dass auf eine äußerst leistungsfähige und flexible Art und Weise komplexe Formen erzeugt werden können.

Im Gegensatz zu Anwendungen, die ihre Ausgabe in Form von Bitmaps an den Drucker weitergaben, war es mit PostScript möglich geworden, den rechenintensiven Vorgang der Umwandlung von Daten in Pixel (*scan conversion*) zu umgehen. So waren in den frühen Phasen von Donald Knuths TEX und METAFONT für eine hochauflösende Ausgabe ein leistungsstarker Rechner und eigene Gerätetreiber notwendig, da die entsprechende Leistung von damaligen Mikrocomputern nicht erbracht werden konnte. Da PostScript aber nicht jeden einzelnen Bildpunkt eines Dokuments berechnen und die speicherintensive Bitmap an das Ausgabegerät weitergeben, sondern lediglich eine abstrakte Beschreibung der auszugebenden Seiten erstellen musste, war ein erheblicher Schritt hin zu einer höheren Ökonomie getan. Dies wird dadurch erreicht, dass die PostScript-Datei (oder besser gesagt, das PostScript-Programm) das Ausgabegerät in Form einer auf ASCII (*American Standard Code for Information Interchange*) basierenden Textdatei anspricht, die bekanntlich erheblich schneller und effizienter über eine serielle Schnittstelle zu übertragen ist als beispielsweise eine Bitmap im A4-Format, die je nach Auflösung eine Größe im Megabyte-Bereich aufweisen kann. *Scan conversion* wird nun also nicht mehr vom Rechner selbst, sondern vom *Raster Image Processor* ausgeführt, der hierbei im Grunde als Interpreter fungiert und die in PostScript geschriebene Datei in eine Bitmap (Raster) übersetzt und an ein Ausgabegerät sendet, was gleichzeitig den Weg zu einer weiteren Neuerung ermöglichte.

Da die fertiggestellte PostScript-Datei nur einmal berechnet werden musste, konnte die Ausgabe beliebig oft an verschiedene Geräte gesendet und ausgegeben werden, was PostScript geräteunabhängig machte. Bei der seinerzeit wachsenden Vielfalt an Ein- und Ausgabegeräten zeichneten sich vor allem die letzteren dadurch aus, dass sie sich in den vorinstallierten Zeichensätzen und Anweisungsformaten teilweise erheblich unterschieden. Bis dahin war ASCII der einzige Ausgabestandard, der nahezu geräteunabhängig war, nun aber konnten dank der von PostScript verwendeten Abstraktionsebene verschiedene Ausgabegeräte mit verschiedenen Eigenschaften und Fähigkeiten auf eine Weise angesprochen werden, dass sie eine PostScript-Datei so ausgaben, wie sie konzipiert wurde, d. h. mit Informationen über die vorgesehenen Zeichensätze und dem vorgesehenen Arrangement graphischer Elemente. Da-

bei war das Fabrikat und die Auflösung irrelevant. Die Datei konnte sowohl von niedrigauflösenden Monitoren als auch von hochauflösenden Geräten bei entsprechend guter Qualität ausgegeben werden, da PostScript als geräteunabhängiges Format durch seine Abstraktionsebene letztendlich keine endgültigen Vorgaben an das Ausgabegerät richtet, sondern es vielmehr diesem erlaubt, seine jeweiligen Auflösungseigenschaften auszunutzen.

Der erste PostScript-fähige Drucker war der LaserWriter von Apple (1985), der sowohl Spezifikationen für klassische Fonts im PostScript-Format als auch einen PostScript-Interpreter bereitstellte, der sich für die Übersetzung der PostScript-Befehle für die Druckausgabe verantwortlich zeigte. Nachdem mit der PageMaker-Anwendung von Aldus die erste ernst zu nehmende Seitenlayout-Software auf den Markt gekommen war, war durch die Verbindung von PostScript, Laserdrucktechnologie und der entsprechenden Software der Weg für das geebnet, was uns heute als *Desktop Publishing* bekannt ist, ein Begriff, der schon 1985 geprägt wurde, um die Fähigkeiten der neuen Kombination von Hard- und Software zu charakterisieren. Seither hat sich PostScript zum industriellen Standard entwickelt, wobei 1998 Adobe zufolge 75% aller kommerziellen Publikationen mit Hilfe von PostScript gedruckt wurden [7]. Es lässt sich jedoch festhalten, dass PostScript im DTP-Bereich von geringerer Relevanz ist, da originäre PostScript-Drucker relativ teuer sind. Zur Darstellung und Ausgabe von PostScript-Dateien steht jedoch die freie GhostScript-Software zur Verfügung.

Portable Document Format (PDF)

Trotz seiner Stellung als Industriestandard hat sich PostScript nicht als *ultima ratio* herausgestellt. Weiterhin bestehen im professionellen Bereich Probleme, wenn beispielsweise die Ausgaben von Programmen oder Programmversionen nicht kompatibel sind, wenn Dateien fehlerhaft oder unvollständig sind, Zeichensätze fehlen (auch PostScript verwendet eigene Fonts) usw. Insbesondere das elektronische Versenden von großen gerasterten PostScript-Dateien an eine ausgelagerte Druckerei stellt sich häufig als nicht sehr ökonomisch heraus, wenn die dazu nötige Bandbreite nicht gegeben ist, was durch die kostspieligere Nutzung von Satellitenverbindungen oder landbasierten Hochgeschwindigkeitsleitungen kompensiert wird. Zeitungen sehen sich mit Problemen konfrontiert, wenn von Werbeagenturen Fonts verwendet werden, die dem Verlag nicht zur Verfügung stehen und sich als nicht druckfähig erweisen – vergleiche [7].

Aus diesem Grund hat sich Adobe bemüht, PostScript als Ausgabestandard durch PDF abzulösen, das zwar im Grunde auf PostScript basiert, letztlich diesem gegenüber aber durchaus einige Vorteile bietet. So sind PDF-Dateien nicht nur inhärent komprimiert, sondern auch in ihrer Handhabung erheblich flexibler. Wie bereits oben angedeutet, ist eine PostScript-Datei im Grunde eine eigene Anwendung, die von einem RIP ausgeführt, d. h. in eine Bitmap übersetzt wird. In einer in PostScript geschriebenen Datei sind graphische Elemente nur mit einiger Schwierigkeit zu bearbeiten oder zu extrahieren. Weiterhin können Objekte auf einer Dokumentseite abhängig von anderen Elementen auf anderen Seiten sein, sodass die Manipulation eines Objekts durchaus Auswirkungen auf entfernte andere Objekte haben kann. Die Bearbeitung einer PostScript-Datei erfordert somit entweder Kenntnis der Programmiersprache oder einen Eingriff in die Quelldatei, die der entsprechenden PostScript-Datei zu Grunde liegt.

PDF hingegen ist ein objektbasiertes Format, das mit der entsprechenden Anwendung leicht neu ediert, korrigiert oder ergänzt werden kann. Es besteht die Möglichkeit, das Dokument zu durchsuchen und Hyperlinks einzufügen. Wichtiger noch ist die Tatsache, dass trotz der komprimierten Größe der PDF-Datei das PDF-Master Informationen über alle verwendeten Zeichensätze, Graphiken und Photos enthält. Selbst wenn das Dokument nicht in Form einer PDF-Datei vorliegt, ist es oftmals unproblematischer, es in PDF zu übersetzen und auszugeben, als das Ausgabegerät auf anderem Wege, beispielsweise mit Informationen über den verwendeten Zeichensatz, zu versorgen.

METAFONT, TEX und LATEX

Der bisher beschriebene digitale Arbeitsprozess zeichnet sich insbesondere dadurch aus, dass er dem eigentlichen Autor des zu druckenden Textes verborgen bleibt. Durch die ständig fortschreitende Entwicklung im DTP-Bereich sind aber in zunehmendem Maße die Grenzen zwischen dem Autor als Produzenten des Textinhalts und dem Typographen und Setzer als Produzenten der endgültig manifesten Textgestalt aufgelöst. Bevor diese Problematik eingehender diskutiert werden soll, werde ich anhand des Beispiels von Donald Knuths METAFONT, seines Satzsystems TEX und dessen Weiterentwicklung LATEX von Leslie Lamport darstellen, wie der Autor in Abhängigkeit von der logischen Struktur seines Textes gleichzeitig nicht nur vollständige Kontrolle über den Inhalt sondern auch über die Form des Dokuments ausüben kann.

Das Setzen eines „gewöhnlichen“ Textkörpers mit einem System wie Monotype war zwar eine anspruchsvolle, aber auch routinemäßig Aufgabe für einen ausgebildeten Schriftsetzer. Komplizierter verhielt es sich hierbei beispielsweise schon beim Setzen mathematischer Formeln, bei dem mehrere Arbeitsgänge vonnöten waren. Beim Setzen einer Formel wie

$$|\det(a_{ij})| \leq \prod_{1\leq i\leq n} \left(\sum_{1\leq j\leq n} a_{ij}^2 \right)^{1/2}$$

war es beispielsweise nötig, dass die Zeichen auf der Grundlinie und die Superskripte und dann erst in einem zweiten Schritt die Subskripte eingegeben wurden. Hierbei musste vom Setzer erwartet werden, dass er die Größe und Breite eines jeden Zeichens kannte, sodass jedes nachträglich eingefügte Zeichen passend integriert werden konnte. Erst nachdem diese erst vorläufige Formel in Metall gegossen worden war, wurden von Hand die großen Symbole wie Klammern oder Summen- und Produktzeichen eingefügt. Dies alles erforderte natürlich die erfahrene Hand eines bzw. mehrerer Experten, jedoch war das Ergebnis dieses aufwändigen Prozesses bis dahin unübertroffen, auch durch die neuen Photosatzmaschinen, die in den 60er Jahren langsam die veraltenden mechanischen Methoden zu verdrängen begannen, vgl. hierzu [3].

Einem Mathematiker wie Donald Knuth, der ein großes Interesse daran hatte, dass seine Bücher mit der bestmöglichen Qualität verlegt wurden, musste so der langsame Abschied von der hochwertigen Monotype-Technik einige Kopfschmerzen bereiten, denn das Ergebnis des Photosatzes bei der anstehenden Neuauflage des zweiten Bandes seines Buchs *The Art of Computer Programming* erreichte bei weitem nicht die vorher bekannte Qualität. Als Mathematiker und Informatiker boten sich Knuth jedoch einige Zeit später durch die neu aufkommende digitale Methode völlig neue Verständnismöglichkeiten:

> The newest machines made images on film by *digital* instead of analog means—something like the difference between television and real movies. The shapes of letters were now made from tiny little dots, based on electronic pulses that were either ON or OFF [...]. Aha! This was something I could understand! It was very simple, like the lights on a scoreboard at a sports match. [4]

Von diesem Ausgangspunkt war es schließlich nur noch eine Frage, Bildpunkte adäquat zu positionieren und sie von einem hochauflösendem Gerät ausgeben

zu lassen – die Typographie als künstlerisches Handwerk lag plötzlich in den Händen von Informatikern.

METAFONT

Knuth selbst beschloss kurzerhand, sich in typographische Grundlagen einzuarbeiten, seine eigenen Programme zu schreiben, die ihn befähigen würden, Zeichensätze zu generieren, zu setzen und somit seine Bücher mit der höchstmöglichen Qualität verlegen zu lassen. Nach einigen fruchtlosen Versuchen, die alten Monotype-Zeichensätze anhand von Photographien und Filmaufnahmen zu rekonstruieren, begann Knuth die Arbeit an einer *high-level*-Programmiersprache, die jeden Glyph (d. h. jedes einzelne Zeichen) eines Zeichensatzes auf einem diskreten Raster zeichnete, ebenso wie dies von Hand mit einem Stift geschehen würde. Nachdem Knuth von nur einigen Monaten Arbeit ausgegangen war, erforderte es schließlich knappe zehn Jahre und die Zusammenarbeit mit angesehenen Typographen wie Hermann Zapf u. a., bis sein Programm METAFONT (zusammen mit dem dazu gehörigen Satzsystem TEX) so weit entwickelt war, dass er seine Bücher zu seiner vollen Zufriedenheit veröffentlichen konnte.

Die METAFONT zu Grunde liegende Idee ist es, für jeden Glyph zwar eine explizite Beschreibung in Form eines Programmcodes zu liefern, dabei aber auch die Regeln, die die Zeichnung steuern, durch Parametrisierung flexibel zu halten. Ein „Meta-Font“ definiert sich so als eine schematische Beschreibung der Art und Weise, wie der Zeichensatz zu zeichnen ist. Damit bezeichnet der Begriff nicht die individuellen Zeichnungen selbst, sondern die übergeordneten Eigenschaften einer Familie von Zeichensätzen, die die individuellen dazu gehörigen Schriften charakterisieren. So beinhalten Zeichensatzfamilien im Allgemeinen kursive, fette, fett-kursive Schriften etc. Für die Caslon-Familie beispielsweise sind nicht weniger als ca. 270 Schriften gezählt worden [5]. In Anlehnung an die in den früheren Auflagen seiner Bücher verwendeten Monotype-Modern-Zeichensätze entwarf Knuth schließlich eine hochwertige Gruppe von Fonts, die als „Computer Modern“ bekannt sind und noch heute in der angloamerikanischen Welt als Standardausgabe von TEX dienen. Computer Modern (CM) schloss bereits 1981 Meta-Fonts für griechische und kyrillische Zeichen ein, des Weiteren eine beachtliche Menge von mathematischen Zeichen und Symbolen, deren Verwendung TEX und LATEX gerade im mathematisch-naturwissenschaftlichen Bereich zum Durchbruch verholfen hat.

Für europäische Bedürfnisse hat sich allerdings herausgestellt, dass die von Knuth entworfenen CM-Fonts in ihrem Umfang zu eingeschränkt waren. Zwar können damit viele europäische Sonderzeichen, wie zum Beispiel die deutschen Umlaute problemlos von TEX und LATEX dargestellt werden, dies aber nur mit Hilfe zusammengesetzter Zeichen und unter daraus resultierenden möglichen Problemen mit dem Trennalgorithmus, da der Trennprozess durch Befehle wie `\accent` unterbrochen wird. So wurde 1990 auf der in Cork in Irland von der TUG abgehaltenen TEX-Konferenz ein Kodierungsschema definiert, das im Gegensatz zu Knuths alten Fonts nicht nur 128, sondern nun 256 Glyphen umfasst und somit viele der Sonderzeichen, die von den auf dem lateinischen Alphabet basierenden europäischen Schriften benötigt werden. Dies gilt beispielsweise für die im Isländischen immer noch gebräuchlichen Zeichen þ und ð, die nicht mittels zusammengesetzter Zeichen dargestellt werden können. Aber dennoch ist auch dieses Schema nicht vollständig – für die Darstellung des walisischen „ŵ“ muss so zum Beispiel immer noch auf die Zusammensetzung zweier Zeichen zurückgegriffen werden. Die diesem Kodierungsschema (T1 für *TEX-1-Encoding*, auch *Cork Encoding*) entsprechenden Zeichensätze sind die 1997 veröffentlichten EC-Fonts (*Enhanced Coding*). Um den EC-Zeichensatz zu vervollständigen stehen die TC-Fonts (*Text Companion*) zur Verfügung, die weitere Sonderzeichen und Symbole enthalten.

