PostScript für Workstations

26.7.93

Der folgende Artikel setzt voraus, dass der Leser die Bedeutung der Seitenbeschreibungssprache PostScript kennt. Display PostScript bietet dem Workstation-Benutzer die gleichen Möglichkeiten, welche PostScript zu einem Printing-Standard für “Text und Grafik” gemacht haben. Der Artikel zeigt, wie auf einer Workstation von Silicon Graphics (SGI) eine sehr einfache X11-Anwendung entwickelt werden kann, die neben den Xlib-Aufrufen zusätzlich PostScript-Befehle verwendet.

Von Dr. Peter Vollenweider, Universität Zürich, CH-8057 Zürich

PostScript: Ein eingewickeltes Beispiel (Wrap)

Das folgende PostScript-Beispiel (hitherz) ist als Wrap-Definition formuliert, es ist daher auf allen Plattformen mit Display PostScript (DPS/X) oder X11R5 lauffähig. Eine Wrap-Definition ist ein PostScript-Fragment, das von den Befehlen defineps und endps eingeklammert ist. Das Werkzeug PSWrap dient dazu, die Wrap-Definition in C-Code zu übersetzen. Dieser C-Code kann schliesslich von einer DPS/X-Applikation (X11-Client) importiert werden. Das Beispiel hitherz.psw enthält die folgenden PostScript-Komponenten:

- Bézier-Kurven konstruieren die Rundungen eines Herzens.
- Ein Benutzerpfad definiert das Herz.
- “Hit Detection” prüft, ob eine Koordinatenposition innerhalb oder ausserhalb des Herzens liegt.
- Ein Rechteck-Operator zeichnet ein kleines Quadrat, um die Koordinatenposition zu markieren.

Bézier-Kurven

... können beliebige Formen beschreiben. Betrachten Sie die folgende Abbildung aus einem Tutorial:

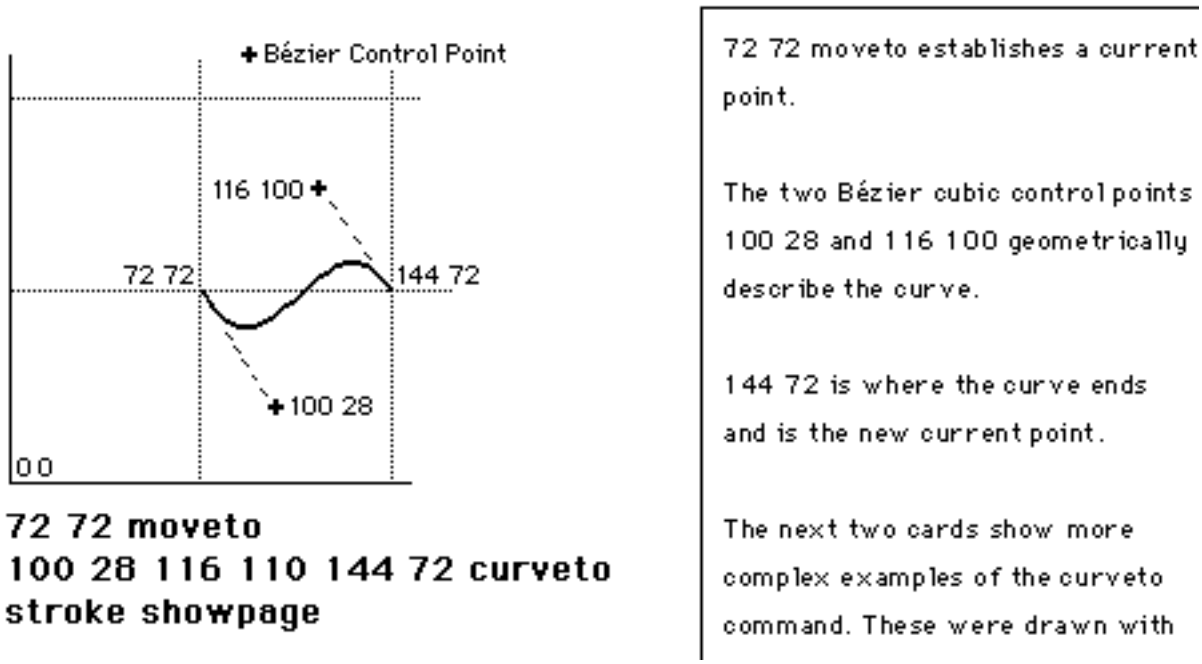

Abbildung 1: Der Operator curveto konstruiert eine geglättete Kurve; die Bézier-Kontrollpunkte (BCP) sind mit Pluszeichen (+) markiert.

