liner V.1.09

J. Kurrek

20.03.2013

Zusammenfassung

Dieses Programm dient zur Graphikprogrammierung mit einfachen Befehlen wie `line()`, `pixel()` oder `clear()`.

Es versucht damit eine Lücke auszufüllen, die seit dem Auslaufen der DOS-Programmierung besteht und den Informatik- bzw. Programmierunterricht trocken erscheinen lässt.

Direkte Graphikprogrammierung unter Benutzeroberflächen wie X11 oder MS-Wind.(TM) kann nämlich sehr zeitaufwändig und frustrierend sein: Für ein einfaches `hello`-Fenster sind oft 200 bis 500 Zeilen notwendig. Auf der anderen Seite gibt es komplexe Objektsammlungen, die diese Arbeit abnehmen – dann aber hat man meist als Anfänger nur noch die Möglichkeit, kleine Programmschnipsel einzubinden – oder aber Schildkröten- oder Hamstersimulationen, die zum Lernen gut geeignet sind, die man aber irgendwann wieder verlassen muss, um auf den rauhen Boden der Programmierwirklichkeit zu stoßen.

Mit dem gleichen Ansatz wie `liner` kann man übrigens auch Ghostscript verwenden, auch dies ist optional nutzbar.

1 Installation

Die Installation wird in der Datei `INSTALL` beschrieben. Für Linux-Nutzer reicht es, die fertig compilierte Version zu verwenden und zu installieren (siehe `QUICK INSTALL`).

Nur für den Fall, dass das nicht klappt, ist die übliche Installation notwendig (siehe `NORMAL INSTALL`).

2 Einbindung bei der Programmierung

Ein Programm, das liner benutzt, muss zuerst die Datei `liner.h` einbinden mit der Zeile

```
#include "liner.h"
```

Die Datei muss sich dazu im aktuellen Verzeichnis befinden. Falls sie sich in einem Include-Verzeichnis (`/usr/include` oder `/usr/local/include`) befindet, darf sie mit der folgenden Zeile eingebunden werden:

```
#include <liner.h>
```

3 Graphikausgabe beginnen

```
int liner_init(void)
```

Nach dem Befehl `liner_init();` können beliebige weitere Graphikbefehle benutzt werden. Ein Rückgabewert ungleich null zeigt einen Fehler an. Das Ausgabefenster hat eine Größe von 640 mal 480 Pixel (Breite mal Höhe).

4 Graphikausgabe beginnen für Fortgeschrittene

```
int liner_init_xywh(int initlang, int initaction,
                    unsigned int x, unsigned int y,
                    unsigned int w, unsigned int h)
```

Hier können zusätzlich Parameter angegeben werden:

a) `initlang`: 0 für liner-Ausgabe, 1 für Ghostscript

b) `initaction`: 0 für Bildschirmausgabe, 1 für Datei, 2 für Ausgabe auf `stdout` zur Weitergabe an eine Pipe.

c) `x`: x-Koordinate des Fensters (vom linken Rand aus)

d) `y`: y-Koordinate des Fensters (vom oberen Rand aus)

e) `w`: Breite des Fensters

f) `h`: Höhe des Fensters

5 Strecke zeichnen

```
void line(unsigned int x1, unsigned int y1,
          unsigned int x2, unsigned int y2)
```

`x1` und `y1` sind die Koordinaten des Anfangspunktes, `x2` und `y2` die Koordinaten des Endpunktes der Strecke. Dieses Beispiel zeichnet ein großes X:

```
line(1,1,100,100);
line(1,100,100,1);
```

6 Punkt zeichnen

```
void pixel(unsigned int x, unsigned int y)
```

An der Position (x,y) wird ein Punkt gezeichnet. Beispiel:

```
pixel(30,60);
```

Bei der Ausgabe mit Ghostscript wird statt eines Punktes eine sehr kurze Linie gezeichnet.

7 Rechteck zeichnen

```
void rectangle(unsigned int x1, unsigned int y1,
               unsigned int x2, unsigned int y2,
               int fill);
```

`x1` und `y1` sind die Koordinaten einer Ecke, `x2` und `y2` die Koordinaten der gegenüberliegenden Ecke eines Rechtecks mit zwei waagerechten und zwei senkrechten Kanten. Ist `fill` gleich null, bleibt das Rechteck hohl, andernfalls wird es gefüllt:

```
rectangle(20,20,100,100,1);
```

8 Kreis oder Kreisbogen zeichnen

```
void arc(unsigned int mx, unsigned int my, unsigned int r,
         int phi1, int phi2, int fill)
```

`mx` und `my` sind die Koordinaten des Mittelpunktes, `r` ist der Radius. `phi1` ist der Startwinkel und `phi1+phi2` der Endwinkel des Kreisbogens (`phi2` gibt also an, welchen Winkel der Bogen umfasst). Wählt man also `phi=360`, erhält man einen vollen Kreis. Bei `fill` ungleich null wird der Kreis (oder der Kreisbogen) ausgefüllt. Beispiel:

```
arc(210, 210, 205, 45, 270, 1);
```

