EBook Gratis

APRENDIZAJE postscript

Free unaffiliated eBook created from
Stack Overflow contributors.

#postscript

Tabla de contenido

Acerca de

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.

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

Capítulo 1: Empezando con postscript

Observaciones

PostScript es un lenguaje de scripting basado en el dinamismo, de tipos dinámicos, basado en apilamiento inverso, con primitivas incorporadas para generar imágenes renderizadas a partir de descripciones de vectores. PostScript emplea el mismo "Modelo de imagen de Adobe" que el formato de archivo PDF.

PostScript se utiliza como formato de salida en muchos programas, ya que está diseñado para ser generado fácilmente por una máquina.

Al igual que LISP, PostScript es *homoicónico* y el código y los datos comparten la misma representación. Los procedimientos pueden tomar los procedimientos como datos y los procedimientos de rendimientos como resultados, prestándose también a técnicas de *programación concatenativa* .

Examples

Instalación o configuración

Los auténticos intérpretes de Adobe PostScript están disponibles en impresoras de alta gama, el producto Display PostScript (DPS) y el producto Acrobat Distiller. Como autores del estándar, estos productos se consideran "la implementación estándar" con el fin de describir las diferencias entre las implementaciones de PostScript.

La interfaz estándar para el intérprete definido en el PLRM es el *flujo de programa* que puede ser de texto o binario, dependiendo de los detalles del canal subyacente o sistema operativo / controlador. Acrobat Distiller tiene una interfaz gráfica de usuario para seleccionar el programa PostScript de entrada y representar su salida en formato PDF. Distiller también tiene un soporte limitado para usar el flujo de texto de salida para informar errores y otros resultados del programa. GSView proporciona una interfaz gráfica de usuario similar para un flujo de trabajo similar utilizando Ghostscript como intérprete.

Ghostscript y Xpost funcionan en modo de línea de comandos. El archivo de programa postscript para ejecutar puede mencionarse en la línea de comandos (`gs program.ps` o `xpost program.ps`) que abrirá una ventana gráfica para mostrar la salida gráfica. Las opciones se pueden usar para renderizar los gráficos en otro lugar como un archivo de disco o suprimir los gráficos por completo y usar postscript como un lenguaje de scripting de texto.

Los distintos intérpretes tienen sus propias instrucciones de instalación y configuración y sería inútil (y propenso a no estar actualizado) reproducirlos aquí.

Intérpretes de PostScript de libre disponibilidad.

- Ghostscript está disponible para todas las plataformas principales y distribuciones de Linux, en formato fuente o binario, bajo la licencia GNU o bajo otros acuerdos de licencia con los autores, el software Artifex . Ghostscript implementa el estándar completo de PostScript 3.

- Xpost está disponible en formato fuente para todas las plataformas principales, bajo la licencia BSD-3-cláusula. Implementa el estándar de Nivel 1 con algunas extensiones de Nivel 2 y algunas extensiones DPS.

Descripción general de PostScript

PostScript es un lenguaje de programación general completo de Turing, diseñado y desarrollado por Adobe Systems. Muchas de las ideas que florecieron en PostScript se habían cultivado en proyectos para Xerox y Evans & Sutherland.

Su principal aplicación en el mundo real históricamente es como un *lenguaje de descripción de página* , o en su EPS de una sola página forma un lenguaje de descripción de imágenes de gráficos vectoriales. Es de tipo dinámico, dinámico y basado en pila, lo que conduce a una sintaxis principalmente polaca inversa.

Hay tres versiones principales de PostScript.

1. **PostScript Nivel 1** : se lanzó al mercado en 1984 como el sistema operativo residente de la impresora láser LaserWriter de Apple, que inauguró la era de la publicación digital.
2. **El nivel 2 de PostScript** : lanzado en 1991, contenía varias mejoras importantes al nivel 1, incluido el soporte para descompresión de imágenes, separación en RIP, diccionarios de crecimiento automático, recolección de basura, recursos con nombre, codificaciones binarias de la secuencia del programa PostScript.
3. **PostScript 3** : la última versión, y quizás la más ampliamente adoptada, se lanzó en 1997. También contiene varias mejoras de importación sobre el Nivel 2, como Smooth Shading. El término "nivel" se ha caído.