Knuths METAFONT ist insbesondere durch zwei Eigenschaften, die es von anderen Formaten abhebt, charakterisiert. Erstens werden in METAFONT nicht die Umrisse einzelner Glyphen definiert, wie dies beispielsweise bei PostScript-Fonts (Type1) der Fall ist, sondern vielmehr wird jeder Glyph mit Hilfe von Pinselstrichen, verschieden bemessenen Stiften und Radierern „gezeichnet“, sodass das Resultat letztlich als Bitmap ausgegeben wird. Diese Methode ist extrem flexibel, beachtet man beispielsweise die Tatsache, dass die Größe des „Stifts“ beim Zeichnen der einzelnen Glyphen variieren kann. Referenzpunkte auf dem Raster können durch Parametereinstellungen definiert werden, wodurch auf relativ einfache Weise eine ganze Zeichensatzfamilie erstellt werden kann, die durch individuelle Schriften repräsentiert wird. Hieraus ergibt sich unter anderem, dass METAFONT skalierbare Zeichensätze erzeugt, sodass beispielsweise Glyphen bei einer Größe von 12 pt proportional gesehen breiter sind und sich so von ihren entsprechenden Gegenstücken bei 24 pt leicht unterscheiden, was in einer besseren Lesbarkeit auch recht kleiner Schriftgrößen resultiert. Die Flexibilität von METAFONT wird schließlich dadurch erhöht, dass der eigentliche „Meta-Font“ in eigenen Basisdateien spezifiziert wird, die grundlegende Beschreibungen der einzelnen

Glyphen enthalten. Weitergehende Parametereinstellungen werden hingegen in anderen Dateien, die bei Bedarf von METAFONT aufgerufen werden, abgelegt. Als Nachteil im Vergleich zu sogenannten Vektorfonts wie PostScript-Type1 bleibt jedoch festzuhalten, dass METAFONT erst ab einer Auflösung von 300 dpi ein zufrieden stellendes Ergebnis liefert, wohingegen PostScript, wie bereits oben angedeutet, die Möglichkeiten des jeweiligen Ausgabegerätes voll ausnutzen kann.

Zweitens erlaubt es die Variierung in den Parametereinstellungen mit METAFONT, verschiedene Faktoren zu beeinflussen, die die letztendliche Form der Zeichen bestimmen, wovon an dieser Stelle jedoch nur einige exemplarisch genannt seien. Für die Bestimmung der Unterlängen stehen einzelne Parameter für die Buchstaben g j p q y zur Verfügung, andere wiederum für Kommata und den Buchstaben Q. Unter- und Überlängen werden dabei unabhängig von der Oberlänge (*x-height*) eines Zeichensatzes definiert, und ein weiterer Parameter bestimmt nicht nur die Höhe des Querstriches im Buchstaben e, sondern sorgt gleichzeitig für die Angleichung anderer damit zusammenhängender Maße, beispielsweise die Rundung der Schulter im Buchstaben h. Für eine weitere Beschreibung einzelner Parameter und deren Wirkung sei jedoch auf den höchst lesenswerten Artikel "The Concept of a Meta-Font" von Donald Knuth verwiesen [5].

TEX

METAFONT allein war jedoch nicht ausreichend, um Knuth aus seinem Dilemma zu befreien, da die von METAFONT gezeichneten Buchstaben und Symbole allein im Prinzip nutzlos waren:

> There was also a chicken-and-egg problem. I couldn't set type until I had fonts of letters and mathematical symbols, but the fonts I needed did not exist in digital form. And I could not readily design the fonts until I could set type with them. I needed both things at once. [4]

Knuth, der von den hochwertigen Ergebnissen der alten Monotype-Maschine verwöhnt war, entwickelte also parallel zu METAFONT das Satzsystem TEX, das die neuen Meta-Fonts einbinden und somit seinen typographischen Ansprüchen gerecht werden konnte. Dabei verfolgte Knuth zwei Zielsetzungen [3]: Erstens wollte er sich 1977, als er die Arbeit an TEX und METAFONT begonnen hatte, nicht damit zufrieden geben, Dokumente zu produzieren, die „gut aussahen". Knuth wollte nichts Geringeres als dass seine Dokumente die

qualitativ hochwertigsten waren, die man mittels eines Computers erzeugen kann. Wie schon oben erwähnt, brauchte es so auch mehrere Jahre, bis diese beiden Systeme halbwegs zur Zufriedenheit ihres Autors arbeiteten. Das zweite Ziel lag in der Unabhängigkeit der beiden Systeme von Ort und Zeit. TEX und METAFONT sollten auch noch lange Zeit nach ihrer „Vollendung“ gleichwertige Ergebnisse auf nahezu jeder verfügbaren Computer-Plattform erzielen – Knuth selbst sprach seinerzeit von nicht weniger als 100 Jahren.

Mittlerweile hat Knuth zwar die Arbeit an beiden Systemen eingestellt, sie aber 1986 vollständig in seiner fünfbändigen Dokumentation *Computers and Typesetting* veröffentlicht, sodass sie für Weiterentwicklungen zur Verfügung stehen, was vor allem in der Entwicklung verschiedenster Makropakete resultierte, von denen LATEX das wohl bekannteste sein dürfte. Die umfangreiche Menge an existierenden Makropaketen, aber auch die ständig anwachsende Menge an TEX-kompatiblen Zeichensätzen hat sicherlich dazu beigetragen, dass TEX bedeutende Maßstäbe setzen konnte, sich aber dennoch in sich selbst erschöpft. Die derzeitige Entwicklung von $\mathcal{NTS}$ (New Typesetting System), einem Java-basierten Satzsystem, das zwar in seiner ersten Version TEX-kompatibel sein, aber dessen Möglichkeiten noch übersteigen soll, zeigt, dass Knuth immer noch auf dem besten Weg ist, seine Ziele zu erreichen.

Im Grunde kann TEX als eine Interpreter-Sprache zum Zweck des Setzens verschiedenster Dokumente angesehen werden. Eine TEX-Datei enthält neben dem zu setzenden Text Instruktionen in Befehlsform darüber, *wie* dieser Text zu setzen ist. Hieraus ergibt sich, dass TEX so konzipiert wurde, dass der Autor des Textes gleichzeitig der Verfasser der zu interpretierenden TEX-Datei ist und somit in Abhängigkeit der logischen Struktur des Textes Kontrolle über dessen Form ausüben kann – hierüber aber später mehr.

Zum Zeitpunkt der Fertigstellung von Knuths *TEXbook* (1983) bestand TEX aus ungefähr 900 Befehlen oder Kontrollsequenzen – eine ungeheuere Menge, wenn man bedenkt, dass Programmiersprachen wie C maximal etwa 200 Befehle umfassen. Ungefähr 300 dieser Befehle sind *low-level*-Befehle (*primitives*), die das „Grundgerüst“ der TEX-Sprache repräsentieren. Aus ihnen sind weitere 600 *high-level*-Befehle (Makros) zusammengesetzt, die den Umgang mit TEX erheblich erleichtern. Beispielsweise ist der Befehl `\bigskip` ein Makro für die Befehle `\vskip \bigskipamount`, wobei `\bigskipamount` in TEX einem Maß von 15 pt und dem Befehl `skip15` entspricht (in LATEX sind es 19 pt, also `\skip19`). Im Falle von TEX bewirkt `\bigskip` somit, dass der Abstand zwischen zwei Zeilen an der entsprechenden Stelle 15 pt beträgt.

Makros können, wie gerade demonstriert, eine recht einfache Form annehmen, aber auch sehr komplex sein und beispielsweise Anweisungen über das Setzen von Kapitelüberschriften bis hin zu Informationen über die globale Gestaltung des jeweiligen Dokuments beinhalten.

Als Wissenschaftler und Autor entwickelte Knuth trotz der Fülle an Befehlen, die TEX charakterisieren, eine Eingabesprache, die von anderen Benutzern (insbesondere, aber nicht ausschließlich, Autoren wissenschaftlicher Literatur) fast intuitiv verstanden werden konnte, und mit deren Hilfe sich professionell gesetzte Dokumente erzeugen lassen. Die Handhabung von Textelementen, wie sie vor allem in wissenschaftlicher Literatur Verwendung finden (mathematische Formeln, Fußnoten, Gleitobjekte und dergleichen), durch TEX entspricht typographischen Standards, wie sie von gewöhnlichen Textverarbeitungsanwendungen bisher nicht erreicht werden konnten. Der Erfolg, den dieses Satzsystem bereits in seiner Entwicklungsphase erfahren hatte, wurde schließlich durch zahlreiche Erweiterungen in Form von Makropaketen noch weiter gefestigt. LATEX von Leslie Lamport ist hierbei sicherlich das bekannteste und am weitesten verbreitete, da es die leistungsfähige, umfangreiche aber nicht immer einfach zu handhabende TEX-Sprache einem Publikum zugänglich macht, das sich weder mit detaillierten Programmierkenntnissen noch mit technischen Fragen des Dokumentlayouts notwendigerweise auseinander setzen muss. Neben LATEX stehen weitere Pakete zur Verfügung, die beispielsweise dem Setzen phonetischer Transkriptionen (TIPA), der verbesserten Darstellung mathematischer Formeln ($\mathcal{AMS}$-TEX), musikalischer Notationen, chemischer Formeln usw. dienen. Nicht zuletzt hat die Tatsache, dass TEX und die dazugehörigen Makropakete frei erhältlich sind, sein Übriges zum Erfolg von TEX beigetragen.

TEX at work

Wurde bisher von METAFONT und TEX als „Programmiersprachen“ gesprochen, so bleibt festzuhalten, dass dies nur ein Teil der ganzen Wahrheit ist. Die Menge der Befehle und Kontrollsequenzen, mit denen sich der Benutzer der beiden Systeme (oder genauer: der Autor von METAFONT- und TEX-Dateien) konfrontiert sieht, stehen dem Computer nicht in ebendieser Form zur Verfügung, sondern müssen von ihm interpretiert werden. Ähnlich wie im Falle einer Markup-Sprache wie HTML handelt es sich bei den Kommandosequenzen bei METAFONT und TEX um Konventionen, die es dem Benutzer ermöglichen und erleichtern, mit dem eigentlichen Interpreter zu kommunizieren. Die eigentlichen, METAFONT als auch TEX zu Grunde liegenden Sprachen

sind so konzipiert, dass sie verschiedene solcher Konventionen verstehen und interpretieren können, von denen die bisher dargestellten nur eine von vielen Möglichkeiten repräsentieren. Ebenso wie TEX besteht METAFONT somit aus einer Reihe von *low-level*-Befehlen und einer Menge von Makroerweiterungen, die beim Aufrufen des Programms üblicherweise geladen werden. In beiden Fällen konstituiert die Menge dieser Makros eine für das System verständliche Konvention, auf die für gewöhnlich mit „plain“ METAFONT bzw. „plain“ TEX Bezug genommen wird. Der Autor eines Dokuments, das mit Hilfe von TEX verfasst wird, braucht sich für gewöhnlich weder mit den Details von METAFONT noch mit den TEX-Primitiven auseinanderzusetzen, da beim „Übersetzen“ der (plain-)TEX-Datei METAFONT automatisch aufgerufen wird. Es mag dennoch von Interesse sein zu sehen, wie dieses Übersetzen etwas genauer funktioniert.

Angenommen, METAFONT wird aufgerufen und aufgefordert, einen Zeichensatz zu generieren, was entweder durch Aufrufen des Programms selbst durch den Befehl `mf` oder automatisch beim Interpretieren einer TEX-Datei erfolgen kann, wobei wir letzteres an dieser Stelle voraussetzen wollen. Grundlage für das Zeichnen des Fonts durch METAFONT sind die oben bereits erwähnten Dateien, die die Anweisungen enthalten, wie einzelne Glyphen zu zeichnen sind. Diese liegen in Form von Textdateien vor und sind somit jederzeit modifizierbar. Ihre Dateierweiterung ist `mf`. METAFONT wird eine weitere Ausgabe in Form von zwei Dateiformaten produzieren, von denen eine Datei die Bitmap-Zeichnungen des Zeichensatzes(`gf`-Dateien, für *Generic Font*), die andere Informationen über die Dimensionen, Ligaturen und Kerning-Eigenschaften, die dem Zeichensatz zu Eigen sind (`tfm` für *TEX Font Metric*), enthält . `tfm`-Dateien sind geräteunabhängig und müssen für jeden Zeichensatz nur einmal erzeugt werde, sodass sie für TEX jederzeit verfügbar sind, wenn sie einmal vorliegen.

Die Erweiterung für `gf`-Dateien ist für gewöhnlich `*gf`, wobei an Stelle des `*` in Unix-Kontexten eine Zahl für die berechnete Auflösung der Bitmaps steht, wie beispielsweise `*.300gf`, wobei dieser Font eine Auflösung von 300 dpi aufweist (da MS-DOS lediglich dreistellige Erweiterungen erlaubt, werden beispielsweise in WINDOWS die entsprechenden Dateien je nach Auflösung in verschiedenen Ordnern abgelegt). Wird METAFONT von TEX aufgerufen, wenn eine Standard-TEX-Datei übersetzt und ausgegeben werden soll, so wird auf eine komprimierte Fassung des Zeichensatzes zugegriffen (`pk`), im konkreten Fall vielleicht auf einen Computer-Modern-Zeichensatz aus der „Roman“-Zeichensatzgruppe mit einer Größe von 12 pt bei einer Auflösung von 600 dpi.

Die entsprechende Datei heißt somit `cmr12.600pk`. Die `gf-` bzw. die entsprechende `pk`-Datei enthält hierbei eine kompakte binäre Repräsentation eines digitalisierten Zeichensatzes (d. h. die Bitmaps), die alle relevanten Informationen einschließt, die letztendlich vom Treiber des Ausgabegerätes benötigt werden, um den Zeichensatz darzustellen [2]. Somit existieren für jeden Zeichensatz mit individuellen Größen und Auflösungen eigene `gf`-Dateien (meist aber komprimiert als `pk`), die zum letztendlichen Ausdruck in ein anderes Format übersetzt werden können.

Wird ein Dokument von TEX interpretiert, sind die Bitmaps aus den `gf`-Dateien aber erst einmal ohne Relevanz. Vielmehr liest TEX vorerst lediglich die Metriken aus den `tfm`-Dateien ein, die Informationen über die individuellen Zeichenabmessungen, die relative Wortabstände und der gleichen enthalten, nicht aber den Zeichensatz selbst, der ja in den `gf`-Dateien gespeichert ist. Intern generiert TEX aus den einzelnen Zeichen des Eingabetextes eine lange Sequenz von so genannten *tokens*, auf deren Grundlage und mit Hilfe der Informationen aus der entsprechenden `tfm`-Datei TEX schließlich versuchen wird, den optimalen Zeilen- und Seitenumbruch zu erzeugen, indem es an den dafür vorgesehenen Stellen Umbruchpunkte einfügt.

