Dr. Johannes Mayer
Axel Blumenstock
Christian Ehrhardt

Abteilung Angewandte Informationsverarbeitung

5. Dezember 2005
Blatt 7

# Allgemeine Informatik III / Systemnahe Software I (WS 2005/2006)

**Abgabetermin: 13. Dezember 2005**

## Flotter Plotter (10 Punkte)

Dieses Mal wollen wir einen Funktionsplotter schreiben.

(a) Erstellen Sie eine Funktion `plot`, die wie folgt aufgerufen werden kann:

```
plot(image, &f, a, xmin, xmax, ymin, ymax);
```

Dabei sei

| | |
|---|---|
| `image` | ein zweidimensionales int-Array. Die Größe können Sie beispielsweise durch ein `#define` festlegen. Jedes Element steht für ein Pixel, und sein Wert gibt die Farbe an – zunächst 0 für weiß und 1 für schwarz. |
| `f` | eine Funktion, die zwei Werte $x$ und $a$ vom Typ `double` akzeptiert und einen `double`-Wert zurückgibt. Über die Variable $x$ wird geplottet, und $a$ sei der Scharparameter, der für einen Plot konstant ist. Beispiel für eine Funktion, die auf dieses Schema passt, ist *sin(ax)* – ein Sinus mit variabler Frequenz. |
| `a` | der Scharparameter, der an `f` weitergereicht wird. |
| `xmin, xmax` | die Endpunkte des darzustellenden Ausschnitts der x-Achse. |
| `ymin, ymax` | die Endpunkte des darzustellenden Ausschnitts der y-Achse. |

Achsen können, müssen aber nicht dargestellt werden.

(b) Stellen Sie ferner eine Funktion

```
writePBM(FILE *fp, int image[SIZE1][SIZE2])
```

(oder ähnlich) bereit, die das Bild *image* im PBM-Raw-Format in die gegebene Datei schreibt. Dieses ist Ihnen bereits aus Blatt 2 bekannt; hier nochmal in Kürze: Nach der Kennung „P4“ folgen, jeweils durch Whitespace getrennt und als Dezimalzahl formatiert, die Breite und Höhe des Bildes in Pixeln. Nach genau einem Newline folgen die Bilddaten. Für jedes Pixel wird ein Bit abgelegt, wobei ein gesetztes Bit für ein schwarzes Pixel steht. Die Pixel werden zeilenweise von links nach rechts durchgegangen und je acht in ein Byte gepackt – vom höchst- zum niedrigstwertigen Bit. Ist die Breite des Bildes kein Vielfaches von acht, wird das jeweils letzte Bytes einer Pixelzeile mit „don't-care“-Bits (beliebigen Wertes) aufgefüllt. Die ins PBM hinauszuschreibende Zeichenfolge enthält aber keine Newlines an den Zeilenenden!

(c) Verwenden Sie nun beide Funktionen, um in einer weiteren Routine Plots mit variierendem Scharparameter in jeweils eine Datei zu schreiben. Wenn Sie die Dateien mit fortlaufenden Nummern benennen, so dass die lexikalische Reihenfolge der numerischen entspricht (also beispielsweise `plot0000.pbm`, `plot0001.pbm`, ...), können Sie mit dem Kommando

```
convert -delay 10 -loop 0 *.pbm animate.gif
```

in der Shell diese einzelnen Plots zu einem animierten GIF zusammenfügen, das Sie wiederum mit einem Browser oder `xanim` betrachten können.

(d) Wir wollen nun `convert` zusätzlich nutzen, um etwas Text, insbesondere den Scharparameter, zu den Grafiken hinzuzufügen. Zu diesem Zweck rufen wir es aus unserem Programm für jedes Einzelbild auf, und zwar mit der Funktion

```
popen(command, "w");
```

Es gibt wie `fopen()` (siehe unten) einen Dateizeiger zurück, der genauso verwendbar ist, aber mit `pclose()` geschlossen werden muss. Dieser Ausgabestrom wird als Standardeingabe an das gegebene *command* gerichtet, das wie in einer Shell interpretiert wird.

