Programmieren C: Schwerere "Weihnachtsbeispiele"

Klaus Kusche

Für die guten Schüler habe ich zwei ein bisschen größere Grafik-Programme vorbereitet:

- Die **Lissajou'schen Kurven**:
  Sie enthalten relativ viel Mathematik, sind aber sonst ganz kurz und einfach:
  Sie benötigen keine Arrays und keine komplizierten Programm-Strukturen.
- **Game of Life**: Ohne Mathematik, aber deutlich länger und mit jeder Menge Arrays.

Für die Grafik verwenden wir die SDL, die du auf meinen Webseiten
samt Installationsanleitung zum Download findest.

Lege dein Programm laut Anleitung im DevCpp als Projekt an, nicht als einzelnen C-File!

Welche Funktionen wie aufgerufen werden müssen, um etwas graphisch anzuzeigen,
entnimmst du den Kommentaren zu den von mir bereitgestellten Funktionen
in **`sdlinterf.h`** und meinen beiden Beispiel-Programmen **`demo1.c`** und **`demo2.c`**.

Du kannst die Fenstergröße **`SDL_X_SIZE`** und **`SDL_Y_SIZE`** in **`sdlinterf.h`** an deinen Computer anpassen, wenn das Fenster für deine Grafik-Auflösung zu klein / zu groß ist.

1.) Lissajou'sche Kurven

Die Lissajou'schen Kurven kommen eigentlich aus der Analog-Elektronik:
Das sind die kronenförmigen, geschwungenen Linien, die im Oszilloskop entstehen,
wenn man sowohl an die x-Achse als auch an die y-Achse eine Sinusschwingung legt,
und die Frequenzen der beiden Schwingungen verschiedene ganzzahlige Vielfache
derselben Grundfrequenz sind (z.B. x-Achse 400 Hz = 4 * 100, y-Achse 300 Hz = 3 * 100).
Die Form der Kurve hängt nicht von den absoluten Frequenzen (der Grundfrequenz) ab,
sondern nur von ihrem *Frequenzverhältnis* relativ zueinander (den beiden
Multiplikationsfaktoren) und ihrer *Phasenlage* (Verschiebung) zueinander.

Wie zeichnet man so eine Kurve?

Ganz einfach: *Punkt für Punkt*, einen nach dem anderen. Wir brauchen also eine Schleife.

Um einmal die ganze Kurve zu zeichnen, muss die Grundschwingung *eine*
komplette Sinus-Welle durchlaufen. Die Funktion **`sin`** hat als Argument einen Winkel,
und für eine komplette Sinus-Welle muss der Winkel von 0 bis 360 Grad gehen.
Winkel werden aber am Computer immer in *Radiant* gemessen und nicht in Grad:
360 Grad entsprechen $2*\pi$ Radiant. Unsere Schleife braucht also
eine **`double`**-*Zählvariable*, und die muss von 0 bis $2*\pi$ zählen.

Hinweis: **`sin`** kommt aus **`math.h`**, **`M_PI`** für π auch.

Jetzt fragt sich noch mit welcher *Schrittweite* wir zählen. Da pro Schritt ein Bildpunkt
gezeichnet wird, stellt man das am besten durch "scharfes Hinsehen" ein:

- Ist die Schrittweite zu groß, dann sind die Punkte zu weit auseinander,
  und es entstehen zwischen den einzelnen Punkten Löcher in der Kurve.

- Ist die Schrittweite zu klein, liegen die Punkte so dicht, dass viele Bildpunkte mehrfach übereinander gezeichnet werden. Das schadet zwar nichts, zu viele Punkte auszurechnen verschwendet aber unnötig Rechenzeit.
- Bei 800 * 600 Pixel hat sich für die einfache Lissajou-Kurve (x- und y-Frequenz gleich Grundfrequenz, daraus ergibt sich ein Kreis oder eine Ellipse) ein Wert von 1/500 als Schrittweite bewährt. Bei höherer Auflösung bzw. größerem Fenster braucht man entsprechend mehr Bildpunkte, d.h. kleinere Schrittweite (z.B. 1/800).
- Ist die x- oder y-Frequenz ein Vielfaches der Grundfrequenz, wird die Kurve länger, und es müssen entsprechend mehr Punkte gezeichnet werden: Die Schrittweite muss daher *kleiner* sein. Wenn *f* der *größere* der beiden Frequenz-Faktoren ist, so braucht man eine Schrittweite von 1/(*f* * 500)
  (achte darauf, dass das eine *Gleitkomma-Division* sein muss, denn bei einer ganzzahligen Division käme immer 0 heraus).
  Am besten rechnest du die Schrittweite am Anfang des Programmes aus und speicherst sie in einer eigenen Variablen.