Wie oben bereits angedeutet, zeichnet METAFONT auf einer (natürlich nicht sichtbaren) Matrix, und für die Berechnung der Umbruchpunkte ist es für TEX wichtig, die individuellen Größen dieser „Kästchen“ zu kennen, damit die so erzeugten Zeichen ihrer Größe entsprechend angeordnet und Zeilen und Seiten korrekt umgebrochen werden können. Somit beinhaltet die `tfm`-Datei traditionelle typographische Grundeinheiten wie die Höhe des `ex` – das x bestimmt traditionell die Höhe der anderen Kleinbuchstaben in einem Zeichensatz, von Über- und Unterlängen abgesehen – und die Breite des `em` (ebenso wie ein `ex` ist ein `em` eine typographische Grundeinheit, die die Breite des M in einem Zeichensatz beschreibt. Zu Zeiten des mechanischen Buchdrucks füllte das M die ganze Fläche des quadratischen Druckstempels aus, und somit steht ein `em` auch für die Punktgröße des jeweiligen Zeichensatzes). Die Zeichensatzmetrik beinhaltet weiterhin Informationen über Unter- und Überlängen der einzelnen Zeichen sowie über Ligaturen, Unterschneidungen usw. Die Ligaturanweisungen in der `tfm`-Datei werden, in Knuths Worten, etwa folgendermaßen an TEX übermittelt:

> Dear TEX, when you're typesetting an “f” with this font, and when the following character also belongs to this font, look at it closely because you might need to do something special: If that following character is

> another “f”, replace the two f's by character code `oct"013"` (namely “ff”); if it's an “i”, retain the “f” but replace the “i” by character code `oct"020"` (a dotless “ı”). [2]

Informationen über Unterschneidungen (*Kerning*) betreffen die Abstände zwischen den einzelnen Zeichen (Glyphen) in einer Textzeile. In einer Kombination von W und o ist es nicht nur ästhetisch wertvoll, das o so nah an das W zu rücken, dass es nicht nur neben dem W erscheint, sondern auch teilweise von ihm überragt wird, man vergleiche beispielsweise den feinen aber wirkungsvollen Unterschied zwischen „Wo“ und „Wo“. Zweck des Kernings ist es, auch mit einer Proportionalschrift einen gleichmäßigen Zeichen- und Wortabstand zu erreichen, um die Lesbarkeit des Textes zu erhöhen. Dies ist für Proportionalschriften von Wichtigkeit, da in ihrem Fall die Breite der Glyphen unterschiedlich bemessen ist, anders als bei nicht-proportionalen Schriften wie Schreibmaschinenschriften, bei denen auch der Wortabstand gleich der Zeichenbreite ist. Zu diesem Zweck stehen in den Metrikinformationen sogenannte Kerning-Paare zu Verfügung (wie beispielsweise W und o), die von METAFONT aufgerufen werden, aber auch innerhalb von TEX beliebig manipuliert werden können (das TEX-Logo selbst gibt ein gutes Beispiel der Kerningfunktion).

Für jeden Zeichensatz muss entsprechend nur eine `tfm`-Datei erzeugt werden, die von TEX eingelesen wird, um eine geräteunabhängige Datei zu erzeugen, die schließlich an einen Ausgabetreiber weitervermittelt werden kann. Diese Datei (`dvi` für *device independent*) kann in der nun vorliegenden Form weder am Bildschirm angezeigt noch ausgedruckt werden, da sie den eigentlichen Zeichensatz (die Bitmaps) noch nicht enthält. Erst beim Starten eines DVI-Treibers, beispielsweise beim Aufrufen eines DVI-Viewers oder beim Umwandeln in eine druckfähige PostScript-Datei (beispielsweise mit `dvips`) wird METAFONT aufgerufen und werden die `pk`-Dateien eingelesen oder gegebenenfalls erzeugt, wenn sie in der entsprechenden Auflösung oder Größe noch nicht vorliegen.

Durch die jederzeit mögliche Modifizierung der Zeichenspezifikationen in den `mf`-Dateien, die Entwicklung neuer Definitionen (Makros) und den flexiblen Einsatz der Markup-Instruktionen beim Verfassen einer TEX-Datei hat somit der Autor eines Dokuments vollständige Kontrolle nicht nur über Inhalt und die logische Form seines Textes, sondern auch über Design und Layout. Der Verfasser vereint somit Autorenschaft, Designer und Setzer des Textes in einer Person, ähnlich wie es bei sogenannten WYSIWYG-Textverarbeitung („what

you see is what you get“) der Fall ist – abgesehen von der Tatsache, dass TEX und LATEX keine WYSIWYG-Anwendungen sind. Entwicklungen wie LYX oder TEXmacs versuchen jedoch, die Vorteile beider Systeme zu verbinden. Die Frage nach dem Für und Wider der verschwindenden Grenzen zwischen Autor und Typograph soll im nächsten Abschnitt vertieft werden.

Der Autor als Typograph

Arbeitsteilung

Seit der Zeit der gelernten Manuskriptschreiber und dem Aufkommen des Druckerhandwerks war es eine Selbstverständlichkeit, dass Autorenschaft (logische Textgestaltung) und Produktion der manifesten, also zu erscheinenden Textgestalt als verschiedene Aufgaben angesehen wurden, die dementsprechend von verschiedenen Personen bzw. Personengruppen ausgeführt wurden. Die Anforderungen, die an Typographen, Setzer und Drucker gestellt wurden, waren seit jeher ohne besondere Ausbildung und eine gewisse Begabung nicht zu erfüllen. Zu Zeiten, in denen Monotype und Linotype noch den Standard der Satztechnik vorgaben (bis in die 60er Jahre des 20. Jahrhunderts hinein) galt Typographie und Textsatz somit noch als künstlerisches Handwerk oder als „nützliche“ Alltagskunst, wie sie beispielsweise von William Morris' *Arts & Crafts* Bewegung demonstriert wurde.

Dieses über mehr als 500 Jahre gewachsene Handwerk hat trotz der wenigen (wohlgemerkt prinzipiellen) technischen Veränderungen zu Einsichten hinsichtlich typographischer Gestaltung und menschlicher Lesegewohnheiten geführt, die – wie heutzutage von professionellen und traditionsbewussten Typographen befürchtet – durch die Entwicklungen im Bereich des *Desktop Publishing* (DTP) immer mehr in den Hintergrund des Produktionsprozesses eines Textes rücken. Dieser Produktionsprozess beinhaltet traditionell, wie bereits angedeutet, die Arbeit verschiedener Personen oder Personengruppen. Der Autor gilt als Verfasser eines Textes, d. h. er ist für die logische Textstrukturierung verantwortlich. Dies umfasst unter anderem die Wahl der sprachlichen Ausdrucksform seiner Ideen und die Unterteilung dieser Ausdrucksform in gedanklich einheitliche Abschnitte, Unterkapitel, Kapitel usw. Alle diese Texteinheiten stellen verschieden bemessene, textuelle Träger der Ideen und Gedanken des Autors dar und sind somit von ihm als Teil seines Beitrags der Textproduktion mit Bedacht zu wählen. Gleiches gilt, insbesondere in wissenschaftlichen Texten, für Fußnoten, Blockzitate oder beispielsweise die Positionierung von Graphiken oder Illustrationen.

Anhand von Anmerkungen des Autors im Text diente die logische Strukturierung des Textes schließlich dem Typographen und Setzer als Grundlage für weitergehende Entscheidungen bezüglich der beabsichtigten Textgestalt. Auf Grund der Art des Textes (ist es ein wissenschaftlicher, ein literarischer Text, ein Essay oder ähnliches?) und dessen vom Autor vorgegebener Strukturierung ist zu entscheiden, welche Schriftarten und -größen verwendet werden, wie die Gestaltung von Kapitel- und Unterkapitelüberschriften auszusehen hat, welche Zeilenlänge angezeigt ist und dergleichen mehr.

Diese Arbeitsteilung war bis zum Aufkommen des DTP vorherrschend, was ganz offensichtlich auch darin begründet ist, dass dem Autor im Allgemeinen bis dahin weder die Mittel noch die Ausbildung zur Verfügung standen, um seine eigenen niedergeschriebenen Gedanken in eine manifeste, zur Veröffentlichung bereite Textgestalt zu übersetzen. Um es in anderen Worten auszudrücken: „Composition and logical structuring of text is the author's specific contribution to the production of a printed text. Typesetting is the typesetter's business“ [1].

Im Allgemeinen besteht das Ziel eines Textes in der Kommunikation von Ideen, Vorstellungen oder Wissen des jeweiligen Autors. Seine Aufgabe ist es, den Inhalt seines Textes sprachlich und logisch zu strukturieren. Aufgabe des Typographen ist es, mit Hilfe *seines* Wissens über Eigenschaften der verschiedensten textlichen Elemente, wie Schriftarten, Schriftgrößen, Bemessung von Rändern, Zeilenabständen usw., und darüber hinaus auch mit Hinsicht auf den Vorgang des Lesens und der Lesegewohnheiten die strukturierte Vorgabe des Autors in eine möglichst optimale textliche Repräsentation zu übersetzen. Um diese Aufgabe erfüllen zu können, steht dem Typographen somit ein Wissen zur Verfügung, das dem Autor für gewöhnlich vorenthalten ist.

Typographische Aspekte

In der Auswahl des Schriftdesigns und des Seitenlayouts spiegelt sich damit idealerweise die Tatsache wider, dass dem Lesevorgang sowohl auf physiologischer als auch auf kognitiver Ebene verschiedene Regeln zu Eigen sind, die auf der Ebene der typographischen Textproduktion beachtet werden müssen, wie die folgenden Beispiele verdeutlichen. So werden Wörter weder einzeln noch Zeichen für Zeichen, sondern als Ganzes in Wortgruppen decodiert, sodass die relativ schnelle Augenbewegung beim Lesen einer Zeile etwa alle zwei bis drei Wörter pausiert. Dieser immer noch sehr flüssige Bewegungsvorgang wird dann erst ins Stocken gebracht, wenn ein Wort mit einer wenig geläu-

figen Zeichensequenz im Text erscheint, wie beispielsweise ein unbekanntes Fremdwort. Während die Augen beim Lesen über die Zeilen gleiten, fokussieren sie die obere Hälfte der Zeichen, die somit für die kognitive Verarbeitung von besonderer Bedeutung sind. Zeilenlänge, Schriftgröße und Zeilenabstand müssen sorgfältig ausgewählt werden, damit die Augen nicht allzu sehr beansprucht werden und auch zuverlässig von einer Zeile zur nächsten wechseln können, ohne dem Leser das Lesen zu erschweren [8].

Von Bedeutung sind weiterhin kulturspezifische Lesegewohnheiten, die räumlich und zeitlich unterschiedlichen Veränderungen ausgesetzt sind. So rechtfertigt die Tatsache, dass wir es gewohnt sind, von links nach rechts zu lesen die Auswahl einer horizontal ausgerichteten Serifenschrift zumindest in Textdokumenten. Serifen (die kleinen „Häkchen“ an den Buchstaben) helfen dem Auge in seiner vertikalen Bewegung, den Blick auf die aktuelle Zeile zu behalten. Eine vertikal ausgerichtete serifenlose Schrift ist im Gegensatz hierzu lediglich bei markanten, kurzen Textelementen wie Überschriften, Titeln oder in Werbeanzeigen angebracht. Würden wir jedoch nicht von links nach rechts, sondern von oben nach unten lesen, wäre eine serifenlose Schrift auch in Textdokumenten einer Serifenschrift vorzuziehen. Interessant ist hierbei zu bemerken, dass viele Betriebssysteme und Computeranwendungen (abgesehen von WYSIWYG-Software) weiterhin auf serifenlose Schriften zurückgreifen, die dann beispielsweise als Terminalschriften verwendet werden. Die Antwort auf die Frage, warum hier trotz ihrer besseren Lesbarkeit keine Serifenschriften verwendet werden, liegt in dem fast trivial erscheinenden Grund, dass bei vielen Bildschirmen die Auflösung nicht ausreichend ist, eine solche Schrift auch in kleinen Größen sauber darzustellen.

Lesegewohnheiten sind *per definitionem* Veränderungen unterworfen, die aber nicht immer ohne weiteres durchzusetzen sind. Es wurde bereits gesagt, dass der frühe Buchdruck versuchte, die Schriften der Manuskriptschreiber zu imitieren, dies nicht zuletzt mit dem Ziel, die neue Technik einem Publikum vertraut zu machen, das die Einführung einer neuen *Druck*-Schrift vielleicht als Anzeichen von Teufelswerk in der Buchproduktion angesehen hätte. Neuentwicklungen im Bereich der Typographie und des Druckhandwerks stoßen somit häufig auf ein recht konservatives Publikum. In der Zeit nach ihrem ersten Erscheinen in England (Egyptian, 1816) wurden serifenlose Schriften als „Grotesken“ bezeichnet, da ihre nüchterne Erscheinung als unschön angesehen wurde. In Amerika wurden sie schließlich als *gothic* bezeichnet – zu einer Zeit also, da man die Gotik mit eher schauderhaften Vorstellungen as-

soziierte, man vergleiche zum Beispiel auch auch den Begriff der seinerzeit so populären *Gothic Novel*.

So fällt es uns auch heute nicht immer leicht, die jahrhundertelang in Deutschland gebräuchlichen gebrochenen Schriften (wie beispielsweise die Fraktur) zu entziffern, und oftmals bedarf es einer paläographischen Schulung um ein altes Manuskript oder einen Frühdruck lesen zu können. Daß diese Schriften aber nicht immer als „unlesbar“ galten, wie es heute nicht selten intuitiv behauptet wird, zeigt sich allein schon in dem langen Zeitraum, in dem sie hier verwendet wurden.

Lesegewohnheiten und Schrifttypen können darüber hinaus der kulturellen und traditionellen Identifizierung dienen – gebrochene Schriften dienen nicht nur deutschen, sondern auch internationalen traditionsbewussten Zeitungen als Titelschriften. Insbesondere die gebrochenen Schriften werden dabei oftmals mit einem gewissen „Deutschsein“ assoziiert, wie man auch mit einem gewissen Humor feststellen muss, wenn man bei der Lektüre von Asterix-Comics Goten begegnet, die in gebrochenen Schriften „reden“. Interessanterweise bleibt dennoch festzuhalten, dass es Hitler selbst war, der Anfang der 40er Jahre den Befehl gab, den Gebrauch gebrochener Schriften zu Gunsten von Antiqua-Schriften zu unterlassen, sodass in den von den Nazis besetzten Gebieten schriftliche Anschläge und Ankündigungen nicht zu Leseschwierigkeiten und somit zu Nichtbeachten führen würden.

Arbeitsteilung?

Dass ein guter Autor nicht notwendigerweise auch (sein eigener) guter Typograph sein muss, spiegelt sich in der Motivation wider, die Oxford University Press (OUP) bereits 1984, also noch vor dem Auftauchen des Begriffs des *Desktop Publishing* veranlasst hat, ein Büchlein herauszugeben, das die *Oxford Rules for the Preparation of Text on Microcomputers* enthält. OUP beabsichtigte damals schon, dem Autor von Texten mittels eines Computers vorsichtig mitzuteilen, welche typographischen Aspekte seines gedruckten Textes unter seine Kontrolle fallen, aber zur gleichen Zeit auch festzustellen, dass er keinen Einfluss auf die Darstellung signifikanter Textelemente wie Zeichensätze oder das Überschriftenformat haben wird, da die Beurteilung solcher Details unter die Ägide eines professionellen Typographen fällt, vgl. [8].

