Kostenloses eBook

LERNEN postscript

Free unaffiliated eBook created from
Stack Overflow contributors.

#postscript

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: postscript

It is an unofficial and free postscript ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official postscript.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Postscript

Bemerkungen

PostScript ist eine Reverse-Polish-Stack-basierte, dynamisch typisierte, dynamische Namensraum-Skriptsprache mit integrierten Grundelementen zum Generieren gerenderter Bilder aus Vektorbeschreibungen. PostScript verwendet dasselbe "Adobe Image Model" wie das PDF-Dateiformat.

PostScript wird von vielen Programmen als Ausgabeformat verwendet, da es einfach maschinell erstellt werden kann.

Wie LISP ist PostScript *homoikonisch,* und Code und Daten haben dieselbe Darstellung. Prozeduren können Prozeduren als Daten und Ergebnisprozeduren als Ergebnisse verwenden, was sich auch für Techniken der *konkatenativen Programmierung* eignet.

Examples

Installation oder Setup

Die authentischen Adobe PostScript-Interpreter sind in High-End-Druckern, dem Produkt Display PostScript (DPS) und dem Produkt Acrobat Distiller verfügbar. Als Hersteller des Standards werden diese Produkte als "Standardimplementierung" betrachtet, um Unterschiede zwischen PostScript-Implementierungen zu beschreiben.

Die im PLRM definierte Standardschnittstelle für den Interpreter ist der *Programmstrom,* der je nach den Details des zugrunde liegenden Kanals oder des Betriebssystems / Controllers entweder Text oder binär sein kann. Acrobat Distiller verfügt über ein GUI-Frontend, um das Eingabe-Postscript-Programm auszuwählen und seine Ausgabe als PDF-Datei darzustellen. Distiller bietet auch eine begrenzte Unterstützung für die Verwendung des Ausgabetextstroms zum Melden von Fehlern und anderen Programmausgaben. GSView bietet ein ähnliches GUI-Frontend für einen ähnlichen Workflow mit Ghostscript als Interpreter.

Ghostscript und Xpost arbeiten beide in einem Befehlszeilenmodus. Die Postscript-Programmdatei, die ausgeführt werden soll, kann in der Befehlszeile (`gs program.ps xpost program.ps` oder `xpost program.ps`) angegeben werden, die ein Grafikfenster öffnet, in dem die grafische Ausgabe angezeigt wird. Optionen können verwendet werden, um die Grafiken an anderer Stelle wie eine Festplattendatei darzustellen oder die Grafiken vollständig zu unterdrücken und Postscript nur als Skriptsprache zu verwenden.

Die verschiedenen Dolmetscher haben jeweils eigene Installations- und Setupanweisungen, und es wäre verschwenderisch (und dazu neigen, nicht mehr aktuell zu sein), sie hier zu reproduzieren.

Frei verfügbare PostScript-Interpreter

- Ghostscript ist für alle gängigen Plattformen und Linux-Distributionen in Quell- oder Binärform, unter der GNU-Lizenz oder unter anderen Lizenzvereinbarungen mit den Autoren, Artifex-Software , verfügbar . Ghostscript implementiert den vollständigen PostScript 3-Standard.

- Xpost ist in Quellform für alle gängigen Plattformen unter der BSD-3-Klausel-Lizenz verfügbar. Es implementiert den Level-1-Standard mit einigen Level-2-Erweiterungen und einigen DPS-Erweiterungen.

Allgemeine Beschreibung von PostScript

PostScript ist eine allgemeine Programmiersprache für Turing, die von Adobe Systems entwickelt und entwickelt wurde. Viele der Ideen, die in PostScript aufblühten, waren in Projekten für Xerox und Evans & Sutherland entwickelt worden.

Seine Hauptanwendung in der realen Welt ist historisch eine *Seitenbeschreibungssprache* oder in ihrem einseitigen EPS eine Vektorgrafik-Bildbeschreibungssprache. Es ist dynamisch typisiert, dynamisch und stapelbasiert, was zu einer meist polnischen umgekehrten Syntax führt.

Es gibt drei Hauptversionen von PostScript.

1. **PostScript Level 1** - wurde 1984 als residentes Betriebssystem des LaserWriter-Laserdruckers von Apple auf den Markt gebracht und eröffnet damit die Ära des Desktop Publishing.
2. **PostScript Level 2** - veröffentlicht im Jahr 1991, enthielt mehrere wichtige Verbesserungen von Level 1, einschließlich Unterstützung für die Dekomprimierung von Bildern, In-RIP-Trennung, automatisch wachsende Wörterbücher, Speicherbereinigung, benannte Ressourcen und binäre Kodierungen des PostScript-Programmstroms.
3. **PostScript 3** - die neueste und vielleicht am weitesten verbreitete Version wurde 1997 veröffentlicht. Sie enthält auch einige Verbesserungen des Imports gegenüber Level 2, wie beispielsweise Smooth Shading. Der Begriff „Ebene“ wurde gestrichen.