Aunque PostScript se usa normalmente como un lenguaje de descripción de página y, por lo tanto, se implementa dentro de muchas impresoras para generar imágenes de mapa de bits, también se puede usar para otros fines. Como una calculadora de pulido inverso rápido con nombres de operador más memorables que `bc` . Como un formato de salida generado por otro programa (generalmente en algún otro idioma).

Aunque los archivos PostScript son típicamente ASCII de 7 bits limpios, existen varios tipos de codificación binaria descritos en el estándar de nivel 2. Y al ser programable, un programa puede implementar su propio esquema de codificación arbitrariamente complejo por sí mismo. Hay un Concurso Internacional de Postscript ofuscado, algo menos activo que el de C.

Referencias en linea

- Páginas de índice de la documentación de Adobe:
  https://www.adobe.com/products/postscript/resources.html
  http://www.adobe.com/devnet/postscript.html
  http://www.adobe.com/devnet/font.html

- Manual de referencia del lenguaje PostScript, 3ed - El estándar PostScript 3. (7.41MB pdf) (Suplemento , Errata)
- Manual de referencia del lenguaje PostScript, 2ed - El estándar PostScript Nivel 2. (Incluye documentación de Display PostScript). (3.29MB pdf)
- PostScript tutorial y libro de cocina - El libro azul. (847KB pdf)
- Diseño del Programa de Lenguaje Postscript - El libro verde. (911KB pdf)
- Pensando en PostScript : por el autor del libro verde y el tutorial del libro azul. (826KB pdf)
- Especificación de convenciones de estructuración de documentos en lenguaje PostScript 3.0 (521KB pdf)
- Formato de fuente Adobe Type 1 (444KB pdf)
- Especificación de formato de archivo PostScript encapsulado 3.0 (185KB pdf)
- Descripción de la impresora PostScript Formato de archivo Especificación 4.3 (186KB pdf) (Actualización)
- Solucionar problemas de errores PostScript - Consejos de depuración. (158KB html)
- Revista Acumen - Archivo de artículos PostScript y programación en PDF. (directorio html de pdfs comprimidos)
- Ilustraciones matemáticas: Un manual de geometría y postscript - por Bill Casselman. (directorio html de capítulos en pdf y descargas de código)
- Subproceso con muchas implementaciones de algoritmo de clasificación (archivo usenet)
- Páginas de Don Lancaster Gurú
- Uso directo de Anastigmatix del lenguaje Postscript
- Depurador paso a paso de código abierto para el código PostScript

Preguntas frecuentes

- Preguntas frecuentes de Usenet (circa 1995)
- Preguntas Frecuentes de Wikilibros PostScript
- SO preguntas PostScript ordenadas por más vistos con frecuencia

Libros

- Manual de referencia del lenguaje Postscript, 1ed, 1985. Recomendado por su pequeño tamaño y fácil índice de operadores de las páginas de resumen (falta en ediciones posteriores).

- Posdata del mundo real. Capítulos de varios autores sobre diversos temas, incluida la excelente cobertura de medios tonos.

Espacios de nombres locales para funciones

Postscript es un lenguaje de espacios dinámicos o *LISP 1* . Pero proporciona las herramientas para implementar variables locales en procedimientos y otros efectos necesarios para implementar algoritmos.

Para los nombres locales en un procedimiento, cree un nuevo diccionario al principio y ábralo al final.

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

También puede combinar esto bien con un atajo para definir los argumentos de la función como variables.

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

Si necesita actualizar una variable * "global" * mientras el diccionario local está en la parte superior, use `store` lugar de `def` .

Hola mundo ejemplo

Seleccione una fuente y tamaño de fuente, seleccione la ubicación, `show` cadena.

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

Notas y errores comunes:

- No se puede establecer una fuente (lo que da como resultado que no haya texto o una fuente predeterminada (fea))
- Usando `findfont` y `setfont` pero olvidando la `scalefont` en el medio. El uso de `selectfont` nivel 2 evita este problema y es más conciso.
- No se puede establecer un punto con `moveto` , o establecer el punto fuera de la página. Para

el papel de carta de Estados Unidos, 8.5x11 es 792x612 ps puntos. Así que es fácil recordar aproximadamente 800x600 (pero un poco más corto y más ancho). Entonces `300 400` es aproximadamente el centro de la página (poco alto, poco a la izquierda).