OUP schien seinerzeit bereits erkannt zu haben, dass die gerade im Aufkommen begriffene Art und Weise der Textproduktion dem Autor beim Verfassen

und Setzen seines Textes eine Expertise abverlangt, die er mangels besseren Wissens oftmals nicht haben kann. Dabei steht dem Anwender einer Textverarbeitungsanwendung oder eines Seitenlayout-Programms eine fast unüberschaubare Auswahl an Funktionen und Vorlagen zur Verfügung, die es ihm ermöglicht, ein Dokument in nahezu jeder beliebigen Form zu manipulieren. WYSIWYG-Anwendungen erscheinen hierbei recht komfortabel, da typographische Entscheidungen des Autors in Echtzeit angezeigt werden und jede Funktion mehr oder weniger unmittelbar nachzuvollziehen ist. Der Autor kann dabei entscheiden, wieviel seiner Phantasie in das Layout seines Dokuments einfließt und kann seine Vorstellungen jederzeit umsetzen oder auch wieder rückgängig machen. Dabei wird er jedoch während des Schreibprozesses von inhaltlichen und logischen Strukturierungen abgelenkt zu Gunsten von Entscheidungen, die seinen Text vermeintlich lesbarer oder ästhetisch wertvoller machen. Wie aber bereits dargestellt, ist gerade dies nicht Aufgabe des Verfassers, sondern der Typographen und Setzer, die auf Erfahrung eines Jahrhunderte alten künstlerischen Handwerks zurückblicken können.

Abgesehen von weiteren Nachteilen, die solche Anwendungen trotz des Anspruchs, ein professionell aussehendes(!) Dokument zu erzeugen (Verlust typographischer Qualität zu Gunsten von Schnelligkeit bei der Echtzeit-Verarbeitung, Probleme mit proprietären Dateiformaten und dergleichen) haben, bieten Satzsysteme wie TEX eine Alternative, die der traditionellen Arbeitsteilung gerecht wird und dem Autor in vielerlei Hinsicht wichtige typographische Entscheidungen abnimmt, um sie dann selbst auf Grund professioneller typographischer Kriterien umzusetzen. Somit ist – prinzipiell – der Autor einer TEX-Datei von diesen Entscheidungen entbunden, die oftmals im Gebrauch überflüssiger oder lediglich vermeintlich ästhetischer Textelemente resultieren.

Eine gewisse Vorsicht ist aber auch hier geboten, denn selbst *plain*-TEX verlangt von seinem Anwender grundlegende Entscheidungen über verschiedene Details ab. Die Weiterentwicklung zum Makropaket LATEX schließlich vereinfachte nun nicht nur den Gebrauch von TEX für den im Programmieren, sondern auch in typographischen Fragen unbewanderten, aber vielleicht sensibilisierten Anwender. Im Gegensatz zu WYSIWYG-Anwendungen, die dem Autor eine fast vollständige Freiheit in der Textgestaltung überlassen (die dementsprechend auch in WYSIWYD – „what you see is what you deserve“ – resultieren), wird die Textgestaltung im Falle von LATEX dem Autor aus der Hand genommen und ein hochwertiges Dokument produziert, bei dem Form und Inhalt einander entsprechen, wie es das Prinzip der typographischen Ar-

beitsteilung vorsieht. Aber dennoch, auch hier sind Helmut Kopka, Autor einer dreibändigen LaTeX-Einführung, „Beispiele bekannt, bei denen Schreibkräfte sich zu Recht darüber beklagen, dass sie nach der Einführung von LaTeX zwar mehr als 90% ihrer Schreibarbeiten nunmehr in rund 50% ihrer Arbeitszeit erledigen können, aber für weniger als 10% der Textarbeiten die andere Hälfte der Arbeitszeit benötigen, da einige wenige ihrer Auftraggeber sie stets mit immer neuen Layout-Forderungen nerven“ [6].

Wir wollen hier nicht den Eindruck erwecken, Typographen seien Hüter eines fast esoterischen Wissens und in ihrer Urteilskraft unfehlbar. Dass das Lesen, aber auch Veröffentlichen eines wissenschaftlichen Artikels oder einer Monographie auch für den Autor eine Leideserfahrung sein kann, wird (auf äußerst humorvolle Art und Weise) von dem Linguisten Geoffrey Pullum deutlich gemacht. Besonders hervorzuheben ist hier das Problem, einem Text mit Endnoten an Stelle von Fußnoten zu folgen – „You can't hold a volume open at two places simultaneously and still hold your cup of coffee, that's the main problem“ [9].

Somit müssen also weiterhin die oben beschriebenen typographischen Kriterien, solche der Lesbarkeit eines Textes und weitergehende professionelle Expertise bei der Produktion eines zu druckenden Dokuments in Betracht gezogen werden. In Abhängigkeit der Textart sind dann auch individuell unterschiedliche Entscheidungen zu treffen, damit sich Form und Inhalt des betreffenden Dokuments in einer möglichst optimalen Weise entsprechen. Die Frage aber, wie das Verdienst der Möglichkeiten der digitalen Typographie im professionellen als auch im DTP-Bereich zu bewerten ist, bleibt wohl weiterhin bestehen, denn

> [...] whether words are to be printed on paper or to be displayed on monitors, a high degree of skill will always be needed by those who have to decide on the typographical arrangement of words. The risk of making the wrong decision is all the greater when technology enlarges our range of choice; the risk is even higher when decisions are made by people without enough experience or judgement to act with wisdom and imagination [8].

Literatur

[1] Allin Cottrell: *Word Processors: Stupid and Inefficient*; `http://www.ecn.wfu.edu/~cottrell/wp.html`.

[2] Donald E. Knuth: *The METAFONTbook*; Addison-Wesley Publishing Company; Reading, Mass. u. a.; vierte Aufl.; 1991.

[3] Donald E. Knuth: *Computers and Typesetting*; in *Digital Typography* (Hg. Donald E. Knuth); S. 555–562; CLSI; Stanford, Cal.; 1999.

[4] Donald E. Knuth: *Digital Typography*; in *Digital Typography* (Hg. Donald E. Knuth); S. 1–18; CLSI; Stanford, Cal.; 1999.

[5] Donald E. Knuth: *The Concept of a Meta-Font*; in *Digital Typography* (Hg. Donald E. Knuth); S. 289–313; CLSI; Stanford, Cal.; 1999.

[6] Helmut Kopka: *LaTeX: Einführung*; Addison-Wesley (Deutschland) GmbH; Bonn; zweite Aufl.; 1996.

[7] Adobe Systems Inc.: *Adobe Systems in the Newspaper Industry*; Techn. Ber.; Adobe Systems Inc.; 1998.

[8] John Dreyfus: *Who Is to Design Books now that Computers Are Making Books?*; in *Into Print* (Hg. John Dreyfus); S. 283–297; The British Library; London; 1994.

[9] Geoffrey K. Pullum: *Stalking the Perfect Journal*; in *The Great Eskimo Vocabulary Hoax and Other Irreverent Essays on the Study of Language* (Hg. Geoffrey K. Pullum); S. 59–66; The University of Chicago Press; Chicago, Ill.; 1991.

[10] Matt Patterson: *Typography and Electronic Document Specification*; `http://reprocessed.org/writings/esays/dissertation`; 2001.

Optische Darstellungen mit pst-optic

Herbert Voß

Mit diesem Artikel soll die Beschreibung der Teilpakete, die alle unter dem Synonym pstricks zusammengefasst werden und mittlerweile mehr als nur latent undurchsichtig erscheinen, mit einem Paket zur Darstellung der optischen Verhältnisse an Linsen fortgesetzt werden. Derartige Abbildungen gehören zu den Standardthemen in Schule und Universität und können daher für einige Leser interessant sein.

Einführung

Es existieren mehrere gleichartige pstricks-Pakete, die sich zum Teil auch schon mal widersprechen. Hier obliegt es dem Anwender, sich einen entsprechenden Überblick zu verschaffen, was nicht immer einfach ist, denn die Dokumentationen sind teilweise ebenso fragmentiert wie die Pakete selbst. [8] Die TUG India startete vor einiger Zeit eine Initiative zur Überarbeitung der Dokumentation von pstricks, doch ist man bislang nicht über einen ersten Teil hinausgekommen. [2] Unter dem Arbeitstitel *References for TEX and Friends* [7] findet man einen Versuch, die vielfältigen pstricks-Makros mit ihren Optionen systematisch zu ordnen.

pstricks baut vollständig auf PostScript [3] auf und kann somit nicht direkt mit pdfLaTeX benutzt werden. Für eine PDF-Ausgabe kann auf das Paket pdf-tricks [6] oder auf das für Linux und OS/2 freie Programm VTeX (http://www.micropress-inc.com/) zurückgegriffen werden. Zusätzlich ist mit ps2pdf der übliche Weg über dvi→ps→pdf möglich.

Für die Darstellung optischer Linsenabbildungen existieren mehrere Einzelpakete bzw. TEX-Beispieldateien [5] und vor allen Dingen das Paket pst-optic. pst-lens stellt einen Sonderfall dar, denn es behandelt lediglich einen Lupeneffekt, auf den hier aber nicht eingegangen werden soll. [1] Ein Großteil der Beschreibungen ist entgegen den üblichen Gepflogenheiten in französischer Sprache abgefasst, die sicherlich nicht jedem geläufig ist. [4] Obwohl die dort veröffentlichten Beispiele vielfach für sich sprechen, soll im Folgenden eine kurze Einführung in das Paket pst-optic gegeben werden.

pst-optic ist eine Art Zusammenfassung von speziellen pstricks-Anwendungen, die in Form von Makros einfacher in Texte eingebunden werden

können. Dies impliziert, dass dieses Paket immer unvollständig bleiben wird. Dennoch lohnt eine Beschäftigung mit `pst-optic`, denn es weist einige Makros auf, die für alle von Interesse sind, die sich mit optischen Abbildungen beschäftigen. Und wie für fast alle speziellen `pstricks`-Pakete gilt auch für `pst-optic`, dass einelne Makros wiederum für andere Anwendungen interessant sein könnten, beispielsweise `\Parallel`. Das Paket `pst-optic` kann wie üblich von jedem CTAN-Server heruntergeladen werden (`CTAN:graphics/pstricks/contrib/pst-optic/`).

Zu beachten ist noch, dass zur Vermeidung von Fehlern ein eventuelles Laden von `graphicx` unbedingt vor `pst-optic` erfolgen sollte. `pst-optic` selbst lädt automatisch die Pakete `pst-node`, `pst-3d` und natürlich `pstricks`. Im Folgenden wird immer von Knoten gesprochen, wenn die sogenannten `Nodes` gemeint sind, auf die insbesondere `pst-node` Bezug nimmt.

Allgemeingültige Optionen

Grundsätzlich sind sämtliche Optionen dokumentweit gültig, werden jedoch nicht für alle Makros berücksichtigt. Tabelle 1 zeigt eine Zusammenstellung der für die meisten Makros gleichermaßen wichtigen Optionen, während weitere in den Tabellen 2 und 5 erklärt sind.

Linsenarten

Unterstützt werden selbstverständlich die Sammel- und die Zerstreuungslinse:

`\lens[CVG]` **C**on**v**er**g**ente (Sammellinse) - Standard

`\lens[DVG]` **D**i**v**er**g**ente (Zerstreuungslinse)

Für die weitere Betrachtung ist es hilfreich, wenn man sich erst einmal die Lage der vordefinierten Knoten vergegenwärtigt, was durch die Abbildung 1 unterstützt wird.

Ein Aufruf von `\lens[<lensType>]` liefert die in den Abbildungen 2 und 3 gezeigten Linsen mit den in Tabelle 2 angegebenen Standardwerten. Ohne weitere Angaben werden die drei in der Optik üblichen Strahlen eingezeichnet: Mittelpunkts-, Parallel- und Brennpunktstrahl.

Die Werte für `xLeft`, `xRight`, `yBottom` und `yTop` aus Tabelle 2 beziehen sich auf die übliche `pspicture`-Umgebung, wie sie von `pst-optic` intern benutzt wird:

Tabelle 1: Zusammenstellung der allgemeingültigen Optionen mit ihren Standardwerten

Option	*Name*	*Standard*
Linker Wert der Bildbreite in cm	xLeft	-7.5
Rechter Wert der Bildbreite in cm	xRight	7.5
Unterer Wert der Bildhöhe in cm	xBottom	-3
Oberer Wert der Bildhöhe in cm	xTop	3
x-Offset $\overline{\text{XO}}$	XO	0
y-Offset $\overline{\text{YO}}$	YO	0
Knoten A als String	nameA	A
Labelwinkel A in Grad	spotA	270
Knoten B als String	nameB	B
Labelwinkel B in Grad	spotB	270
Knoten F als String	nameF	F
Labelwinkel F in Grad	spotF	270
Knoten O als String	nameO	O
Labelwinkel O in Grad	spotO	225
Knoten A′ als String	nameAi	A′
Labelwinkel A′ in Grad	spotAi	90
Knoten B′ als String	nameBi	B′
Labelwinkel B′ in Grad	spotBi	270
Knoten F′ als String	nameFi	B′
Labelwinkel F′ in Grad	spotFi	270
Strahlfarbe	rayColor	black

```
\begin{pspicture*}(xLeft,yBottom)(xRight,yTop)
        ...
\begin{pspicture*}
```

Der Ursprung des optischen Koordinatensystems liegt automatisch in der Bildmitte, wenn auf die Definition einer eigenen `pspicture`-Umgebung mit anderen Koordinaten verzichtet wird (Abbildung 4). Dies ist in den meisten Fällen ausreichend, da sich viele geometrische Abbildungen auf eine optische Achse beziehen, die der x-Achse entspricht. Da die `lens`- und `telescope`-Makros in jedem Fall intern noch einmal die `pspicture`-Umgebung wie oben angegeben definieren, muss eine `\rput`-Anweisung vorangestellt werden:

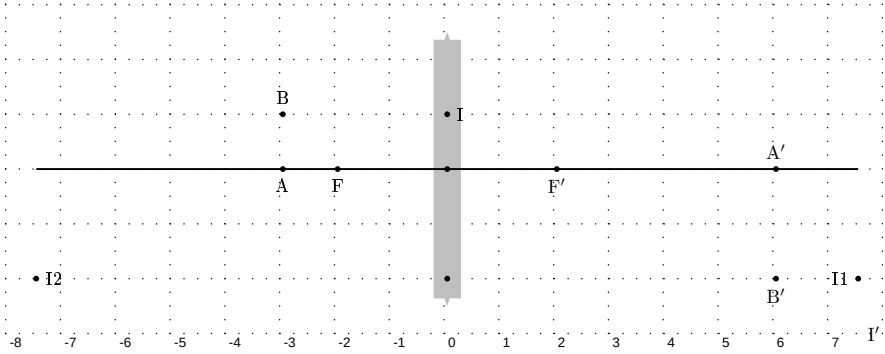


Abbildung 1: Die Lage der von `pst-optic` vordefinierten Knotenpunkte für Sammel- und Zerstreuungslinse mit Bezeichnungen