Die Gestalt einer Bézier-Kurve wird durch den Start- und Endpunkt (Punkt 0 bzw. 3) und durch die Bézier-Kontrollpunkte (BCP) ausserhalb der Kurve (Punkte 1 und 2) bestimmt. Die beiden BCP's im Beispiel sind (100, 28) und (116, 100). Für Mathematiker: eine Bézier-Kurve ist eine kubische Kurve, die jedoch eine Ueberschneidung zulässt. Die im obigen Beispiel gestrichelten Geraden von Punkt 0 nach Punkt 1 sowie von Punkt 2 nach Punkt 3 bilden Tangenten durch Anfangs- bzw. Endpunkt der Bézierkurve. Die meisten grafischen Formen und Schriftzeichen werden mit Bézier-Kurven beschrieben -- ausser man hat es ausschliesslich mit geraden Strichen und Rechtecken zu tun. Ein Beispiel:

```
% Es folgen zwei Bezier-Kurven:
-36.0 36.0 0.0 72.0 36.0 36.0 rcurveto
% Rundung links
36.0 36.0 72.0 0.0 36.0 -36.0 rcurveto
% Rundung rechts
```

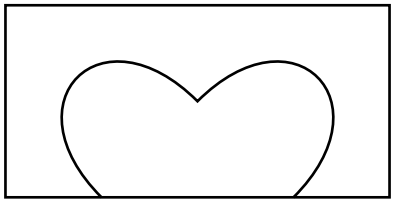

Abbildung 2: Zwei Bézier-Kurven

Die Syntax von *curveto* lautet:

$x_1\ y_1\ x_2\ y_2\ x_3\ y_3$ curveto

Dabei wird dem aktuellen Pfad ein Segment hinzugefügt, das von der aktuellen Position zum Punkt (x_3,y_3) geht und dabei die Punkte (x_1,y_1) und (x_2,y_2) als BCP's verwendet. Nach der Ausführung des Operators wird der Punkt (x_3,y_3) zur aktuellen Position. Bei *rcurveto* werden die werden die Pfade nicht in absoluten Koordinaten (ausgehend vom Ursprung), sondern in relativen Bewegungen angegeben (ausgehend von der aktuellen Position).

Benutzerpfade

Ein vom Benutzer definierter Pfad ist ein gewöhnlicher Pfad, der entweder gefüllt oder ausgezogen wird. Allerdings wird der Programmierer gezwungen, die Bounding-Box des Pfades (die untere linke Ecke und die obere rechte Ecke des Rahmens) anzugeben, damit Display PostScript die Darstellung beschleunigen kann. Um die Performance bis zu dreimal zu verbessern, kann der Pfad in ein Cache aufgenommen werden. Der Einsatz von Display PostScript ist nur dann gerechtfertigt, wenn es genügend schnell ist. Pfad-Konstruktion für ein Herz:

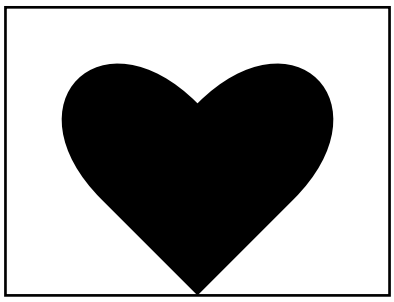

Abbildung 3: Herz als gefüllter Benutzerpfad (Operator ufill)

```
/herz % Definition des Benutzerpfades herz
{
 ucache
 128 350 272 458 setbbox % BoundingBox
 200 350 moveto % Spitze des Herzens unten
 -36.0 36.0 rlineto
 % Es folgen zwei Bezier-Kurven:
 -36.0 36.0 0.0 72.0 36.0 36.0 rcurveto
   % Rundung oben links
 36.0 36.0 72.0 0.0 36.0 -36.0 rcurveto
   % Rundung oben rechts
 closepath
}
```

```
cvlit def
herz ufill
```

Während der Cache-Befehl freiwillig ist, ist der Operator *setbbox* unbedingt erforderlich, um die Bounding-Box des Benutzer-Pfades anzugeben: (128, 350) ist untere linke Ecke, und (272, 458) ist obere rechte Ecke. Beachten Sie, dass die Bounding-Box auch die ausserhalb der kubischen Kurve liegenden Kontrollpunkte einschliesst, z.B. die beiden BCP's der ersten Kurve: (128, 422) und (164, 458). Die neuen Operatoren *ufill* und *ustroke* konstruieren einen Pfad und färben oder füllen diesen. Dabei darf der Pfad aus einer Kombination folgender Elemente bestehen: moveto, rmoveto, lineto, rlineto, curveto, rcurveto, arc, arcn, arct, closepath. Der Operator *cvlit* macht das Objekt "literal", sodass die Prozedur noch nicht ausgeführt wird.