Obwohl PostScript normalerweise als Seitenbeschreibungssprache verwendet wird und daher in vielen Druckern zum Erzeugen von Rasterbildern implementiert ist, kann es auch für andere Zwecke verwendet werden. Als schneller Reverse-Polish-Rechner mit mehr einprägsamen Namen als `bc` . Als Ausgabeformat, das von einem anderen Programm generiert wurde (normalerweise in einer anderen Sprache).

Zwar handelt es sich bei PostScript-Dateien normalerweise um 7-Bit-Bereinigungs-ASCII-Dateien, jedoch gibt es verschiedene Arten von Binärkodierungen, die im Standard der Ebene 2 beschrieben werden. Und weil es programmierbar ist, kann ein Programm sein eigenes willkürlich komplexes Codierschema für sich selbst implementieren. Es gibt einen internationalen verschleierten Postscript-Wettbewerb, etwas weniger aktiv als der C-Wettbewerb.

Online-Referenzen

- Indexseiten der Adobe-Dokumentation: https://www.adobe.com/products/postscript/resources.html

http://www.adobe.com/devnet/postscript.html
http://www.adobe.com/devnet/font.html

- PostScript-Referenzhandbuch, 3ed - Der PostScript 3-Standard. (7.41MB pdf)
(Ergänzung , Errata)
- PostScript-Referenzhandbuch, 2ed - Der PostScript Level 2-Standard. (Enthält die PostScript-Dokumentation.) (3.29MB pdf)
- Postscript Tutorial und Kochbuch - Das blaue Buch. (847KB pdf)
- Postscript Language Program Design - Das grüne Buch. (911KB pdf)
- Denken in Postscript - Vom Autor des grünen Buches und des Tutorials des blauen Buches. (826KB pdf)
- Strukturierungskonventionen für PostScript-Dokumente Spezifikation 3.0 (521KB pdf)
- Adobe Type 1-Schriftformat (444 KB, pdf)
- Encapsulated PostScript File Format-Spezifikation 3.0 (185KB pdf)
- Postscript Printer Description File Format Specification 4.3 (186KB pdf) (Aktualisierung)
- Fehlerbehebung bei PostScript-Fehlern - Tipps zur Fehlersuche . (158 KB HTML)
- Acumen Journal - Archiv von Postscript- und PDF-Programmierartikeln. (html-Verzeichnis der gezippten pdfs)
- Mathematische Illustrationen: Ein Handbuch der Geometrie und Postscript - von Bill Casselman. (HTML-Verzeichnis der PDF-Kapitel und Code-Downloads)
- Thread mit vielen Sortieralgorithmus-Implementierungen (Usenet-Archiv)
- Don Lancasters Guru-Seiten
- Anastigmatix's Direkter Gebrauch der Postscript-Sprache
- Schrittweiser Open-Source- Debugger für Postscript-Code

FAQs

- Usenet-FAQ (circa 1995)
- Wikibooks PostScript-FAQ
- SO PostScript-Fragen sortiert nach den am häufigsten angezeigten

Bücher

- Postscript-Referenzhandbuch für Sprachen, 1ed, 1985. Empfohlen aufgrund der geringen

Größe und des einfachen Bedienerindex auf der Zusammenfassungsseite (fehlt in späteren Ausgaben).

- Postscript der realen Welt. Kapitel verschiedener Autoren zu verschiedenen Themen, einschließlich exzellenter Berichterstattung über Halbtoneffekte.

Lokale Namespaces für Funktionen

Postscript ist eine Sprache für dynamischen Namespace oder *LISP 1* . Es bietet jedoch die Werkzeuge zum Implementieren lokaler Variablen in Prozeduren und andere Effekte, die zur Implementierung von Algorithmen erforderlich sind.

Erstellen Sie für lokale Namen in einer Prozedur am Anfang ein neues Wörterbuch, und platzieren Sie es am Ende.

```
/myproc {
    10 dict begin
    %... useful code ...
    end
 } def
```

Sie können dies auch gut mit einer Verknüpfung kombinieren, um die Argumente der Funktion als Variablen zu definieren.

```
% a b c  myproc  result
/myproc {
    10 dict begin
    {/c /b /a} {exch def} forall
    %... useful code yielding result ...
    end
 } def
```

Wenn Sie eine * "globale" * - Variable aktualisieren müssen, während das lokale Wörterbuch oben ist, verwenden Sie `store` anstelle von `def` .