- Olvidando llamar a `showpage` . Si `showpage` una vista previa de un programa ps con `gs` y no termina en `showpage` , `gs` puede mostrarle una imagen. Y, sin embargo, el archivo misteriosamente no producirá ningún resultado cuando intentes convertir a pdf o alguna otra cosa.

Plan de estudios

Lea la documentación en este orden para aprender fácilmente postscript:

1. Excelente tutorial de Paul Bourke: http://paulbourke.net/dataformats/postscript/
2. Libro azul, primera mitad, el tutorial oficial original: http://www-cdf.fnal.gov/offline/PostScript/BLUEBOOK.PDF
3. Libro verde, cómo usar postscript efectivamente: http://www-cdf.fnal.gov/offline/PostScript/GREENBK.PDF
4. Pensando en PostScript, 'nuff dijo: http://wwwcdf.pd.infn.it/localdoc/tips.pdf
5. Ilustraciones matematicas . Comience pequeño, construya a lo grande. Las matemáticas detrás de las curvas Bezier. El algoritmo de recorte de polígonos Hodgman-Sutherland. Transformaciones afines y transformaciones *no lineales* de la ruta. Dibujo 3D y sombreado de Gouraud. Desde el prefacio:

   Cuál [de las muchas herramientas para ayudar a producir gráficos matemáticos] para elegir aparentemente implica una compensación entre la simplicidad y la calidad, en la que la mayoría opina lo que perciben como simplicidad. La verdad es que la compensación es innecesaria: una vez que se ha realizado una pequeña inversión inicial de esfuerzo, lo mejor que se puede hacer en la mayoría de las situaciones es escribir un programa en el lenguaje de programación de gráficos PostScript. Prácticamente no hay límite para la calidad de la salida de un programa PostScript, y a medida que uno adquiere experiencia, las dificultades de usar el lenguaje disminuyen rápidamente. La aparente complejidad involucrada en la producción de figuras simples mediante la programación en PostScript, como espero que demuestre este libro, es en gran medida una ilusión. Y la cantidad de trabajo involucrado en la producción de figuras más complicadas será generalmente ni más ni menos de lo que es necesario.

Lea Empezando con postscript en línea:
https://riptutorial.com/es/postscript/topic/5616/empezando-con-postscript

Capítulo 2: Construcción de caminos

Examples

Dibujar (describir) un polígono

Este ejemplo intenta imitar el comportamiento de los operadores de construcción de ruta incorporados como `arc` .

Si hay un punto actual, `poly` primero dibuja una línea en (x, y) + (r, 0), de lo contrario comienza moviéndose hacia ese punto.

En lugar de `gsave` ... `grestore` (que tiene el efecto no deseado de descartar los cambios en la ruta actual que deseamos), guarda una copia de la matriz de transformación actual (CTM) tal como existe cuando se inicia la función.

Luego se `lineto` con cada punto posterior, que al escalar y rotar la matriz siempre está en (0,1). Finalmente, llama a `closepath` y luego restaura la matriz guardada como el CTM.

```
% 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
```

Iterando por un camino

Este fragmento vuelca el contenido de la ruta actual a la salida estándar. Utiliza el procedimiento ghostscript `=only` que puede no estar disponible en todos los intérpretes. Un procedimiento equivalente en los intérpretes de Adobe se llama `=print` .

`pathforall` es un operador de bucle que toma 4 cuerpos de procedimiento como argumentos que se llaman para los tipos específicos de elementos de trayectoria, el resultado de `moveto` , `lineto` , `curveto` , `closepath` y todos los demás operadores de construcción de trayectoria que se reducen a estos elementos.

```
{ 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
```

Papel cuadriculado

```
/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
```

Lea Construcción de caminos en línea:
https://riptutorial.com/es/postscript/topic/6679/construccion-de-caminos

Capítulo 3: Manejo de errores

Sintaxis

- { *-code-* } se detuvo { *-error-* } { *-no-error-* } ifelse% error atrapando marco
- $ error / errorname get% stackunderflow typecheck rangecheck etc
  $ error / newerror obtener% bool. poner falso para desactivar el error
  $ error / ostack obtiene% copia de la pila de operandos en el punto de error
- errordict / stackoverflow { *-additional-code-* / stackoverflow signalerror} poner
  % ejecuta código adicional en tipos de errores (aquí, el error / stackoverflow).