Tabelle 2: Zusammenstellung der Optionen für die Linsen mit ihren vordefinierten Standardwerten

Option	*Name*	*Standard*
Linsentyp (nur für `\lens`)	lensType	CVG
Linsenhöhe in cm	lensHeight	5cm
Linsendicke in cm	lensWidth	0.5cm[1]
vertikale Skalierung (obsolet)	lensScale	1
Darstellung	lensGlass	false
Mehrlinsensystem	lensTwo	false
Focus (Brennpunkt) in cm	focus	2
Abstand $\overline{OA}$ (Gegenstandsweite)	OA	-4
Abstand $\overline{AB}$ (Gegenstandshöhe)	AB	1.5
Linsenfarbe	lenscolor	black
Pfeilgröße in cm	lensarrowsize	0.2
Pfeilende in cm	lensarrowinset	0.5

[1] nur für `lensGlass=true`, ansonsten gleich `2\pslinewidth`

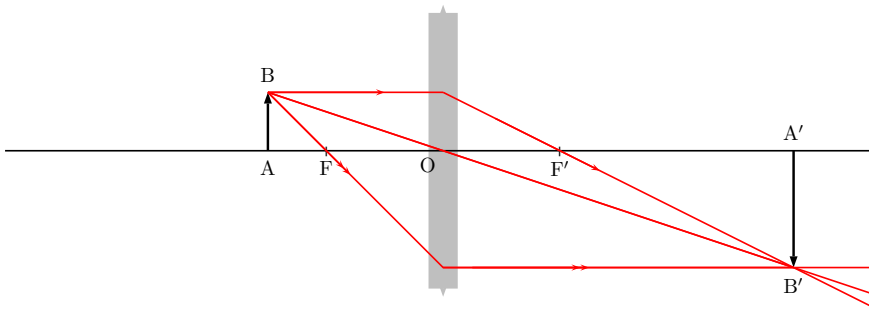


Abbildung 2: `\lens[lensType=CVG]` (Sammellinse)

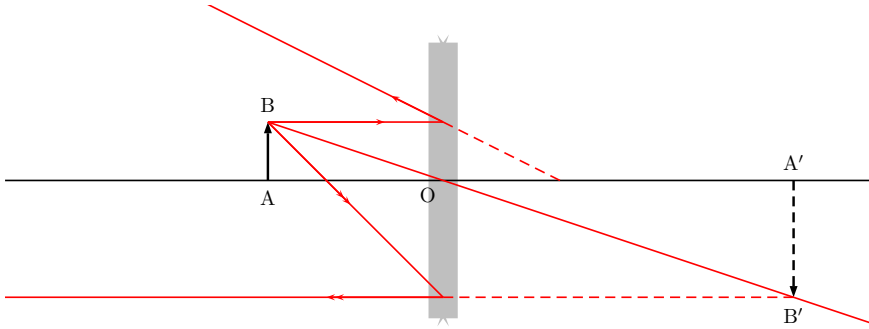


Abbildung 3: `\lens[lensType=DVG]` (Zerstreuungslinse)

```
\begin{pspicture*}(-7.5,-3)(7.5,3)
  \rput(0,0){\lens[...]}
\begin{pspicture*}
```

Dies ist notwendig, da eine geschachtelte `pspicture`-Umgebung den vorhergehenden Koordinatenursprung als untere linke Ecke benutzt, wohingegen mit `\rput` eine vertikale und horizontale Zentrierung erfolgt, womit beide Umgebungen kongruent sind, wenn sie die gleichen Abmessungen aufweisen.

Die Sternversion ist hier grundsätzlich zu empfehlen, da dadurch automatisch ein clipping auf die angegebenen Bildgrenzen erfolgt. Grundsätzlich kann auf die explizite Angabe der `pspicture`-Umgebung verzichtet werden, wenn nicht zusätzliche Makros ausgeführt werden.

Tabelle 3: Zusammenstellung der Optionen für das `Arrows`-Makro

Option	*Name*	*Standard*
Entfernung Pfeilanfang-Startknoten in cm	posStart	0
Länge des Pfeils in cm	length	2

Weitere Anweisungen

`\Arrows`

Hiermit kann die Art des Pfeils, der von einem Knoten zum anderen geht, beeinflusst werden. Die allgemeine Syntax ist

```
\Arrows[Optionen](Startknoten)(Endknoten)
```

Start- und Endknoten legen lediglich die Linie fest, auf denen der Pfeil verläuft, während Anfangs- und Endpunkt des Pfeils durch die in Tabelle 3 angegebenen Optionen festzulegen sind. Daneben sind die bei `pstricks` für Linien gültigen Optionen möglich.

Um beispielsweise den Gegenstandspfeil ($\overline{\mathrm{AB}}$) und den Bildpfeil ($\overline{\mathrm{A'B'}}$) deutlicher hervorzuheben, werden folgende zwei `\Arrows`-Makros angewendet:

```
\begin{pspicture*}(-7.5,-3)(7.5,3)
  \psgrid[subgriddiv=0,griddots=4,gridlabels=3]
  \rput(0,0){\lens[lensType=DVG,lensGlass=true,lensWidth=0.5]}
  \Arrows[length=1,linewidth=5pt](A)(B)%
  \Arrows[linewidth=5pt,linestyle=dashed](A')(B')%
\end{pspicture*}%
```

Der Abbildung 4 ist ein Gitter überlagert, so dass die einzelnen Koordinaten besser zu erkennen sind.

Zu beachten im Umgang mit `Arrows` sind folgende Punkte:

- Es gelten die `pstricks`-Standardoptionen [8] für Linien mit der Ausnahme, dass die Pfeilrichtung mit `{->}` festliegt und nicht ohne weiteres verändert werden kann.
- Die einzigen neu hinzugekommenen Optionen sind in Tabelle 3 erklärt.

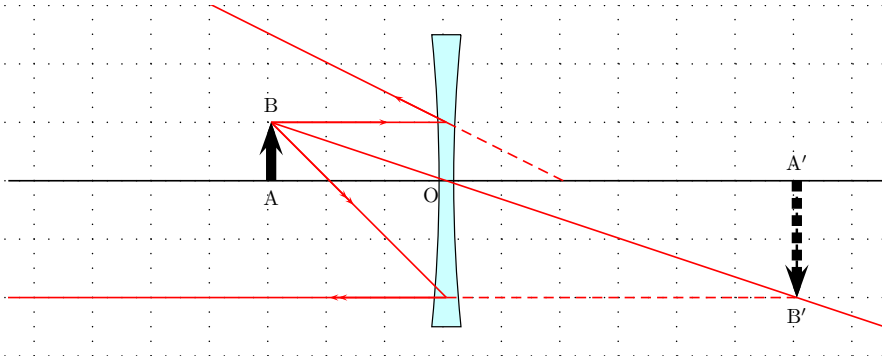


Abbildung 4: Anweisung `\lens[lensType=DVG,lensGlass=true]` und Aufruf eines `\Arrows` Macros

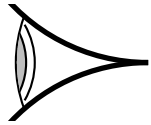


Abbildung 5: Anwendung des `\eye`-Makros

- Das Makro `\Arrow` muss *nach* dem `\lens` oder `\telescope` eingefügt werden, wenn man sich auf Knoten bezieht, die erst durch vorhergehende Makros definiert werden.
- `\Arrows`-Makros sollten in eventuelle Skalierungen mit einbezogen werden, da sonst der Maßstab falsch ist, denn `\Arrows` ist ein eigenständiges Makro.

`\eye`

Damit kann ein symbolisches Auge des Betrachters zur Verdeutlichung eingefügt werden (Abbildung 5).

`\psOutLine`

Mit diesem Makro können auf einfache Weise beliebige Linien verlängert werden, wozu dieses Makro drei Parameter benötigt:

`\psOutLine[Optionen](Startknoten)(Zwischenknoten){Endknoten}`

Neben der Option `length=...` (Vorgabe ist `length=1`) sind alle Optionen möglich, die auch für `\psline` definiert sind. Der Endknoten ist prinzipiell uninteressant, denn er wird intern in Abhängigkeit von den ersten beiden Knoten neu berechnet. Allerdings verlangt die Definition von `\psOutLine` die Angabe dieses dritten Parameters in geschweiften Klammern, so dass hier alles außer dem leeren Argument angegeben werden kann. Auf diesen Endknoten kann aber beim Zeichnen von weiteren Linien Bezug genommen werden.

Als Beispiel sollen einige Linien in Abbildung 4 verlängert werden. Eine Linie von F' nach B' kann einfach mit `\psline(F')(B')` erreicht werden, wohingegen eine Verlängerung dieser Linie bis an den Bildrand hinaus mit `\psOutLine(F')(B'){END0}` möglich ist, wobei `END0` der neu definierte Knotenpunkt ist. Somit ergibt sich die Abbildung 6, der die folgende `pspicture`-Umgebung zugrundeliegt:

```
\begin{pspicture*}(-7.5,-3)(7.5,3)
  \psgrid[subgriddiv=0,griddots=4,gridlabels=3]
  \rput(0,0){\lens[lensType=DVG,lensGlass=true]}
  \Arrows[posStart=0,length=1,linewidth=5pt](A)(B)
  {\psset{linestyle=dotted, linewidth=2pt}
    \psline(F')(B')
    \psOutLine(F')(B'){END0}
    \psOutLine[length=3](I')(B){END1}%
  }
\end{pspicture*}%
```

`\psBeforeLine`

Dieses Makro zeigt prinzipiell das gleiche Verhalten wie `psOutLine`, mit dem einzigen Unterschied, dass eine Linie der Länge `length` nicht an den Zwischenknoten sondern *vor* den Startknoten gezeichnet wird. Die allgemeine Syntax ist völlig identisch, was nicht unbedingt hilfreich ist, denn der in geschweifte Klammern zu setzende und vom Anwender festzulegende Startknoten sollte eigentlich von der Logik her als erstes in der Parameterliste

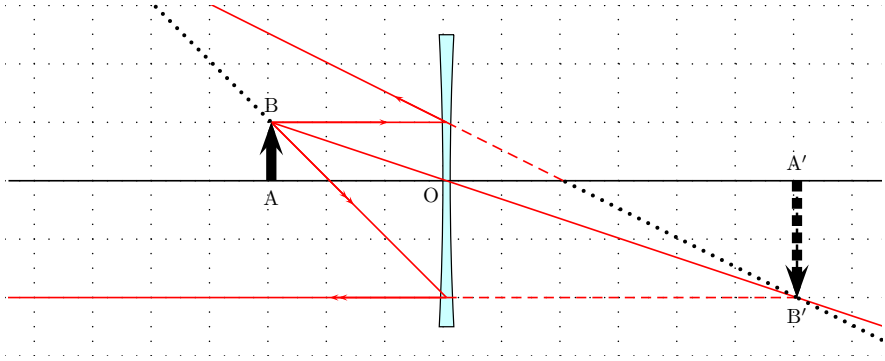


Abbildung 6: Anwendung von `\psOutLine` zum Zeichnen verlängerter gepunkteter Linien F'→B'→END0 sowie I'→B→END1

erscheinen. Dieser wird vom Makro durch die anderen beiden Punkte und die Linienlänge berechnet.

```
\psBeforeLine[Optionen](Zwischenknoten)(Endknoten){Startknoten}
```

Für die Optionen ist hier wieder im Wesentlichen `length=...` von Interesse.

`\Parallel`

Dies ist ein sehr nützliches Makro und vereinigt im Prinzip die Vorteile der vorhergehenden Linien-Makros. In der Optik treten sehr häufig parallele Strahlen auf, sodass es hier sehr hilfreich ist, wenn man die Richtung nur ein einziges Mal festlegt und alle anderen Linien parallel dazu zeichnet. Das Makro `\Parallel` erwartet vier Parameter mit den bereits oben hinreichend beschriebenen Optionen, wie beispielsweise `length=...`:

```
\Parallel[Optionen](KnotenA)(KnotenB)(Startknoten){Endknoten}
```

Die beiden Bezugsknoten `A` und `B` geben die Steigung vor und können als bereits bekannte Knoten definiert sein oder werden durch zwei Koordinaten festgelegt. Der Startknoten bezeichnet den Beginn der zu $\overline{AB}$ parallelen Linie, deren Länge über die Option `length` festzulegen ist und deren intern berechneter Endknoten mit einem beliebigen Namen versehen werden kann. Dieser

kann dann wieder wie ein normaler Knoten benutzt werden. Die folgende Zusammenstellung zeigt die LaTeX-Sequenz für Abbildung 7, wobei zur Vereinfachung das `multido`-Makro (`CTAN:/macros/generic/multido/`) verwendet wurde. Die Länge der einzelnen Parallelen ist willkürlich mit `length=9` festgelegt worden, denn durch die Sternversion der `pspicture`-Umgebung ist die clipping-Option aktiviert.

```
\begin{pspicture*}(-5,-3.5)(5,3.5)
  \psgrid[subgriddiv=0,griddots=5]
  \pnode(2,-2){FF}\qdisk(FF){1.5pt}
  \pnode(-5,5){A}
  \pnode(0,0){O}
  \multido{\nCountA=-2.4+0.4}{9}{%
    \Parallel[linecolor=red,length=9](O)(A)(0,\nCountA){P1}
    \psline[linecolor=red](0,\nCountA)(FF)
    \psOutLine[linecolor=red,length=9](0,\nCountA)(FF){P2}
    }
  \psline[linecolor=blue](A)(FF)
  \psOutLine[linecolor=blue,length=5](A)(FF){END1}
  \rput(0,0){%
    \lens[yBottom=-3.5,yTop=3.5,lensGlass=true,lensHeight=6.5,%
    drawing=false,spotFi=315,lensWidth=0.5]%
    \psline[linewidth=1pt](xLeft)(xRight)
    \psline[length=2,linewidth=2pt,arrows=->](F')(FF)
    }
\end{pspicture*}
```

`\Transform`

Das `Transform`-Makro übernimmt es, die Knoten der ersten optischen Abbildung in solche mit einem Index „1“ zu übertragen. Tabelle 4 zeigt eine Aufstellung der betroffenen Knoten. Weiterhin wird ein neuer Knoten `factice` mit den Koordinaten `(XO1,YO1)` definiert. Grundsätzlich wird diese Transformation nicht benötigt, sie erleichtert jedoch die Erstellung von teilweise komplexen Abbildungen mit mehreren Linsen. Werden zwei Linsen unabhängig voneinander gezeichnet, so geht `pst-optic` jeweils von einer Anordnung aus, wie sie beispielsweise in Abbildung 2 gezeigt ist. Die Strahlengänge sind jedoch zu verketten, was mit der Option `lensTwo=true` möglich ist. Abbildung 8 verdeutlicht die Bedeutung von `Transform` und `lensTwo`.