Hallo Weltbeispiel

Wählen Sie eine Schrift und Schriftgröße aus, wählen Sie den Ort aus, `show` String an.

```
%!PS
/Palatino-Roman 20 selectfont
300 400 moveto
(Hello, World!) show
showpage
```

Anmerkungen und häufige Fallstricke:

- Festlegen einer Schriftart (führt zu keinem Text oder einer (hässlichen) Standardschriftart)
- Verwenden Sie `findfont` und `setfont` , vergessen `setfont` jedoch nicht, zwischen den `setfont` zu `scalefont` . Durch die `selectfont` des Level-2- `selectfont` dieses Problem vermieden und

es wird `selectfont` .

- Einen Punkt nicht mit `moveto` oder den Punkt außerhalb der Seite setzen. Für US-Briefpapier ist 8.5x11 792x612 ps Punkte. Es ist also leicht, sich an 800x600 zu erinnern (aber ein bisschen kürzer und breiter). So ist `300 400` ungefähr die Mitte der Seite (wenig hoch, wenig links).

- Vergessen, die `showpage` . Wenn Sie ein ps-Programm mit `gs` in der `showpage` anzeigen und nicht mit der `showpage` , zeigt `gs` möglicherweise ein Bild an. Und dennoch wird die Datei auf mysteriöse Weise keine Ausgabe erzeugen, wenn Sie versuchen, in PDF oder etwas anderes zu konvertieren.

Lehrplan

Lesen Sie die Dokumentation in dieser Reihenfolge, um das Postscript leicht zu erlernen:

1. Paul Bourkes hervorragendes Tutorial: http://paulbourke.net/dataformats/postscript/

2. Blue Book, erste Hälfte, das ursprüngliche offizielle Tutorial: http://www-cdf.fnal.gov/offline/PostScript/BLUEBOOK.PDF

3. Green Book, wie man Postscript effektiv einsetzt: http://www-cdf.fnal.gov/offline/PostScript/GREENBK.PDF

4. Denken in Postscript ', sagte Nuff: http://wwwcdf.pd.infn.it/localdoc/tips.pdf

5. Mathematische Illustrationen . Fange klein an, baue groß. Die Mathematik hinter Bezier Curves. Der Hodgman-Sutherland-Polygon-Ausschnittsalgorithmus. Affine Transformationen und *nichtlineare* Transformationen des Pfads. 3D-Zeichnung und Gouraud-Schattierung. Aus dem Vorwort:

   Welche der vielen Werkzeuge, mit denen man mathematische Grafiken erstellen kann, zu wählen ist, scheint offensichtlich ein Kompromiss zwischen Einfachheit und Qualität zu sein, wobei die meisten sich für das halten, was sie als Einfachheit empfinden. Die Wahrheit ist, dass ein Kompromiss nicht notwendig ist - wenn man erst einmal eine kleine Anfangsanstrengung getätigt hat, ist es bei weitem das Beste, ein Programm in der Grafikprogrammiersprache PostScript zu schreiben. Die Qualität der Ausgabe eines PostScript-Programms ist praktisch unbegrenzt, und mit zunehmender Erfahrung nehmen die Schwierigkeiten bei der Verwendung der Sprache rapide ab. Die scheinbare Komplexität, die mit der Erstellung einfacher Zahlen durch das Programmieren in PostScript verbunden ist, wie ich hoffe, dass dieses Buch zeigen wird, ist weitgehend eine Illusion. Und der Aufwand für die Herstellung komplizierterer Figuren ist in der Regel weder höher noch geringer als notwendig.

Erste Schritte mit Postscript online lesen: https://riptutorial.com/de/postscript/topic/5616/erste-schritte-mit-postscript

Kapitel 2: Fehlerbehandlung

Syntax

- { *-code-* } gestoppt { *-error-* } { *-no-error-* } ifelse% Fehler beim *Abrufen des* Frames
- $ error / errorname get% stackunderflow typecheck rangecheck etc
  $ error / newerror% bool erhalten. setze false um den Fehler zu deaktivieren
  $ error / ostack get% Kopie des Operanden-Stacks zum Zeitpunkt des Fehlers
- errordict / stackoverflow { *-additional-code-* / stackoverflow signalerror} put
  % führt zusätzlichen Code für Fehlerarten aus (hier den / stackoverflow-Fehler).