Man könnte noch einen Schritt weiter gehen und den Benutzerpfad in kompakter Form darstellen, indem die Pfad-Elemente hexadezimal verschlüsselt werden:

```
[
   [
   128 350 272 458
   200 350
   -36 36
   -36 36 0 72 36 36
   36 36 72 0 36 -36
   ]
   <0B00010406060A> % in kompakter Form
]  ufill
```

Während die Operanden in herkömmlicher Art und Weise angegeben sind, werden die Pfad-Elemente wie folgt verschlüsselt:

0B	ucache
00	setbbox
01	moveto
04	rlineto
06	rcurveto
06	rcurveto
0A	closepath

Um eine weitere Verbesserung zu erreichen, könnten zusätzlich auch die Operanden hexadezimal kodiert werden.

Wir könnten *noch* einen Schritt weiter gehen und den Benutzerpfad in eine eigene Wrap-Definition auslagern, sowie den Benutzerpfad als Benutzerobjekt im PostScript-Server abspeichern (defineuserobject). Ein Benutzerobjekt lässt sich durch eine ganze Zahl identifizieren (execuserobject). Adobe Systems empfiehlt, jene

Objekte im PostScript-Server abzulegen, die einfach sind, häufig verwendet und selten verändert werden.

Hit Detection

Eine Workstation kann nicht nur Grafik ausgeben, sondern sie unterstützt auch die grafische Eingabe mittels der Maus. Display PostScript enthält eine Reihe von Operatoren, die feststellen, ob die "Mausspitze" oder ein Benutzerpfad auf oder innerhalb des aktuellen Pfades bzw. eines zweiten Benutzerpfades liegt. Das Beispiel prüft, ob die als Argument übergebene Koordinatenposition innerhalb oder ausserhalb des Benutzerpfades liegt:

```
X Y herz inufill
% gibt true oder false auf den Stack
/bool exch def
```

Der Befehl

x y benutzerpfad inufill

gibt *true* auf den Stack, wenn die von *benutzerpfad* umschlossene Form das Pixel am Ort (x,y) berührt.

Adobe Systems empfiehlt, vor dem Aufruf von inufill in der Applikation zu testen, ob die Koordinatenposition überhaupt innerhalb der Bounding Box des Benutzerpfades liegt. Wenn nicht, erübrigt sich der Aufruf des PostScript-Operators inufill, dessen Ausführung relativ lang dauert (round-trip, Antwort vom Server).

Wie man die Position des Mauszeigers festhalten kann, ist natürlich von Plattform zu Plattform verschieden. Ein X-Server kommuniziert mit den X-Clients durch das Senden von Ereignissen (events). Solche Ereignisse können durch Benutzeraktionen, durch X-Clients oder als Folge eines anderen Ereignisses erzeugt werden. Der X-Server speichert jedes Ereignis in einer Datenstruktur (Ereignis-Struktur) ab. Dieser Datensatz enthält auch die beiden Koordinaten x und y der Mausposition: *event.xbutton.x* bzw. *event.xbutton.y.* Voraussetzung ist allerdings, dass das C-Programm die Ereignis-Struktur deklariert hat:

```
/* C-Code */
XEvent      event; /* Ereignis-Struktur */
XSelectInput(dpy, win, ...); /* Ereignis anfordern */
XNextEvent(dpy, &event); /* liest naechstes Ereignis */
```

Bei der Uebernahme der x- und y-Koordinaten muss das Koordinatensystem konvertiert werden (siehe Client Library: z.B. *PSitransform*), da bei PostScript der Ursprung nicht oben links, sondern unten links liegt. Das *X Window System Programmer's Supplement* meint dazu: "Coordinate conversions are required under the following conditions: If you use the PostScript imaging model to render gra-

phics using coordinates received from X11 events, the X11 coordinates must first be converted into user space coordinates. ...”

Adobe Systems weist allerdings darauf hin, dass die Ausführung von itransform relativ lange dauert, weil der PostScript-Server der Applikation die Argumente zurückgeben muss (round-trip). Es wäre von Vorteil, wenn die Applikation die Koordinaten-Konversion selbst berechnen könnte. Dies setzt allerdings voraus, dass die aktuelle Transformationsmatrix (currentmatrix, invertmatrix) in der Applikation, d.h. im C-Programm, abgespeichert worden ist (was im einfachen Beispiel nicht der Fall ist).