Erst durch die Verknüpfung von `\Transform` mit der `TwoLens`-Option erhält man das in Abbildung 8(c) dargestellte Ergebnis, welchem folgende LaTeX-

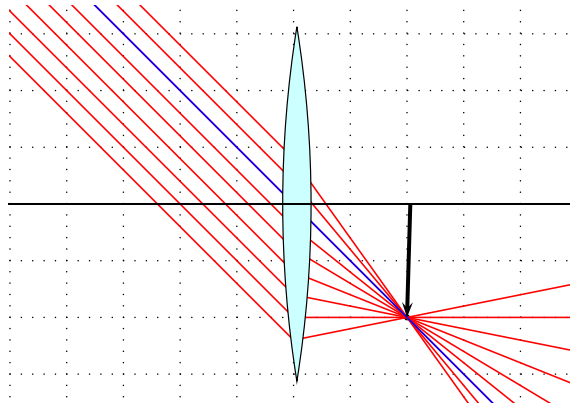


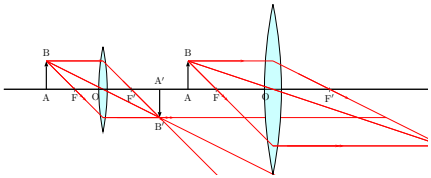
Abbildung 7: Anwendung des `\Parallel`-Makros

Tabelle 4: Transformation der wichtigsten Knoten nach dem Aufruf des Makros `\Transform`

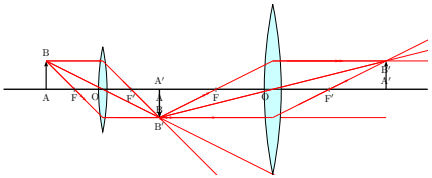
Alt	A	B	A'	B'	O	F	F'	I	I'	XO	YO	OA'	A'B'
Neu	A1	B1	A'1	B'1	O1	F1	F'1	I1	I'1	XO1	YO1	O1A1'	A'1B'1

Sequenz zugrundeliegt. Anhand des `polygon`-Makros lässt sich sehr gut die Bedeutung von `\Transform` erkennen, denn es werden die gleichen Parameter verwendet, die sich jedoch einmal auf die linke Linse und nach der Transformation der Knoten auf die rechte Linse beziehen:

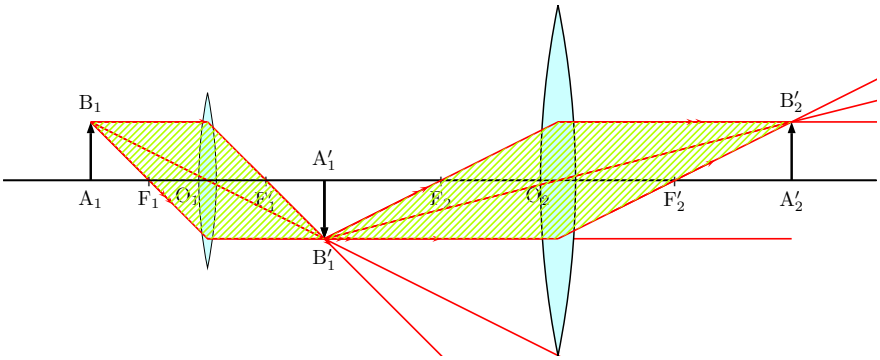
```
\begin{pspicture*}(-7.5,-3)(7.5,3)
\rput(0,0){%
    \lens[lensScale=0.6,XO=-4,%
          nameF=F_1,nameA=A_1,nameB=B_1,%
          nameFi=F'_1,nameAi={ },nameBi={},nameO=O_1,
          focus=1,OA=-2,lensGlass=true, lensWidth=0.5]%
}
\pspolygon[style=rayuresJaunes,linestyle=none](B)(I)(B')(I')(B)
\Transform
\rput(0,0){%
    \lens[lensScale=1.2,XO=2,focus=2,%
          nameA=A'_1,spotA=90,nameB=B'_1,spotB=270,%
```



(a) Definition zweier unabhängiger Linsen



(b) Definition zweier abhängiger Linsen mit `\lens[...]`, `\Transform`, `\lens[...]` und `lensTwo-Option`



(c) Definition zweier abhängiger Linsen mit zusätzlicher Modifikation der Knotenlabel sowie zusätzlicher Färbung des Strahlverlaufs

Abbildung 8: Die Bedeutung des `\Transform`-Makros mit standardmäßiger Beschriftung

```
        nameO=O_2,nameAi=A'_2,spotAi=270,%
        nameBi=B'_2,spotBi=90,nameF=F_2,nameFi=F'_2,%
        lensTwo=true,%
        lensGlass=true,lensWidth=0.5]%
}
\pspolygon[style=rayuresJaunes,linestyle=none](B)(I)(B')(I')(B)
\end{pspicture*}
```

`\rayInterLens`

Dieses Makro macht insbesondere bei mehreren Linsen Sinn, denn es ermöglicht, auf einfache Weise Knotenpunkte zu setzen, deren Lage nicht ohne

weiteres bekannt ist. Als Beispiel betrachten wir Abbildung 9, welche ein einfaches Zweilinsensystem darstellt. Die Knoten `B1, I11, F'1, B'1` liegen durch die Lage des Gegenstandes und der Brennweite der ersten Linse fest. Um nun die beiden von der linken Linse ausgehenden Strahlen über `B'1` bis zur zweiten Linse zu verlängern, müssten die entsprechenden Punkte in der Linsenmitte berechnet werden. Mithilfe des Makros `\rayInterLense` kann jedoch ein entsprechender Knoten gesetzt werden. Die Syntax ist

```
\rayInterLense(Start)(Zwischenpunkt)(Linsenabstand){Linsenknoten}
```

Für den in Abbildung 9 eingezeichneten Knoten `Inter1L2` ergibt sich damit

```
\rayInterLens(I11)(B'1){4}{Inter1L2}
```

Die Angabe des Linsenabstandes bezieht sich auf den horizontalen Abstand des Linsenursprungs vom Koordinatenursprung, der hier vier Längeneinheiten entspricht und über die Linsenoption festgelegt wurde (Tabelle 2). In Abbildung 9 ist zum Vergleich extra die y-Achse sowie ein Koordinatengitter angegeben. Aus dieser Angabe und aus der Lage des internen Knoten `B'1` berechnet das Makro die Lage des Endknotens, der hier den willkürlich gewählten Namen `Inter1L2` aufweist. Eine Linie kann dann mit dem normalen Linienmakro `\psline` gezeichnet werden:

```
\rayInterLens(I11)(B'1){4}{Inter1L2}
\psline(B1)(I11)(B'1)(Inter1L2)
\rayInterLens(O1)(B'1){4}{Inter2L2}
\psline(B1)(O1)(B'1)(Inter2L2)
```

Die beiden parallelen Linien wurden einfach mit dem `\Parallel`-Makro gezeichnet, denn sie sind ebenfalls parallel zur Geraden $\overline{\mathrm{B}'_1\mathrm{O}}$.

Spiegel

Abbildung 10 zeigt die möglichen Spiegelformen und Tabelle 5 die möglichen Optionen. Zum jetzigen Zeitpunkt können die beiden Parabolspiegel nur in aufrechter Form und mit dem aktuellen Koordinatenursprung als Bezugspunkt gezeichnet werden. Es stellt jedoch keine Schwierigkeit dar, sie mit dem `\rput`-Makro an beliebiger Stelle zu platzieren, wie es auch intern beim `telescop`-Makro geschieht. Allerdings machen Drehungen mit einem der Makros aus dem `rotating`-Paket wenig Sinn, da die interne Berechnung des Strahlverlaufs bislang vom senkrechten Spiegel ausgeht.

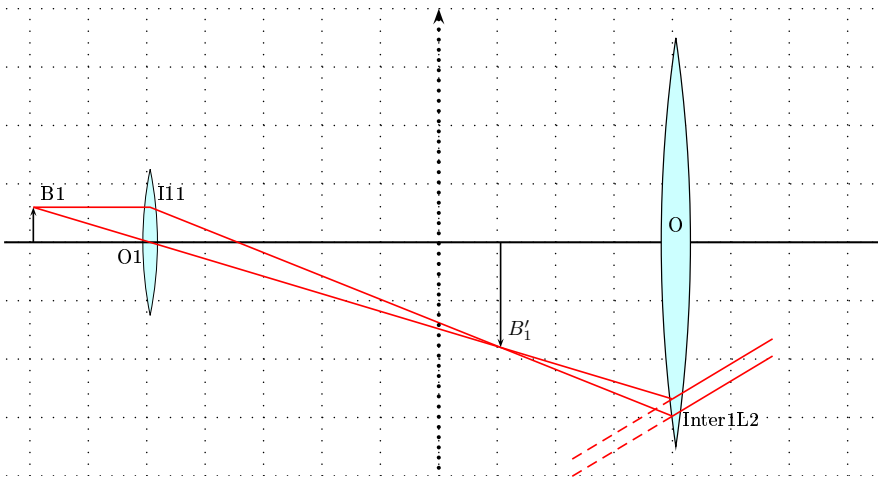


Abbildung 9: Demonstration des `\rayInterLens`

Tabelle 5: Zusammenstellung der Optionen für die Spiegel mit ihren vordefinierten Standardwerten

Option	*Name*	*Standard*
Linker Wert der Bildbreite in cm	xLeft	-0.5
Rechter Wert der Bildbreite in cm	xRight	11
Unterer Wert der Bildhöhe in cm	xBottom	-6
Oberer Wert der Bildhöhe in cm	yTop	2.5
Spiegelhöhe in cm	mirrorHeight	5
Spiegeltiefe in cm	mirrorDepth	1
Spiegeldicke in cm	mirrorWidth	0.25
Spiegelfarbe	mirrorColor	lightgray
Linienfarbe	rayColor	black
Brennpunkt in cm (nur in Verbindung mit der Option `posMirrorTwo` sinnvoll)	mirrorFocus	8
Position des zweiten Spiegel in cm	posMirrorTwo	8
Neigung des zweiten Spiegel in Grad	mirrorTwoAngle	45
Linienzüge zeichnen	drawing	true

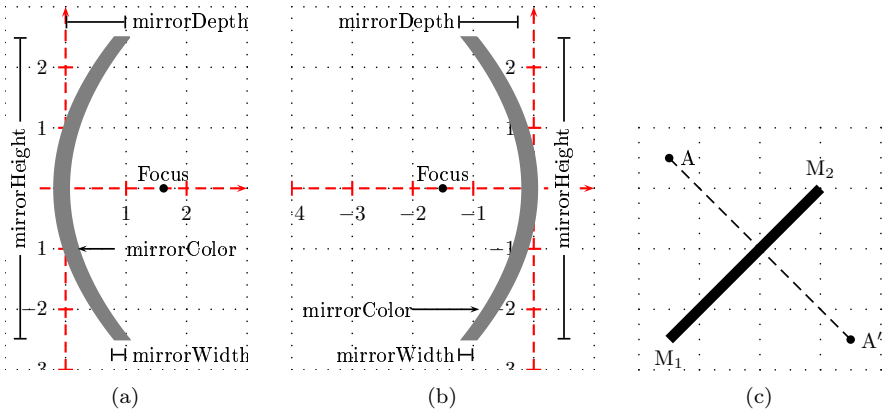


Abbildung 10: Zusammenstellung der verschiedenen Spiegel-Makros jeweils für die Standardwerte: a) `\mirrorCVG` b) `\mirrorDVG` c) `\planMirrorRay`

Parabolspiegel `\mirrorCVG`

Abbildung 11 zeigt den Standardspiegel mit den vordefinierten Knoten, die sich auf Mittelpunktsstrahl (M), Brennpunktstrahl (F) und Parallelstrahl (P) beziehen. Diese Abbildung ergibt sich durch Anwendung des Makros `mirrorCVG` ohne Optionen.

Parabolspiegel `\mirrorDVG`

Abbildung 13 zeigt den konvexen Parabolspiegel mit den vordefinierten Knoten, die sich wieder auf Mittelpunktsstrahl (M), Brennpunktstrahl (F) und Parallelstrahl (P) beziehen. Diese Abbildung ergibt sich durch Anwendung des `mirrorDVG`-Makros ohne Optionen.

Strahlverlauf im Parabolspiegel

Für beide Parabolspiegel kann jeder beliebige Strahlverlauf intern berechnet werden. Die Syntax der Makros ist

```
\mirrorCVGRay[Optionen](Knoten1)(Knoten2){Endknoten}
\mirrorDVGRay[Optionen](Knoten1)(Knoten2){Endknoten}
```

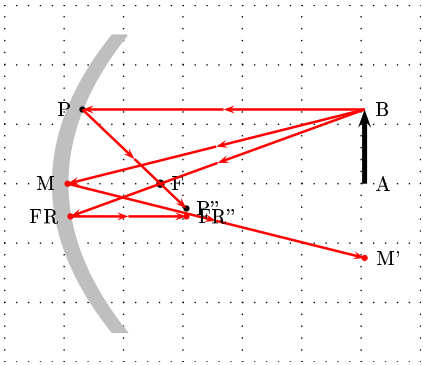


Abbildung 11: Konvergenter Parabolspiegel `\mirrorCVG`

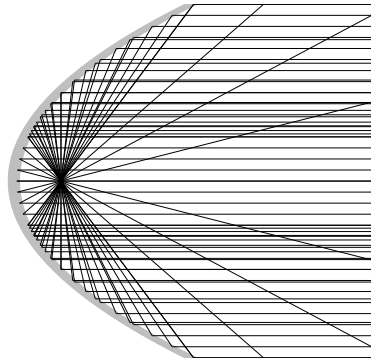


Abbildung 12: Anwendung des Makros `\mirrorCVG` als Scheinwerferdemo

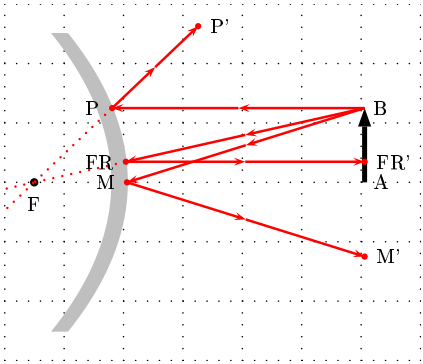


Abbildung 13: Divergenter Parabolspiegel `\mirrorDVG`

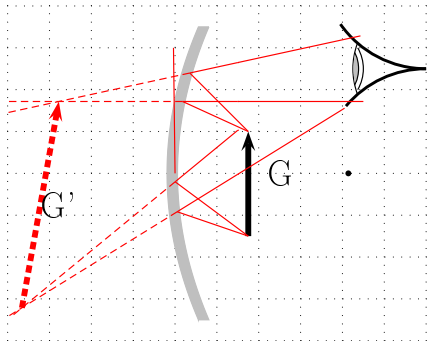


Abbildung 14: Anwendung `\mirrorDVG` als Vergrößerungsspiegel

Die Unterscheidung bezüglich der Anwendung des Parabolspiegels ist notwendig, da für den konvexen Spiegel eine Reflexion innerhalb des Spiegels stattfinden kann (Abbildung 14). Jede Richtungsänderung des Strahlverlaufs wird intern als Knoten festgelegt, so dass man nachträglich jederzeit die Abbildung beliebig erweitern kann. Die Reihenfolge der intern festgelegten Knoten ist:

Knotenname	erster Punkt auf dem Spiegel
Knotenname′	Endknoten oder zweiter Punkt auf dem Spiegel, falls eine zusätzliche Reflektion im Spiegel stattfindet
Knotenname″	Endknoten bei einer zusätzlichen Reflektion

Findet im Spiegel keine zusätzliche Reflektion statt, so sind der gestrichene und der doppelt gestrichene Knoten identisch. Für einen vom Anwender festgelegten Knotennamen „A“ werden dann intern noch zusätzlich „A'“ und „A''“ berechnet. `Knoten1` und `Knoten2` müssen lediglich auf dem Strahl liegen, wobei sie auch virtuelle Punkte auf einem verlängerten Strahl sein können, der durch den Spiegel geht. Mit der Option `drawing=false` werden nur die Knotenpunkte berechnet, mit denen dann weiter gearbeitet werden kann.