Bemerkungen

Die Fehlerbehandlung in Postscript umfasst zwei Ebenen. Diese Zweiteilung bezieht sich sowohl auf die Art und Weise, wie der Interpreter mit Fehlern umgeht, als auch auf die Mittel, die dem Benutzer (Programmierer) zur Verfügung stehen, um die Handhabung zu steuern.

Die untere Ebene ist eine ungewöhnliche Steuerungsstruktur, `stop ... stopped` . `stopped` , ähnlich wie bei einem Looping in verhält sich konstruieren , dass sie eine Markierung auf dem Ausführungsstapel legt , die gesprungen werden, wenn der `exit` Operator (eine Schleife) oder `stop` - Operator (nach einem `stopped` -Kontext) aufgerufen wird. Im Gegensatz zu einem Schleifenkonstrukt, `stopped` ergibt eine Boolsche auf dem Stapel anzeigt , ob `stop` (andernfalls die Prozedur aufgerufen wurde weitergegeben `stopped` ist bekannt, zur Vollständigkeit ausgeführt worden.

Wenn ein Postscript - Fehler auftritt, wie `stackunderflow` vielleicht sucht der Interpreter der Fehler Namen in bis `errordict` , die lebt `systemdict` . Wenn der Benutzer die Prozedur nicht in `errordict` , erstellt die Standardfehlerprozedur Momentaufnahmen des gesamten Stapels und platziert sie in `$error` , einem anderen Wörterbuch in `systemdict` . Schließlich ruft die Standardprozedur `stop` wodurch das Benutzerprogramm aus dem Exec-Stack gezogen wird und die Fehlerdruckprozedur des Interpreters, die als `handleerror` `errordict` .

Wenn Sie all dieses Wissen verwenden, können Sie Fehler *abfangen* , indem Sie einen Codeabschnitt in `{ ... } stopped` . Sie können durch den Aufruf eines Fehler *erneut auslösen* `stop` . Sie können feststellen, welche Art von Fehler mit `$error /errorname get` .

Sie können das Standardverhalten für einen bestimmten Fehlertyp auch ändern, indem Sie die Prozedur durch diesen Namen in `errordict` . Oder ändern Sie das Format für das Drucken des Fehlerberichts, indem Sie `/handleerror` im `errordict` Format `errordict` .

Examples

Gibt es einen aktuellen Punkt?

Ausbeute `true` auf dem Stapel , wenn `currentpoint` erfolgreich ausgeführt wird , oder `false` , wenn es einen signalisiert `/nocurrentpoint` Fehler.

```
{currentpoint pop pop} stopped not  % bool
```

Folge von Ereignissen, wenn ein Fehler gemeldet wird

Die Reihenfolge für einen Fehler lautet normalerweise:

1. Fehler wird ausgelöst, indem der `errordict` in `errordict` und diese Prozedur ausgeführt wird.
2. Die `errordict` ruft den `signalerror` und übergibt ihm den `signalerror` .
3. `signalerror` Momentaufnahmen der Stapel, speichert die Momentaufnahmen in `$error` und ruft dann die Aufrufe `stop` .
4. `stop` öffnet den Exec-Stack, bis der nächstliegende umschließende gestoppte Kontext vom gestoppten Operator festgelegt wird.
5. Wenn das Programm keinen eigenen angehaltenen Kontext zum Abfangen des Fehlers erstellt hat, wird es von einem äußeren `stopped { errordict /handleerror get exec } if` der vom `stopped { errordict /handleerror get exec } if` das gesamte Benutzerprogramm `stopped { errordict /handleerror get exec } if` .
6. `handleerror` verwendet die Informationen in `$error` , um einen Fehlerbericht zu drucken.

Fehler melden

Die meisten Werkzeuge sind standardisiert, mit Ausnahme des Namens des Bedieners, der einen Fehler auslöst. In Adobe-Interpreten wird dies als `.error` . In Ghostscript wird es `signalerror` . Mit dieser Zeile können Sie den `signalerror` in Postscript-Code für Adobe-Interpreter oder Ghostscript oder xpost verwenden.

```
/.error where {pop /signalerror /.error load def} if
```

Befehlsname Fehlername **Signalfehler** -
Machen Sie Momentaufnahmen des Stapels in `$error` und `stop` dann.

Z.B.

```
% my proc only accepts integer
/proc {
    dup type /integertype ne {
        /proc cvx /typecheck signalerror
    } if
    % ... rest of proc ...
} def
```

Fehler abfangen

Da die letzte Aktion des Standardfehlerhandlers der Aufruf von `stop` , können Sie Fehler von Operatoren abfangen, indem Sie Code in ein `{ ... } stopped stop` Konstrukt einschließen.

```
{
    0 array
    1 get
} stopped {
    $error /errorname get =
} if
```

gibt " `rangecheck` " aus, der Fehler wird durch `get rangecheck` , wenn der Index außerhalb des zulässigen Bereichs für dieses Array liegt.