Rechteck-Operatoren

Rechteck-Operatoren: z.B. *rectfill, rectstroke.* Beide Operatoren leiten sich von gewöhnlichen PostScript-Sequenzen ab. *rectfill* bildet ein gefülltes Rechteck, während bei *rectstroke* der Pfad nicht gefüllt, sondern ausgezogen wird. Der PostScript-Programmierer braucht also nur noch einen Operator aufzurufen, z.B.:

```
1 1 0 setrgbcolor
0 0 100 100 rectfill   % gelbes Quadrat

1 0 0 setrgbcolor
100 0 100 100 rectfill % rotes Quadrat
```

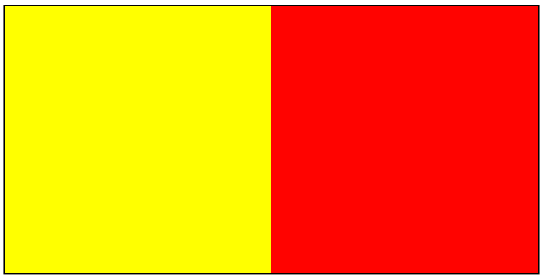

Abbildung 4: Quadrate in Gelb und Rot

Die Syntax von rectstroke und rectfill lautet:

```
x y width height rectstroke
x y width height rectfill
```

Dabei geben *x* und *y* die untere linke Ecke an, und *width* und *height* die Breite und Höhe des Rechtecks. Falls PostScript Level 2 nicht verfügbar ist, können Sie den Operator rectfill wie folgt emulieren: newpath -- moveto -- rlineto -- rlineto -- rlineto -- closepath -- fill. Das Beispiel *100 100 72 36 rectfill* kann wie folgt emuliert werden:

```
newpath 100 100 moveto
72 0 rlineto
0 36 rlineto
```

```
-72 0 rlineto
closepath fill
```

Auf einer Workstation sind nicht nur die Fenster, sondern auch viele andere Objekte rechteckig. Das folgende Beispiel zeichnet ein kleines weisses Quadrat:

```
bool
{
 1 setgray % weiss
 X 2 sub Y 2 sub 4 4 rectfill % Rechteck
}
if
```

Das kleine Quadrat wird aber nur dann gezeichnet, wenn die Variable bool *true* ist.

Die Wrap-Definition hitherz.psw

Nun folgt der vollständige Code des PostScript-Beispiels:

```
defineps hitherz(float X, Y)
% Argumente: Uebernahme der Koordinatenposition

% ------------------------------------------------

% 1. Schritt:
% Zuerst wird der Benutzerpfad definiert
% und gefuellt:

/herz % Definition des Benutzerpfades herz
[
   [
   128 350 272 458 % setbbox
   200 350   % moveto
   -36 36   % rlineto
   -36 36 0 72 36 36 % rcurveto
   36 36 72 0 36 -36 % rcurveto
   ]
   <0B00010406060A> % kompakte Form
]
cvlit def
```

```
herz ufill

% ----------------------------------------

% 2. Schritt:
% Jetzt folgt die Pruefung,
% ob die uebergebene Koordinatenposition
% innerhalb oder ausserhalb von herz liegt:

X Y herz inufill
% gibt true oder false auf den Stack
/bool exch def

% ----------------------------------------

% 3. Schritt:
% Schliesslich wird ein kleines weisses Quadrat
% an der uebergebenen Position gezeichnet,
% falls bool true ist:
bool
{
 1 setgray
 X 2 sub Y 2 sub 4 4 rectfill % Rechteck
}
if

% ----------------------------------------

endps
```

Bevor eine DPS/X-Anwendung die obige Wrap-Definition aufrufen kann, kommt das Werkzeug PSWrap zum Einsatz. Der Präprozessor PSWrap kann auch die meisten Operatoren von PostScript Level 2 verarbeiten (im Beispiel etwa *ufill*, *inufill*, *rectfill*). Die Umwandlung erfolgt im UNIX-Fenster mit dem Befehl

```
pswrap hitherz.psw -o hitherz.h
```

PSWrap liest das File hitherz.psw und schreibt in das File hitherz.h.