Planspiegel `\planMirrorRay`

Das `planMirrorRay`-Makro gestattet dagegen Spiegelpunkte automatisch bestimmen zu lassen. In Abbildung 10(c) wurde nur der Knoten `A` vorgeben, `A'` wird automatisch bestimmt und mit einem vom Anwender willkürlich festzulegenden Knotennamen belegt. Die Syntax ist:

```
\planMirrorRay(Spiegelbeginn)(Spiegelende)(Originalpunkt){Bildknoten}
```

Hiermit werden grundsätzlich keinerlei Linien gezeichnet, sondern lediglich der Bildknoten `A'` mit den richtigen Koordinaten belegt, der dann entsprechend benutzt werden kann.

Spiegeln zweidimensionaler Objekte mit `\symPlan`

Das Makro `\symPlan` gestattet es, komplette zweidimensionale Objekte entlang einer virtuellen Spiegelachse zu spiegeln. Dabei erfolgt dieser Vorgang, wie Abbildung 15 zeigt, im mathematischen und nicht physikalischen Sinne, denn es handelt sich formal um „durchlässige“ Spiegel. Weitere Beispiele kann man [4] entnehmen. Die Syntax ist

```
\symPlan(Knoten1)(Knoten2){Graphikobjekt}
```

Die beiden Knoten bezeichnen Spiegelanfang beziehungsweise Spiegelende. Als Graphikobjekt wird man in der Regel eine neue Anweisung definieren, der dann zweimal angewendet wird, beispielsweise:

```
\newcommand{\dtk}{%
    \pstextpath(0,0){%
```

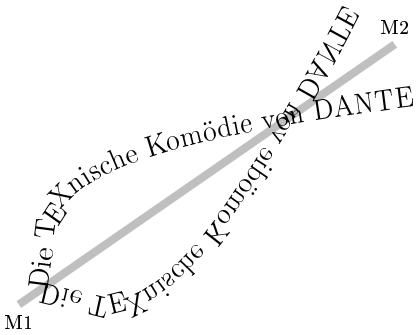


Abbildung 15: Demonstration des `\symPlan`-Makros

```
        \psplot[linestyle=none]{0}{8}{x sqrt sqrt 2 mul}}%
        {\Large Die \TeX{}nische Komödie von DANTE}%
}
\begin{pspicture}(-4.5,-2.5)(3,3.5)
    \pnode(-4,-2){M1}  \uput[-90](M1){M1}
    \pnode(2.5,2.5){M2}\uput[90](M2){M2}
    \psline[linewidth=5\pslinewidth,linecolor=lightgray](M1)(M2)
    \rput(-3.5,-1.75){\dtk}% Original schreiben
    \symPlan(M1)(M2){\rput(-3.5,-1.75){\dtk}}% Spiegelbild schreiben
\end{pspicture}
```

Zu beachten ist, dass `\pstextpath` das Paket `pst-text` benötigt, welches wie üblich auf jedem CTAN-Server zur Verfügung steht (`CTAN:/graphics/pstricks/generic/pst-text.tex`).

\telescope

Abbildung 16 zeigt die Grundkonfiguration für ein Teleskop. Tabelle 5 zeigt eine Zusammenstellung der speziellen Optionen für das `\telescop`-Makro, die sich vor allen Dingen auf die Lage des zweiten Spiegels beziehen.

\ABinterCD

Dieses Macro wird intern von `\telescop` benutzt. Es ermittelt den Schnittpunkt zweier Geraden, die jeweils durch zwei Punkte (Knoten) gegeben sind.

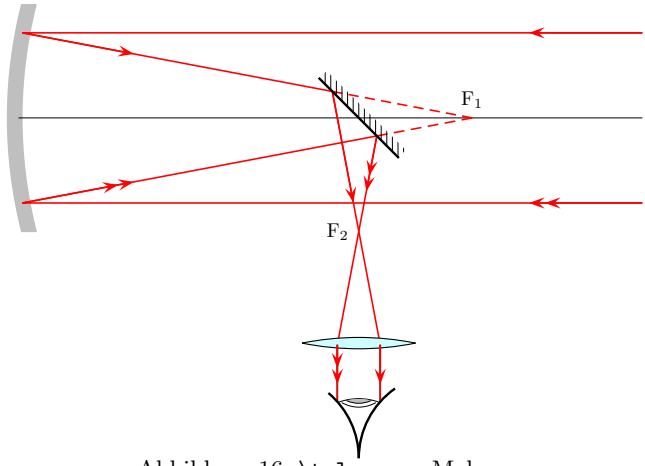


Abbildung 16: `\telescope`-Makro

Abbildung 17 zeigt prinzipiell einen Teil der Abbildung 16. Gegeben sind aufgrund des Strahlverlaufs die Punkte A, B (Brennpunkt), C/D (Planspiegel). Benötigt wird der Knoten E, um den weiteren Strahlverlauf zum Okular einzeichnen zu können. Dieser lässt sich durch

```
\ABinterCD(A)(B)(C)(D){E}
```

bestimmen und dann in der üblichen Weise verwenden.

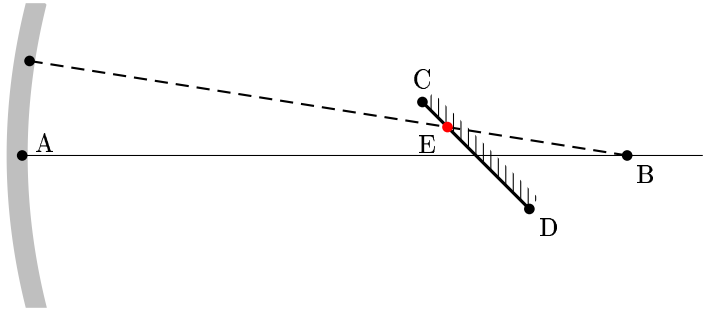


Abbildung 17: `\ABinterCD`-Makro

Literatur und Software

[1] Denis Girou und Manuel Luque: *PST-lens - PostScript macros for Generic TeX*; `ftp://ftp.dante.de/tex-archive/graphics/pstricks/contrib/pst-lens/`; 2001.

[2] The Indian TEX Users Group: *Online LATEX Tutorial; Part II - Graphics with PSTricks*; `http://www.tug.org.in/tutorial/pstricks/chap1.pdf`; 2002.

[3] Nikolai G. Kollock: *PostScript richtig eingesetzt: vom Konzept zum praktischen Einsatz*; IWT; Vaterstetten; 1989.

[4] Manuel Luque: *Lentilles convergentes: PST-optic v. 0.2*; `http://members.aol.com/ManuelLuque2/optique.htm`; 2001.

[5] Manuel Luque: *TEX Sources for Optical Systems*; `http://members.aol.com/ManuelLuque2/sources/`; 2002.

[6] Herbert Voss: *PSTricks Support for pdf*; `http://www.educat.hu-berlin.de/~voss/lyx/pdf/pdftricks.phtml`; 2002.

[7] Michael Wiedmann und Peter Karp: *References for TEX and Friends*; `http://www.miwie.org/tex-refs/`; 2003.

[8] Timothy Van Zandt: *PSTricks - PostScript macros for Generic TeX*; `http://www.tug.org/application/PSTricks`; 1993.

Anwendungen des LaTeX-Pakets preview

Rolf Niepraschk

Dieser Artikel beschreibt ein auf dem LaTeX-Paket preview basierendes Konzept zur Erzeugung von Grafikdateien, die in HTML-Dateien, mit pdfLaTeX oder anderen Verfahren verwendet werden können.

Einleitung

Im Zusammenhang mit dem Emacs-Zusatzpaket[1] preview-latex spielt das LaTeX-Paket preview eine besondere Rolle (siehe [6] und [7]). Mit seiner Hilfe gelingt es, die für die WYSIWYG-Funktion nötigen Grafikdateien in effektiver Weise zu erzeugen. Je nachdem, welche Bereiche des Dokuments im Editor realitätsnah dargestellt werden sollen, wird eine spezielle PostScript-Datei aus dem unveränderten Dokument generiert. Sie enthält genau die gewünschten Teile – jeweils ein Objekt pro Seite. Um welche Bestandteile des Dokuments es sich handelt, kann dem Paket preview über Paketoptionen mitgeteilt werden. In einem nachfolgenden Bearbeitungsschritt wird mit Hilfe von ghostscript eine Anzahl Pixelgrafiken erzeugt, die, wenn erforderlich, in das Editorfenster des Emacs „projiziert“ werden. Das Paket preview ist derart konzipiert, dass es auch unabhängig vom Emacs verwendet werden kann. Im Folgenden sollen nützliche Anwendungen, die von preview profitieren, vorgestellt werden.

Pixelgrafiken für HTML-Seiten

Innerhalb von HTML-Seiten ist es gelegentlich wünschenswert, auch mathematische Formeln darstellen zu können. Derzeit sind die verbreiteten Web-Browser nicht oder nur unzureichend in der Lage, die zu diesem Zweck erfundene Codierung – MathML genannt – darzustellen. Behelfen kann man sich, indem man die Formeln in Form von Grafiken in die HTML-Seite einfügt. preview kann die Erzeugung solcher Grafiken vereinfachen. Bei der Anwendung von preview spielen die hier kurz genannten Paketoptionen eine Rolle:

[1] Emacs ist einer der leistungsfähigsten Texteditoren, der auch beim Schreiben von TeX-Quelltexten extrem hilfreich ist.

active: Bewirkt, dass eine spezielle Art der Bearbeitung des Dokuments stattfindet. Die erzeugte DVI-Datei enthält nun pro Seite genau ein über Optionen auswählbares Objekt (Inhalt bestimmter Makros oder Umgebungen). Alle anderen Bestandteile des Dokuments werden ignoriert.

tightpage: Bewirkt, dass jede Seite genau die Größe des enthaltenen Objekts hat.

floats, displaymath, textmath, sections, graphics: Die Art der pro Seite auszugebenden Objekte. Zur Auswahl stehen Fließumgebungen, abgesetzte mathematische Formeln, Textformeln, Abschnittsüberschriften oder der Inhalt von `\includegraphics`-Anweisungen. Weiteres kann mit in preview enthaltenen Anweisungen festgelegt werden.

dvips: Benennt den zu verwendenden DVI-Treiber.

psfixbb: Zusätzlicher Code zur Unterstützung von dvips bei der Festlegung der Boundingbox wird eingefügt.

Nähere Einzelheiten zu diesen und weiteren Paketoptionen sind [7] zu entnehmen.

Das folgende Beispieldokument möge die gewünschten Formeln enthalten:

```
% formeln.tex
\documentclass{article}
\usepackage[active,displaymath,tightpage,dvips]{preview}
\begin{document}
  \begin{displaymath}
    c = \sqrt{a^2 + b^2}
  \end{displaymath}
  \begin{displaymath}
    \sigma(t) = \frac{1}{\sqrt{2\pi}} \int^t_0 e^{-x^2/2} dx
  \end{displaymath}
  Irgendwelcher Text ...
\end{document}
```

Das eingefügte und aktivierte Paket preview bewirkt, dass nur ausgewählte Teile des Dokuments in die erzeugte dvi-Datei gelangen; in diesem Fall aufgrund der Option `displaymath` nur abgesetzte mathematische Formeln. Nach Wandlung der Datei `formeln.dvi` ins PostScript-Format lassen sich mit ghostscript zwei Pixelgrafiken erzeugen. Im einfachsten Fall kann das so geschehen:

```
gs -sDEVICE=pngmono -r300 -dSAFER -q -dNOPAUSE -dBATCH \
   -sOutputFile=formel%d.png formeln.ps
```

Nachteilig dabei ist der Umstand, dass die Grafiken, wie das LaTeX-Dokument normalerweise auch, einen weißen Hintergrund haben. Das spielt nur dann keine Rolle, wenn die Grafiken in HTML-Seiten mit ebenfalls weißem Hintergrund eingefügt werden sollen. Im Folgenden werden daher die von ghostscript erzeugten Grafiken noch weiter bearbeitet, sodass ihr Hintergrund transparent wird (Option `-transparent` des Programms pnmtopng). Nähere Einzelheiten zu pnmtopng, einem Teil der Programmsammlung netpbm[2], können in [8] nachgelesen werden. Zusätzlich wird beim ghostscript-Aufruf das so genannte Antialiasing (`-dTextAlphaBits=4 -dGraphicsAlphaBits=4`) aktiviert, um störende Treppeneffekte zu mindern:

```
gs -sDEVICE=pgm -r300 -dTextAlphaBits=4 -dGraphicsAlphaBits=4 \
   -dSAFER -q -dNOPAUSE -dBATCH -sOutputFile=formel%d.png.tmp \
   formeln.ps
pnmtopng -transparent white formel1.png.tmp > formel1.png
pnmtopng -transparent white formel2.png.tmp > formel2.png
```

Programme zur Wandlung von kompletten LaTeX-Dokumenten ins HTML-Format, wie beispielsweise LaTeX2HTML oder HyperLaTeX, gehen ähnlich vor, allerdings ohne dass sie intern die mit preview möglichen Vorteile nutzen. Hauptsächlich sei dabei die Möglichkeit hervorgehoben, mit einem einzigen LaTeX-Lauf alle gewünschten Grafiken zu erzeugen.

PSTricks- und psfrag-Grafiken

Die LaTeX-Pakete PSTricks und psfrag sind Beispiele dafür, wie mit Unterstützung von PostScript sehr interessante grafische Effekte erzielt werden können (siehe [3], [4] und [5]). Erforderlich ist ein PostScript-Interpreter, der die aus der dvi-Datei erzeugte PostScript-Datei nachbearbeiten muss. Der PostScript-Interpreter ist entweder in einem realen Drucker enthalten oder kann seine Arbeit in Form eines „virtuellen“ PostScript-Druckers (ghostscript) verrichten. Dieser Weg ist jedoch dann nicht gangbar, wenn man sein Dokument mit pdfLaTeX bearbeiten will. Eine Lösung könnte folgendermaßen aussehen: Man lagert die betreffenden Grafiken in ein Hilfs-LaTeX-Dokument aus, bearbeitet dieses mit LaTeX, dvips und ghostscript (in Form des Programms epstopdf) und fügt die so erhaltenen PDF-Grafiken ins Hauptdokument ein. Der Umstand, eine Hilfsdatei verwenden zu müssen, kann durchaus als Vorteil

[2] netpbm ist auch Bestandteil der Windows-TeX-Distributionen fpTeX und MikTeX.

angesehen werden. Man hat damit alle Grafiken an einem Ort konzentriert und das Hauptdokument bleibt übersichtlich. Auch die Bearbeitungszeit für die Grafiken fällt nur im Falle von Änderungen der Grafiken an, dann allerdings werden sämtliche Grafiken neu erzeugt. Und nicht zuletzt verfügt man am Ende des Bearbeitungsprozesses über einzeln vorliegende Grafikdateien, die auch anderweitig verwendbar sind. Die normalerweise umständliche und manchmal fehlerträchtige Behandlung eines solchen Hilfsdokuments lässt sich dank preview stark verbessern, wie im Folgenden zu sehen ist.