Observaciones

Hay dos niveles para el manejo de errores en postscript. Esta dicotomía se aplica tanto a la forma en que el intérprete maneja los errores como a los medios disponibles para que el usuario (programador) controle el manejo.

El nivel inferior es una estructura de control inusual que se `stop ... stopped` . `stopped` comporta de manera muy similar a una construcción de bucle, ya que establece una marca en la pila de ejecución que se puede saltar si se llama al operador de `exit` (para un bucle) o al operador de `stop` (para un -contexto `stopped`). A diferencia de una construcción en bucle, `stopped` produce un valor booleano en la pila que indica si se llamó a la `stop` (de lo contrario, se sabe que el procedimiento pasado a `stopped` se ha ejecutado hasta su finalización).

Cuando se produce un error de PostScript, como puede ser el `stackunderflow` , el intérprete busca el nombre del error en `errordict` que vive en `systemdict` . Si el usuario no ha reemplazado el procedimiento en `errordict` , entonces el procedimiento de error predeterminado tomará instantáneas de toda la pila y las colocará en `$error` , otro diccionario en `systemdict` . Finalmente, el procedimiento por defecto llamará a `stop` que saca el programa del usuario de la pila de exec y ejecuta el procedimiento de impresión de error del intérprete llamado `handleerror` en `errordict` .

Entonces, al utilizar todo este conocimiento, puede *detectar* errores si ajusta una sección de código en `{ ... } stopped` . Puede *volver a emitir* un error llamando a `stop` . Puede determinar qué tipo de error ocurrió con `$error /errorname get` .

También puede cambiar el comportamiento predeterminado para un tipo específico de error al reemplazar el procedimiento con ese nombre en `errordict` . O cambie el formato de impresión del informe de errores reemplazando `/handleerror` en `errordict` .

Examples

¿Hay un punto de corriente?

Rendimiento `true` en la pila si `currentpoint` ejecuta con éxito, o `false` si señala un error de `/nocurrentpoint` .

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

Secuencia de eventos cuando se señala un error.

La secuencia de un error suele ser:

1. el error se activa al buscar el nombre del error en `errordict` y ejecutar este procedimiento.
2. `errordict` procedimiento `errordict` llama a `signalerror` , pasándole el nombre de error.
3. `signalerror` toma instantáneas de las pilas, guardando las instantáneas en `$error` , y luego las llamadas se `stop` .
4. `stop` hace estallar la pila exec hasta que el operador detenido detiene el contexto de cierre más cercano.
5. si el programa no ha establecido su propio contexto detenido para detectar el error, será capturado por un nivel externo `stopped { errordict /handleerror get exec } if` el código de inicio lo llamó para poner en paréntesis a todo el programa del usuario.
6. `handleerror` utiliza la información en `$error` para imprimir un informe de error.

Señalizando (tirando) un error

La mayoría de las herramientas están estandarizadas con la excepción del nombre del operador para lanzar un error. En los intérpretes de Adobe, se llama `.error` . En ghostscript, se llama `signalerror` . Así que con esta línea puede usar `signalerror` en el código postscript para intérpretes de Adobe o ghostscript o xpost.

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

nombre de comando errorname **signalerror** -
Tome instantáneas de la pila en `$error` , luego `stop` .

P.ej.

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

Atrapando un error

Dado que la acción final del controlador de errores predeterminado es llamar a la `stop` , puede detectar errores de los operadores al incluir el código en una construcción `{ ... } stopped` .

```
{
```

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

imprimirá " `rangecheck` ", el error señalado por `get` cuando el índice está fuera del rango aceptable para esa matriz.

Errores de relanzamiento

Este fragmento implementa un procedimiento que se comporta como un operador de bucle postscript. Si el usuario `proc` llama a `exit` , detecta el error `invalidexit` para arreglar la pila de discos para el `end` al final. Cualquier otro error, excepto `invalidexit` se relanza llamando a `stop` .

```
% 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 ()=
```

Lea Manejo de errores en línea: https://riptutorial.com/es/postscript/topic/6199/manejo-de-errores

Creditos

S. No	Capítulos	Contributors
1	Empezando con postscript	Community, Kurt Pfeifle, luser droog
2	Construcción de caminos	luser droog
3	Manejo de errores	luser droog