Falls nun der Aufruf in einem C-Programm

```
hitherz (200., 400.); /* Wrap-Aufruf */
```

lautet, dann zeichnet die Wrap-Definition das Herz, gibt den Wert *true* auf den Stack zurück und markiert daher den Punkt (X=200,Y=400) durch ein kleines weisses Quadrat. Läge der Punkt (X,Y) ausserhalb des Herzens, würde das kleine weisse Quadrat nicht gezeichnet, da der if-Operator den Operator rectfill nicht ausführen würde. Im File hitherz.psw wird der vom Operator inufill zurückgegebene Wert explizit vom Operanden-Stack geholt und in die Variable *bool* geschrieben, um zu verdeutlichen, dass dieser Wert -- entweder *true* oder *false* -- dem if-Operator als Operand dient.

Der von PSWrap erzeugte C-Code, kürzen???

Der Präprozessor PSWrap erzeugt einen recht umfangreichen Code. Aus Platzgründen sind hier nur der Anfang und der Schluss des Files *hitherz.h* abgedruckt:

```
/* hitherz.h generated from hitherz.psw
by unix pswrap V1.009  Wed Apr 19 17:50:24 PDT 1989
*/

#include <DPS/dpsfriends.h>
#include <string.h>

#line 1 "hitherz.psw"
#line 10 "hitherz.h"
void hitherz(X, Y)
float X, Y;
{
  typedef struct {
  unsigned char tokenType;
  unsigned char topLevelCount;
  unsigned short nBytes;
. . .

   {DPS_EXEC|DPS_NAME, 0, 0, 0}, /* herz */
   {DPS_EXEC|DPS_NAME, 0, DPSSYSNAME, 179}, /* ufill */
. . .
```

Der grösste Teil des erzeugten Codes ist hier weggelassen.

```
  _dpsP[6].val.realVal =
  _dpsP[18].val.realVal = X;
  _dpsP[7].val.realVal =
  _dpsP[21].val.realVal = Y;
  DPSBinObjSeqWrite(_dpsCurCtxt,(char *) &_dpsF,436);
}
#line 49 "hitherz.psw"
```

NeWS kennt im Gegensatz zu X11R5 keinen PSWrap-Präprozessor.

Der C-Code des DPS/X-Client

Jede Display-PostScript-Applikation enthält mindestens vier Elemente:

• X11-Initialisierungen

• Erzeugen eines PostScript-Kontextes

• Haupt-Ereignisschleife

• Ereignisbehandlung mit PostScript-Grafik bzw. PostScript-Ausgabe

Nun folgt ein vollständiges C-Programm. Die sehr einfache DPS/X-Anwendung öffnet zuerst ein Fenster, erzeugt einen PostScript-Kontext und fängt dann bei jedem Mausklick (ButtonRelease) die Mausposition ab, ruft die Wrap-Definition hitherz auf und zeichnet das Herz. Der entsprechende C-Code sieht wie folgt aus:

```
/* C-Code, eine kleine X-Applikation 19.1.93/vo */

#include <stdio.h>
#include <signal.h>
#include <X11/X.h>
#include "X11/Xlib.h"

#include <DPS/XDPS.h>
#include <DPS/XDPSlib.h>
#include <DPS/dpsXclient.h>
#include <DPS/psops.h>

#define W_HEIGHT 600
#define W_WIDTH 600
#define FENSTER_HOEHE 600

/* Wrap-Definition: */
#include "hitherz.h" /* PostScript-Wrap */
```

Die hier importierte Wrap-Definition *hitherz* finden Sie weiter vorne beschrieben; sie wurde von pswrap umgewandelt. Nun folgt das Hauptprogramm:

```
main(argc, argv)
int argc;
char **argv;
{
 char *displayname = "";
 Display *dpy;   /* Bildschirmstruktur */
 GC gc;          /* X-Ressource */
```

```
long mask;
int c;
Window win;     /* X-Ressource */

DPSContext ctxt;    /* PostScript-Kontext */

struct {float x; float y;} hitpoint;
XEvent event;              /* Ereignis */

dpy = XOpenDisplay(displayname);
if (dpy == NULL)
        {
  fprintf(stderr, "%s: Can't open display %s!\n",
  argv[0], displayname);
  exit(1);
        }
```

Falls bei der Programmausführung die Fehlermeldung *Can’t open display* erscheint, ist vermutlich die DISPLAY-Variable nicht richtig auf den X-Server gesetzt.

```
gc = XCreateGC(dpy, RootWindow (dpy, DefaultScreen
   (dpy)), 0, NULL);
```