Fehler erneut werfen

Dieses Snippet implementiert eine Prozedur, die sich wie ein PostScript-Schleifenoperator verhält. Wenn der Benutzer `proc` das `exit` aufruft, wird der Fehler `invalidexit` abgefangen, um den Dictstack am `end` zu beheben. Alle anderen Fehler außer `invalidexit` durch Aufruf von `stop` erneut ausgelöst.

```
% array n proc  .  -
% Like `forall` but delivers length=n subsequences produced by getinterval
/fortuple { 4 dict begin
    0 {offset proc n arr} {exch def} forall
    /arr load length n idiv
    {
        {
            /arr load offset n getinterval
            [ /proc load currentdict end /begin cvx ] cvx exec
            /offset offset n add def
        } stopped {
            $error /errorname get /invalidexit eq
            { 1 dict begin exit }{ stop } ifelse
        } if
    } repeat
end
} def

%[ 0 1 10 {} for ] 3 {} fortuple pstack clear ()=
```

Fehlerbehandlung online lesen: https://riptutorial.com/de/postscript/topic/6199/fehlerbehandlung

Kapitel 3: Pfadaufbau

Examples

Ein Polygon zeichnen (beschreiben)

In diesem Beispiel wird versucht, das Verhalten der eingebauten Operator für die Pfadkonstruktion wie `arc` nachzuahmen.

Wenn es einen aktuellen Punkt gibt, zieht `poly` zuerst eine Linie zu (x, y) + (r, 0), ansonsten beginnt es mit dem Punkt.

Anstelle von `gsave` ... `grestore` (was den unerwünschten Effekt hat, dass alle Änderungen des aktuellen Pfads, die wir möchten, `grestore`), speichert es eine Kopie der aktuellen Transformationsmatrix (CTM), die beim Start der Funktion vorhanden ist.

Dann ist es `lineto` zu jedem nachfolgenden Punkt, der durch Skalieren und Drehen der Matrix ist stets bei (0,1). Schließlich ruft es `closepath` und stellt die gespeicherte Matrix als CTM wieder her.

```
% x y n radius  poly  -
% construct a path of a closed n-polygon
/poly {
    matrix currentmatrix 5 1 roll  % matrix x y n radius
    4 2 roll translate             % matrix n radius
    dup scale                      % matrix n
    360 1 index div exch           % matrix 360/n n
    0 1 {lineto currentpoint moveto}stopped{moveto}if   % start or re-start subpath
    {                              % matrix 360/n
        dup rotate                 % matrix 360/n
        0 1 lineto                 % matrix 360/n
    } repeat                % matrix 360/n
    pop                     % matrix
    closepath               % matrix
    setmatrix               %
} def
```

Durch einen Pfad iterieren

Dieses Snippet gibt den Inhalt des aktuellen Pfads in stdout aus. Es verwendet die Ghostscript-Prozedur `=only` die möglicherweise nicht für alle Interpreter verfügbar ist. Ein äquivalentes Verfahren für Adobe-Interpreter lautet `=print` .

`pathforall` ist ein Schleifenoperator, der 4 Prozedurgremien als Argumente verwendet, die für die spezifischen Arten von `moveto` , das Ergebnis von `moveto` , `lineto` , `curveto` , `closepath` und allen anderen `closepath` , die sich auf diese Elemente `curveto` , `closepath` werden.

```
{ exch =only ( ) print =only ( ) print /moveto =}
{ exch =only ( ) print =only ( ) print /lineto =}
{ 6 -2 roll exch =only ( ) print =only ( ) print
  4 2 roll exch =only ( ) print =only ( ) print
```

```
  exch =only ( ) print =only ( ) print /curveto =}
{ /closepath = }
pathforall
```

Millimeterpapier

```
/in {72 mul} def
/delta {1 in 10 div} def
/X 612 def
/Y 792 def
0 delta Y {
    0 1 index X exch  % i 0 X i
    moveto exch       % 0 i
    lineto
    stroke
} for
0 delta X {
    0 1 index Y  % i 0 i Y
    moveto       % i 0
    lineto
    stroke
} for
showpage
```

Pfadaufbau online lesen: https://riptutorial.com/de/postscript/topic/6679/pfadaufbau

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Postscript	Community, Kurt Pfeifle, luser droog
2	Fehlerbehandlung	luser droog
3	Pfadaufbau	luser droog