Die Hauptdatei könnte – sehr spartanisch – folgendermaßen aussehen:

```
% haupt.tex
\documentclass{article}
\usepackage{graphicx}
\begin{document}
  \includegraphics{bild1} \par \includegraphics{bild2}
\end{document}
```

Die Hilfsdatei muss also zur Erzeugung von zwei Grafiken dienen. Der Einfachheit halber sind hier zum Test Grafiken aus dem PSTricks-Kapitel des „graphics companion“ verwendet worden (siehe [4] und [9]):

```
% bilder.tex
\documentclass{article}
\usepackage[active,floats,tightpage,dvips]{preview}[2002/11/05]
\usepackage{pstcol,pst-node,pst-tree,multido}
\definecolor{lightblue}{cmyk}{0.65,0.13,0,0}
\begin{document}
  \begin{figure} \input{4-6-41.inl} \end{figure}
  \begin{figure} \input{4-10-8.inl} \end{figure}
\end{document}
```

Der Code zur Erzeugung der Grafiken wurde hier in `figure`-Umgebungen gekapselt, um preview das seitenweise Aufteilen – durch die Paketoption `floats` in Gang gesetzt – zu ermöglichen. Nach dem Übersetzen der Hauptdatei und Wandlung ins PostScript-Format[3] sind die nachfolgend beschriebenen Programmaufrufe durchzuführen, um die benötigten Grafikdateien `bild1.pdf` und `bild2.pdf` zu erhalten. Der im Vergleich zum vorigen Abschnitt aufwändige Ablauf erklärt sich mit einigen diffizilen Unzulänglichkeiten von ghostscript und dvips, die so umgangen werden können.

[3] Da das PDF-Format Ziel der Bemühungen ist, sollte darauf geachtet werden, dass die erzeugte PostScript-Datei die Schriften im Type1-Format enthält, was im Fall von dvips durch den Aufruf »`dvips -Ppdf -G0 -o datei.ps datei.dvi`« erreicht werden kann.

1. `ps2pdf bilder.ps bilder.pdf`

   Die enthaltenen PostScript-Anweisungen werden während der Wandlung ins PDF-Format ausgeführt (speziell auch die für PSTricks sowie für preview nötigen). Die resultierende Datei `bilder.pdf` enthält ein Bild pro Seite, wobei die Seitengröße mit der Bildgröße identisch ist (ein Umstand, der zusätzlichem von preview eingefügten PostScript-Code zu verdanken ist). Das Programm ps2pdf ist Bestandteil von ghostscript.

Bereits die so erzeugte Datei `bilder.pdf` enthält sämtliche Grafiken in einer verwendbaren Form. Zum Darstellen der zweiten Grafik wäre beispielsweise `\includegraphics[page=2]{bilder.pdf}` anzugeben. Wenn man aber Wert auf Einzeldateien legt, sind noch die folgenden Schritte nötig.

2. `pdftops -f 1 -l 1 -eps bilder.pdf bild1.eps`

   Die erste Seite bzw. die erste Grafik wird in Form einer eps-Datei extrahiert (Auswahl durch `-f` *`Startseite`* `-l` *`Endseite`*). Sie ist nur ein Zwischenprodukt, kann aber natürlich auch von Nutzen sein. Das Programm pdftops ist Bestandteil der Programmsammlung xpdf (siehe [10]).

3. `epstopdf --outfile=bild1.pdf bild1.eps`

   Im letzten Schritt wird die gewünschte PDF-Grafik erzeugt. Das Programm epstopdf ist in den meisten modernen TeX-Distributionen enthalten und setzt die Existenz von ghostscript voraus, wobei darauf geachtet werden sollte, dass dessen Programmversion 6.0 oder höher ist.

Die Punkte 2. und 3. müssen angepasst für alle weiteren Grafiken wiederholt werden. Aus diesem Grund ist es von großem Vorteil, die nötigen Programmaufrufe in einem `Makefile` oder einer Script-Datei zusammenzufassen, wodurch man einen automatisierten Ablauf erreichen kann.

Schlussbemerkung

Ziel des Artikels war es, Anwendungen des LaTeX-Pakets preview sowie von anderen Hilfsprogrammen zu zeigen, um Anregungen für eigene Experimente zu geben. Um das Prinzip zu verdeutlichen, ist auf mögliche Optimierungen verzichtet worden. Mit dem Gezeigten sind die Möglichkeiten von preview lange noch nicht ausgeschöpft. Insbesondere kann es die Grundlage bilden für künftige LaTeX-Pakete, die die Grafikeinbindung noch mehr vereinfachen.

Literatur

[1] *Die Beispieldateien zu diesem Artikel*; `http://www.dante.de/dante/dtk02_4/preview/`.

[2] *GNU + Cygnus + Windows = cygwin*; `http://www.cygwin.com/`.

[3] Denis Girou: *PSTricks (by Timothy van Zandt)*; `http://www.tug.org/applications/PSTricks/`.

[4] Michel Goossens, Sebastian Rahtz und Frank Mittelbach: *The LaTeX Graphics Companion*; Addison-Wesley Longman, Reading. Mass.; 1997.

[5] Michael C. Grant und David Carlisle: *The PSFrag system, version 3*; Nov. 1996; `ftp://dante.ctan.org/tex-archive/macros/contrib/supported/psfrag/pfgguide.tex`.

[6] David Kastrup: *LaTeX und WYSIWYG? preview-latex unter Emacs und andere Ansätze*; *Die TeXnische Komödie*; 4/2002, S. 10–26; Dez. 2002.

[7] David Kastrup: *The preview Package for LaTeX*; Nov. 2002; `http://sourceforge.net/projects/preview-latex/` und CTAN: `tex-archive/support/preview-latex/`.

[8] *Netpbm home page*; `http://netpbm.sourceforge.net/`.

[9] Sebastian Rahtz: *Test file for the PSTricks 97 distribution*; CTAN: `tex-archive/graphics/pstricks/doc/test-pst.tex`.

[10] *Xpdf*; `http://www.foolabs.com/xpdf/` und CTAN: `tex-archive/support/xpdf/`.

Berichtigung zu „Tipps und Tricks: Tierkunde einmal anders“

Rolf Niepraschk

Ich bin von einem aufmerksamen Leser auf einen Fehler in meinem Beitrag in Heft 3/2002, S. 31–33 bei der Umdefinition der description-Umgebung

hingewiesen worden. In seinem Beispiel wurde die angestrebte einheitliche Einrückung nur bei jedem zweiten LaTeX-Lauf erreicht, mit anderen Worten: Der Ablauf erreichte keinen stabilen Zustand. Eine Nachfrage beim Autor des eqlist-Pakets brachte die eigentlich naheliegende Erkenntnis, dass die dem Makro `\eqlistauto` übergebene Länge nicht `.33\linewidth` sondern `.33\columnwidth` hätte sein müssen. Der Wert von `\columnwidth` ist fest, `\linewidth` dagegen hängt auch von der sich ergebenden Geometrie der description-Umgebung ab, wodurch sich dieses schwingende Verhalten ergab. Hier die korrekte Definition:

```
\renewenvironment{description}
{\begin{eqlist}
[%
  \eqlistauto{.33\columnwidth}%
  \def\makelabel##1{\hspace{-\labelsep}\descriptionlabel{##1}}
]}
{\end{eqlist}}
```

Spielplan

Termine

2.–4. 4. 2003 DANTE 2003 und 28. Mitgliederversammlung von DANTE e.V.
Universität Bremen, Bremen
Kontakt: DANTE e.V.

1.–3. 5. 2003 BachoTEX 2003
11th Annual Meeting of the Polish TeX Users' Group
GUST Bachotek, Brodnica Lake District, Polen
Kontakt: GUST, Jolanta Szelatynska

24.–27. 6. 2003 EuroTEX 2003
14^{th} Annual Meeting of the European TEX Users' Group
Bretagne, Frankreich
Kontakt: Secrétariat de la conférence EuroTEX 2003

20.–26. 7. 2003 TUG 2003 Waikaloa Beach Resort, Big Island, Hawaii
`http://www.tug.org/tug2003/`
Kontakt: Wendy McKay

8.–9. 9. 2003 29. Mitgliederversammlung von DANTE e.V.
Schloss Rauischholzhausen (bei Marburg),
Universität Gießen, Gießen
Kontakt: DANTE e.V.

Stammtische

In verschiedenen Städten im Einzugsbereich von DANTE e.V. finden regelmäßig Treffen von TEX-Anwendern statt, die für jeden offen sind. Im WWW gibt es aktuelle Informationen unter `http://www.dante.de/events/stammtische/`.

Berlin
Rolf Niepraschk
Tel.: 0 30/3 48 13 16
`niepraschk@ptb.de`
Gaststätte „Bärenschenke“
Friedrichstraße 124
Zweiter Donnerstag im Monat, 19.00 Uhr

Bremen
Martin Schröder
Tel.: 04 21/2 23 94 25
`martin@oneiros.de`
Wechselnder Ort
Erster Donnerstag im Monat, 18.30 Uhr

Chemnitz
Ralf König
Tel.: 03 41/4 11 58 00
`ralf.koenig@s1998.tu-chemnitz.de`
Universitätsteil 1, Straße der Nationen 62, Raum 1/068
Dritter Mittwoch im Monat, 18.00 Uhr

Darmstadt
Karlheinz Geyer
`karlheinz.geyer@LHSystems.com`
Gaststätte „Kartoffelkiste“
Darmstadt, Ortsteil Bessungen
Weinbergstraße 10
Erster Freitag im Monat, ab 19.30 Uhr

Dortmund
Stephan Lehmke
`Stephan.Lehmke@cs.uni-dortmund.de`
Café Durchblick
Universität Dortmund, Campus Nord
Zweiter Mittwoch im Monat, 20.00 Uhr

Dresden
Carsten Vogel
`lego@wh10.tu-dresden.de`
Studentenwohnheim, Borsbergstraße 34, Dresden/Striesen
ca. alle 8 Wochen, Donnerstag, 19.00 Uhr

Erlangen
Walter Schmidt, Peter Seitz
`was@VR-Web.de`,
Gaststätte „Erlanger Gärtla“
Marquardsenstraße 1
Dritter Dienstag im Monat, 19.00 Uhr

Freiburg
Heiko Oberdiek
Tel.: 07 61/4 34 05
`oberdiek@uni-freiburg.de`
Wechselnder Ort
Dritter Donnerstag im Monat, 19.30 Uhr

Hamburg
Volker Hüttenrauch
`volker_huettenrauch@hh.maus.de`
Vereinsheim der Hamburger Microcomputer-Hochschulgruppe
Grindelallee 143 (Hinterhof)
Letzter Donnerstag im Monat, 18.00 Uhr

Hannover
Mark Heisterkamp
`heisterkamp@rrzn.uni-hannover.de`
Seminarraum RRZN
Schloßwender Straße 5
Zweiter Mittwoch von geraden Monaten, 18.30 Uhr

Heidelberg
Luzia Dietsche
Tel.: 0 62 21/54 45 27
`luzia.dietsche@urz.uni-heidelberg.de`
China-Restaurant „Palast“
Lessingstraße 36
Letzter Mittwoch im Monat, 20.00 Uhr

Karlsruhe
Klaus Braune
Tel.: 07 21/6 08 40 31
`braune@rz.uni-karlsruhe.de`
Universität Karlsruhe, Rechenzentrum
Zirkel 2, 3. OG, Raum 316

Erster Donnerstag im Monat, 19.30 Uhr

Köln
Bruno Hopp
b.hopp@lepkes-frings.de
Institut für Kristallographie
Zülpicher Straße 49b
Letzter Mittwoch im Monat, 19.30 Uhr

Konstanz
Matthias Weisgerber, Hraban Ramm
weisgerb@fmi.uni-konstanz.de,
hraban@fiee.net
Restaurant Rheingold
Spanierstraße 3
unregelmäßig

München
Michael Niedermair
m.g.n@gmx.de
Gastwirtschaft „Rhætenhaus“
Luisenstraße 27
Erster Dienstag im Monat, 19.00 Uhr

Münster
Johannes Reese
reese@linguist.de
Gaststätte „Sabroso“
Mauritzstraße 19
Erster Montag im Monat, 20.00 Uhr

Stuttgart
Bernd Raichle
bernd.raichle@gmx.de
Gaststätte „Alte Mira“
Büchsenstraße 24
Zweiter Dienstag im Monat, 19.30 Uhr

Wuppertal
Andreas Schrell
Tel.: 02 02/50 63 81
schrell@wupperonline.de
Restaurant Croatia „Haus Johannisberg“
Südstraße 10
an der Schwimmoper Wuppertal-Elberfeld
Zweiter Donnerstag im Monat, 19.30 Uhr

Adressen

DANTE, Deutschsprachige Anwendervereinigung TEX e.V.
Postfach 10 18 40
69008 Heidelberg

Tel.: 0 62 21/2 97 66 (Mo, Mi–Fr, 10^{00}–12^{00} Uhr)
Fax: 0 62 21/16 79 06
E-Mail: dante@dante.de

Konten: Volksbank Rhein-Neckar eG
BLZ 670 900 00
Kontonummer 2 310 007

Postbank Karlsruhe (Auslandsüberweisungen)
BLZ 660 100 75
Kontonummer 213 400 757

Präsidium

Präsident:	Volker RW Schaa	president@dante.de
Vizepräsident:	Klaus Höppner	vice-president@dante.de
Schatzmeister:	Tobias Sterzl	treasurer@dante.de
Schriftführer:	Günter Partosch	secretary@dante.de
Beisitzer:	Thomas Koch	
	Bernd Raichle	advisor@dante.de

Server

ftp: ftp.dante.de [134.100.9.51]
E-Mail: ftpmail@dante.de
WWW: http://www.dante.de/

Autoren/Organisatoren

Tim Doll [14]
uzsdz5@uni-bonn.de

Secrétariat de la conférence EuroTEX 2003 [67]
Départment Informatique
ENST Bretagne
France
Yannis.Haralambous@enst-bretagne.fr

Christian Faulhammer [9]
Jakobstr. 136
52064 Aachen
v-li@gmx.de

Klaus Höppner [5]
siehe Seite 70

Wendy McKay [67]
wgm@cds.caltech.edu

Gerd Neugebauer [3]
Im Lerchelsböhl 5
64521 Groß-Gerau
gene@gerd-neugebauer.de

Rolf Niepraschk [60, 65]
Persiusstr. 12
10245 Berlin
niepraschk@ptb.de

Volker RW Schaa [5, 7]
siehe Seite 70

Herbert Voß [40]
Wasgenstr. 21
14129 Berlin
voss@perce.de

Roland Weibezahn [7]
Universität Bremen
IWD
Postfach 33 04 40
28334 Bremen

Die TEXnische Komödie

15. Jahrgang Heft 1/2003 Februar 2003