Die obige Prozedur XCreateGC() gibt die X-Ressource “Graphics Context” an, d.h. die Wiedergabeattribute, mit denen gezeichnet werden soll.

```
XSetForeground(dpy, gc, BlackPixel (dpy,
   DefaultScreen (dpy))); /* Vordergrund */
XSetBackground(dpy, gc, WhitePixel (dpy,
   DefaultScreen (dpy))); /* Hintergrund */

win = XCreateSimpleWindow(dpy,
      DefaultRootWindow(dpy),
        3, 385, W_WIDTH, W_HEIGHT, 1,
             BlackPixel(dpy, DefaultScreen(dpy)),
            WhitePixel(dpy, DefaultScreen(dpy)));
```

Die obige Prozedur XCreateWindow() erzeugt ein neues X-Fenster bzw. ein X-Drawable.

```
XMapWindow(dpy, win);
```

```
ctxt = XDPSCreateSimpleContext(
  dpy, win, gc, 0, W_HEIGHT,
  DPSDefaultTextBackstop,
     DPSDefaultErrorProc, NULL);
```

Nachdem Drawable und GC festgelegt sind, kann die obige Prozedur nun den “Execution Context” von PostScript erzeugen.

```
if (ctxt == NULL)
        {
 fprintf (stderr, "first: XDPSCreateSimpleContext
 returns NULL.\n");
 exit (-1);
}
```

Falls bei der Programmausführung die Fehlermeldung *XDPSCreateSimpleContext returns NULL* erscheint, ist Ihr X-Server (z.B. NeWS) vermutlich kein DPS/X-Server. Als DPS/X-Server kommen die Workstations von IBM, Silicon Graphics, DEC und (ab Mitte 1993) von Sun in Frage.

```
DPSSetContext(ctxt); /* PostScript-Kontext */

XSelectInput(dpy, win, ButtonPressMask |
                    ButtonReleaseMask);
                  /* Ereignisse anfordern */
```

Jetzt folgt die Ereignis-Schleife (main event loop), welche alle Ereignisse behandelt:

```
while (TRUE) /* main event loop */
   {
   XNextEvent(dpy, &event); /*naechstes Ereignis*/

   switch(event.type) {

     case ButtonRelease:
     PSitransform((float) event.xbutton.x,
        (float) -(FENSTER_HOEHE - event.xbutton.y),
                  &hitpoint.x, &hitpoint.y);
                    /* konvertiert x und y */
     PSsetrgbcolor(.0, 1., .0);
```

```
        hitherz(hitpoint.x, hitpoint.y);
                    /* Wrap-Aufruf zeichnet Herz */
         }
     }
  }
```

Bei der Uebernahme des nächsten Ereignisses durch XNextEvent() werden die Koordinaten des Mauszeigers im Datensatz “event record” unter *event.xbutton* abgelegt. Das erste Feld *type* jedes Datensatzes ist der Event-Typ. Falls ein bestimmtes Ereignis, nämlich “ButtonRelease”, eintritt, wandelt die Client-Prozedur PS-itransform( ) die Zeigerposition in das PostScript-Koordinatensystem um. Zusätzlich wird die grüne Farbe festgelegt (PSsetrgbcolor). Schliesslich kann die Wrap-Definition *hitherz* das Herz zeichnen und prüfen, ob sich die übergebene Koordinatenposition *hitpoint* innerhalb oder ausserhalb des Herzens befindet. Falls das Resultat positiv ist, markiert die Wrap-Definition die Position mit einem kleinen weissen Quadrat.

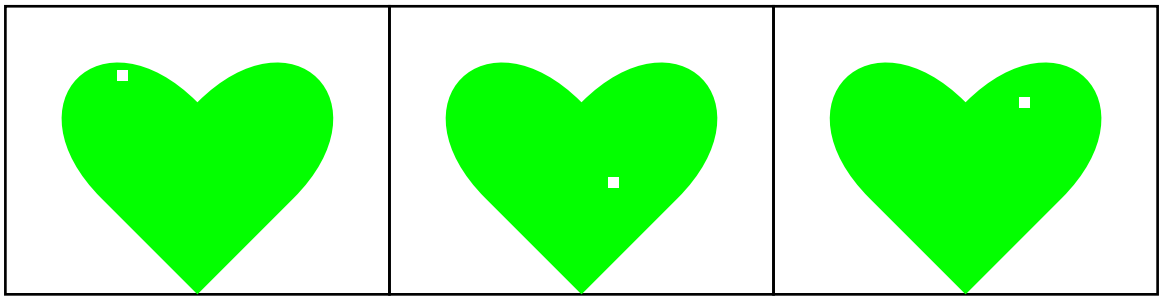

Abbildung 5: X-Window: dreimal auf das Herz geklickt

Noch eine Bemerkung zu den beiden “Client Library”-Aufrufen *PSitransform( )* und *PSsetrgbcolor( )*: derartige Aufrufe bilden neben den PostScript-Wraps eine weitere Möglichkeit, PostScript-Operatoren an einen Interpreter zu senden.