9 Linienzug zeichnen

`void polyline(unsigned int n, int filled, unsigned int koords[])`

Mit dieser Funktion kann ein Linienzug, bestehend aus mehreren zusammenhängenden Strecken, erstellt werden. `n` gibt an, wie viele Punkte vorhanden sind, `filled` sagt wieder aus, ob das Gebilde gefüllt werden soll, und `koords` ist ein Array aus den x- und y-Koordinaten der Punkte (x0, y0, x1, y1, x2, y2 usw.):

```
int x[10]={0,100, 0,40, 50,0, 100,40, 100,100};
polyline(5,1,x);
```

10 Graphikfenster leeren

`void clear(void);`

Mit dem Befehl `clear();` kann man alle bisherige Ausgaben im Graphikfenster löschen.

11 Farbe ändern (RGB)

`void color(unsigned int r, unsigned int g, unsigned int b)`

In `r`, `g` und `b` können die Helligkeiten der drei Grundfarben rot, grün und blau angegeben werden. Nach diesem Befehle erfolgt die Ausgabe in der angegebenen Farbe. Beispiel:

```
color(255,0,255);
line(2,1,101,100);
```

12 Farbe ändern (Farbname)

– nicht bei Ghostscript-Ausgabe verfügbar –

`void ncolor(const char *t)`

`t` ist der englischsprachige Name einer Farbe (z.B. `black`). Beispiel:

```
ncolor(yellow);
line(3,1,102,100);
```

In der Datei `/etc/X11/rgb.txt` sind alle auf dem System verfügbaren Farbnamen abgelegt. Die Zahl der verfügbaren Farben ist natürlich (fast immer) größer!

13 Text ausgeben

`void text(unsigned int x, unsigned int y, const char *t)`

Der Text t wird an der Position (x,y) ausgegeben. Beispiel:

`text(20,70,"Hallo, Welt!");`

Der verwendete Zeichensatz ist ISO-8859-1.

14 Schriftart ändern

– nur bei Ghostscript-Ausgabe verfügbar –

`void setfont(const char *newfnt, double newfntsz)`

Es wird ab jetzt die Schriftart (*font*) mit dem Namen `newfnt` in der Größe `newfntsz` verwendet. Beispiel:

```
setfont("Helvetica", 16.0);
text(20,70,"Hallo, Welt!");
```

15 Graphikausgabe schließen

```
void liner_exit(void)
```

Mit diesem Befehl wird das Programm angewiesen, das Fenster zu schließen und sich zu beenden. Anwendung:

```
liner_exit();
```

16 Benutzung

Ein Beispielprogramm ist `example1.c`. Es wird wie gewohnt mit `gcc example1.c` übersetzt in die Datei `a.out`. Ein Fehler kann auftreten, wenn man vergessen hat, `liner.h` einzubinden

```
example1.c:(.text+0x22): undefined reference to ‘line’
```

Ebenso gibt es einen Fehler, wenn man die Datei `liner.h` falsch schreibt oder falsch einbindet (s.o.), Fehlermeldung:

```
example1.c:20:18: error: leiner.h: No such file or directory
```

Jetzt kann das Programm `a.out` aufgerufen werden. Dazu muss sich das Programm `liner` in einem Verzeichnis der Suchpfadliste für Programme befinden. Wenn ja, ist folgender Systembefehl erfolgreich:

```
type liner
```

Die aktuelle Suchpfadliste kann man sich mit dem folgenden Systembefehl anzeigen lassen:

```
echo ${PATH}
```

Gut ist es, wenn `liner` sich z.B. im Verzeichnis `/usr/bin` oder im Verzeichnis `/usr/local/bin` befindet. Dann kann man das Beispielprogramm mit der Eingabe von `./a.out` ausführen. Ist das Programm `liner` nicht vorhanden oder liegt es nicht im richtigen Verzeichnis, erhält man nur die Meldung:

```
sh: liner: command not found
```

In diesem Fall kann man

- entweder `liner` an die richtige Stelle kopieren (z.B. nach `/usr/local/bin`; empfehlenswert)
- oder den kompletten Namen des Verzeichnisses, in dem liner liegt, an den Suchpfad anhängen:

  ```
  PATH="${PATH}:/hier/liegt/liner"
  ```

Im Installationsverzeichnis liegen einige Beispieldateien, die man ebenfalls ausführen kann:

```
./example1
./example2
./example3
./example4
./example5
```

17 Fehler

- Einige Funktionen sind noch nicht für beide Ausgabearten fertig.
- Die Ghostscript-Ausgabe hat den Nullpunkt des Koordinatensystems unten links, die Liner-Ausgabe hat ihn oben links. Das ist noch zu vereinheitlichen.