Für uns kommt hier das Kommando

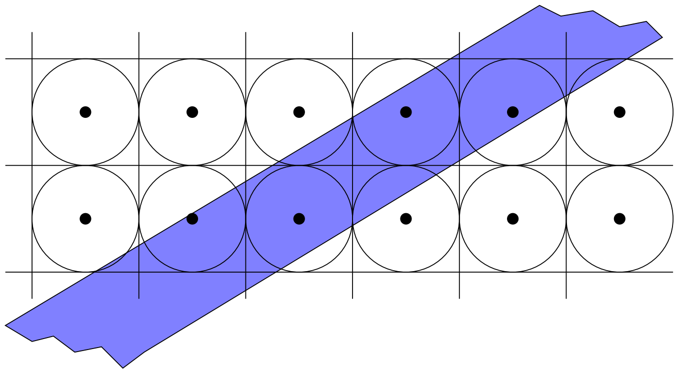
```
convert [OPTIONEN] - Ausgabedatei
```

in Frage. Das „-“ weist convert an, statt aus einer Datei von der Standardeingabe zu lesen. Die große Zahl der Optionen entnehmen Sie der Manualseite, die für uns nützlichsten sind (mit „-“!)

| | |
|---|---|
| `-font helvetica` | wählt helvetica als Font; |
| `-pen` *`color`* | setzt die angegebene Farbe (z.B. „red“); |
| `-draw 'text` *`x,y`* `"..."'` | druckt den bei „...“ angegebenen Text an Position ($x$,$y$) ins Bild. Enthält der Text besondere Zeichen, muss er in doppelte Anführungsstriche gesetzt werden. |

(e) *(Zusatzaufgabe, 5 Punkte)* Damit die Kurven nun auch noch richtig schön erscheinen, implementieren Sie schließlich noch Anti-Aliasing. Als Übergabeformat der dabei entstehenden Grauwertbilder können Sie das in Blatt 4 bereits verwendete PGM-Raw-Format nutzen. (Hier gilt: Jedes Pixel wird durch ein Byte abgelegt, wobei 0 für ein schwarzes Pixel steht.)

Für das Anti-Aliasing sind verschiedene Vorgehensweisen denkbar. Dabei müssen Sie für den Zeichenstift jeweils eine gewisse Breite, beispielsweise ein Pixel, annehmen.



- Möglichkeit 1: Zeichnen Sie die Kurve als viele aneinandergesetzte Segmente gerader Linien. Für jedes Segment können Sie für die berührten Pixel einen der überlappten Fläche entsprechenden Grauwert errechnen.
- Möglichkeit 2: Gehen Sie in jeder Pixelspalte vom Kurvenpunkt jeweils ein paar Pixel nach oben und unten und berechnen Sie jeweils den Abstand zur Tangenten an die Kurve. Wenn Sie die Pixel als kreisrund ansehen, ergibt sich ihr Überdeckungsgrad (und damit der Grauwert) als Funktion des Abstandes. Hinweis: $\int 2\sqrt{r^2 - x^2}\,\mathrm{d}x = x\sqrt{r^2 - x^2} + r^2 \cdot \arcsin \frac{x}{r}$

Einige nützliche Bibliotheksfunktionen:

- `sprintf(string, format, ...)` und `snprintf(string, size, format, ...)` erlauben wie `printf` eine formatierte Ausgabe, nur dass das Ergebnis nicht in nach stdout, sondern in die Zeichenkette *string* geschrieben wird.
- `fopen(path, "w")` öffnet die durch *path* bezeichnete Datei zum Schreiben und gibt einen `FILE *`-Zeiger zurück, der in Funktionen wie `fputc()` und `fprintf()` verwendet werden kann.
- `fclose(file)` schließt eine zuvor mit `fopen()` geöffnete Datei.
- `unlink(path)` „löscht“ die mit *path* bezeichnete Datei.

**Viel Erfolg!**