Uebersetzen, Laden und Testen

Das obige C-Programm wurde auf einer IBM RS/6000 wie folgt geladen und ausgetestet:

```
cc -ldps -lX11 -lm main.c
```

Es ist aber auch auf anderen Plattformen lauffähig. Das Flag “-ldps” steht für die PostScript Client Library. Die IM-Library (-lIM), die Xt-Library (-lXt) und die Xm-Library (-lXm) werden nicht benötigt. Auf anderen Workstations, z.B. Silicon Graphics Crimson, lautet der Ladeaufruf wie folgt:

```
cc main.c -ldps -lXext -lX11 -lm
```

Als X-Server setzte ich eine IBM-Workstation und eine Workstation von Silicon Graphics ein. Szenario an der Uni Zürich:

- der Endbenutzer sitzt an einer Iris Indigo,
- die DPS/X-Applikation läuft auf einer IBM-Maschine des Typs RS/6000.

Das Beispiel liesse sich noch verbessern, indem der Benutzerpfad (herz) in einer eigenen Wrap-Definition beschrieben und allenfalls als Benutzerobjekt definiert würde (defineuserobject). Auch das C-Programm könnte noch verfeinert werden und ein Benutzerobjekt direkt ausführen (PSexecuserobject). Ferner wäre es empfehlenswert, vor dem Aufruf von inufill im C-Programm zu testen, ob die Koordinatenposition überhaupt innerhalb der Bounding Box des Benutzerpfades liegt.

Programmieren mit Display PostScript

Um mit Display PostScript zu programmieren, stehen Ihnen somit zwei Wege offen:

1. Sie verwenden PostScript-Operatoren in Form von “Client Library”-Aufrufen, z.B. *PSsetrgbcolor(.0, 1., .0)*. Die Client Library ist eine Prozedurensammlung, welche die Schnittstelle zwischen einer Client-Anwendung und Display PostScript (dem Bildschirm-Server) bildet.

   Bei Display PostScript führt der PostScript-Server einen eigenen Execution Context für jede Applikation, da der Benutzer oder die Benutzerin manchmal mehrere Applikationen gleichzeitig aktiviert hat. Die aktuelle Ausführungsumgebung wird bei jedem Aufruf einer DPS-Funktion angegeben. Beispiel:

   ```
   DPStranslate (context, 100.0, 100.0);
   ```

   Dabei gibt *context* den Execution Context an, in dem der PostScript-Operator ausgeführt werden soll. Das Argument “context” ist ein *DPSContext*, der in dpsclient.h definiert ist. Im Gegensatz zu den Funktionen mit dem Präfix PS ist das erste Argument einer DPS-Funktion immer ein DPSContext. Wenn Sie es nur mit einer einzigen Ausführungsumgebung zu tun haben, könnten (und sollen) Sie die Funktion *PStranslate( )* statt DPStranslate( ) aufrufen.

2. Sie nutzen den Präprozessor PSWrap, der Ihre PostScript-Fragmente in C-Code übersetzt. Dieser Weg ist der einfachere. Die PostScript-Fragmente müssen keineswegs vollständige Programme sein; eine PSWrap-Definition (kurz: Wrap) baut z.B. einen Pfad auf, definiert eine PostScript-Prozedur, setzt einen Schriftzug oder zeichnet ein grafisches Objekt. PSWrap wandelt ein .psw-File in ein C-File um, worauf der C-Compiler ein Objektmodul erzeugt, das dann gebunden werden kann, vergleiche das Beispiel *hitherz*.

   Dabei wird das “eingewickelte” PostScript-Programm weder interpretiert noch überprüft. Der PostScript-Interpreter entdeckt einen PostScript-Fehler

daher erst bei der Ausführung einer Anwendung, “when running the application”. Auf der UNIX-Workstation finden Sie allfällige Fehlermeldungen im Konsolenfile.

Das weiter vorne vorgestellte C-Programm enthält insgesamt drei Arten von Bibliotheksaufrufen:

X11, Xlib:	DPS Client Library:	PSWrap:
Fenster: XOpenDisplay XCreateGC XSetForeground XSetBackground XCreateSimpleWindow XMapWindow *Ereignis:* XSelectInput XNextEvent	*PostScript-Kontext:* XDPSCreateSimpleContext DPSSetContext *PostScript-Operatoren:* PSitransform PSsetrgbcolor	hitherz *(siehe vorn)*

Diese Zusammenstellung zeigt, dass die PostScript-Programmierung unter X11 nicht gerade einfach ist. Eine objektorientierte Entwicklungsumgebung würde vermutlich die Programmierung vereinfachen. In NextStep könnten einfach die beiden Methoden *mouseDown* (zum Abfangen der Cursor-Position) und *drawSelf* (für die PostScript-Ausgabe) neu implementiert werden. Dabei erfolgt der Aufruf einer Wrap-Definition, z.B. hitherz, auf gleiche Art und Weise wie in der X-Anwendung:

```
/* Objective C, Implementation */

@implementation <neuer Klassenname>

- mouseDown:(NXEvent *) e
{
  [self display]; /* ruft drawSelf */
}

- drawSelf:(NXRect *)r: (int) count
{
  hitherz (X, Y); /* Wrap-Aufruf */
}
@end
```

Das *Application Kit* von NextStep hat die Methode mouseDown in der Klasse *Control* implementiert, die Methode drawSelf in der Klasse *View.*

Der grafische Status (X-Drawable)

Das wichtige Thema des grafischen Status ist bisher zu kurz gekommen. Zum grafischen Status zählen Parameter wie Farbe, Font, Schriftgrösse, Position, Strichdicke, Koordinatensystem, usw. Mit PostScript Level 2 können Sie beliebig viele grafische Zustände als gstate-Objekte festhalten (currentgstate) und bei Bedarf in beliebiger Reihenfolge wieder aktivieren (setgstate). Adobe Systems empfiehlt, gstate-Objekte nur im Rahmen von Display PostScript einzusetzen, z.B. beim Wechseln des Ausgabegerätes, sprich X-Drawable. Auszug aus *Programming the Display PostScript System with X:* "The most important use for a gstate object is as a link to an X drawable (window or pixmap). One component of the graphics state is the X drawable, as specified by the *device* graphics state parameter. ... Because a gstate object describes the graphics state, it can be used to install a different drawable at different times."

Der Operator setgstate erhält eine ähnliche Bedeutung wie setXgcdrawable. Die gstate-Objekte können somit zum Zug kommen, wenn eine Anwendung mehrere X-Fenster verwendet, d.h. wenn ein PostScript-Kontext mit mehreren X-Fenstern verbunden ist. Dies ist in unserem Beispiel allerdings nicht der Fall.

Vier Tips betreffend Performance

1. Verwende Wrap-Definitionen statt einzelner Client-Library-Aufrufe, wenn mehrere PostScript-Operatoren zusammen ausgeführt werden können!
2. Verzichte auf Berechnungen innerhalb einer Wrap-Definition, wenn die Berechnungen innerhalb der Applikation (im C-Programm) durchgeführt werden können!
3. Das X-Protokoll ist asynchron und gilt gleichzeitig in beide Richtungen (Applikation -- Server). Die Applikation sendet gewöhnlich einen Strom von Requests, ohne auf die Antworten vom DPS/X-Server zu warten. Requests, die eine Antwort erfordern, sollten auf ein Minimum reduziert werden, da die Applikation auf die Antwort warten muss. Die Kombination von Request und Antwort wird manchmal als *round-trip* bezeichnet. Beschränke round-trips auf ein Minimum!
4. Adobe Systems schlägt vor, jene grafischen Objekte im PostScript-Server abzulegen, die einfach sind, häufig verwendet und selten verändert werden.

Und was ist mit NeWS ?

Nachdem Sunsoft angekündigt hat, 1993 Display PostScript in das Betriebssystem Solaris einzubauen, scheint die Zukunft von NeWS nicht mehr gesichert zu

sein. Dazu meint Jonathan Monsarrat, Brown University: “As of October 1992, Sun management signed a deal with Adobe to adopt Display PostScript for the Sun. The future of NeWS is still undecided (but it looks bad).”

Literatur

Adobe Systems Inc. (1990),
PostScript Language, Reference Manual, 2nd Edition,
Addison-Wesley.

Adobe Systems Inc. (1993),
Programming the Display PostScript System with X,
Addison-Wesley.

Adobe Systems Inc. (1990),
X Window System Programmer’s Supplement to the Client Library Reference Manual,
IBM SC23-2211.

Vollenweider, Peter (1992),
PostScript für Workstations -- Display PostScript usw.
Addison-Wesley.