Wie zeichnet man einen einzelnen Punkt der Kurve?

Wir haben einen Winkel *winkel* der Grundschwingung (die Zählvariable der soeben besprochenen Schleife für den Winkel-Wert), einen Phasenwinkel *phase* (siehe unten) und (aus der Eingabe) die beiden ganzzahligen Faktoren *xfaktor* und *yfaktor*, die angeben, welche Vielfachen der Grundfrequenz die x- und die y-Frequenzen sind.

Der *Sinus-Wert* für die x- und y-Richtung berechnet sich daraus wie folgt:

$x = \sin(winkel * xfaktor)$ und $y = \sin(winkel * yfaktor + phase)$

Das ergibt Werte zwischen -1 und 1, aus denen wir noch *Pixelkoordinaten* zwischen 0 und (SDL_X_SIZE – 1) bzw. (SDL_Y_SIZE – 1) machen müssen (oder besser zwischen 10 und (SDL_X_SIZE – 10), damit die Kurve nicht ganz am Fensterrand klebt). Das geht mit

$x_{pixel} =$ SDL_X_SIZE / 2 + x * (SDL_X_SIZE / 2 – 10) (und in selber Art für y)

und an diese Stelle kommt der Bildpunkt für die aktuellen Winkelwerte.

Warum bewegt sich die Kurve noch nicht?

Am Oszilloskop entsteht die Bewegung dadurch, dass die Frequenzen eben nicht exakt ganzzahlige Vielfache sind: Durch kleine Fehler in den Frequenzen ändert sich die *Phasenverschiebung* (das ist der Unterschied zwischen den beiden Winkeln, d.h. an welcher Stelle der Sinuswelle ist y, wenn die Sinuswelle von x im Nulldurchgang ist) zwischen x- und y-Schwingung ganz langsam, und das lässt die Kurve rotieren.

Am Computer können wir die schon gezeichnete Kurve nicht mehr verschieben, sondern wir müssen sie mit leicht geänderter Phasenverschiebung immer wieder neu zeichnen.

Wir brauchen also ganz außen noch eine Schleife für die Phase. Das ist auch eine Schleife mit einer **`double`**-*Zählvariable*, eben dem Unterschied zwischen den beiden Winkeln. Dieser Unterschied fängt bei 0 an und wird jedesmal ganz wenig erhöht (die Schrittweite wird eingegeben, je nachdem, wie schnell die Kurve rotieren soll: Zwischen 0,0001 und 0,01 wirkt gut). Diese Schleife läuft endlos bis wir das Programm abbrechen.

In dieser Schleife passiert jedesmal folgendes:

- Den Inhalt des Grafikfensters löschen.
- Die ganze Kurve wie oben beschrieben komplett neu zeichnen (mit der Schleife über den Winkel).
- Das Grafikfenster aktualisieren.

Wie wird mein Programm aufgerufen?

Mit zwei positiven ganzen Zahlen, die angeben, welches Vielfache der Grundfrequenz die x- und y-Schwingung ist, und einer Kommazahl, die angibt, wie schnell die Kurve rotiert (das ist die Schrittweite der Phasen-Schleife, siehe oben).

2.) "Game of Life"

Das "Game of Life" ist ein beliebtes theoretisches Spiel bei Informatikern, Mathematikern usw.. Es soll die Entwicklung der "Bevölkerung" einer primitiven Zellkultur simulieren. Die "Natur" ist stark vereinfacht:

- Die gesamte besiedelbare Fläche ist in ein rechteckiges (zweidimensionales) Raster eingeteilt, jedes Feld entspricht dem Platz für *ein* Lebewesen und ist entweder lebendig oder tot.
- Das Leben in jedem Feld wird Generation für Generation betrachtet und hängt nur von den acht Nachbarfeldern ab (links, rechts, oben, unten und die vier diagonalen Nachbarn):
    - Auf einem *toten* Feld, das *genau 3 lebende* Nachbarn hat, entsteht in der nächsten Generation Leben: Das sind die optimalen Lebens-Bedingungen.
    - Ein *lebendes* Feld lebt auch in der nächsten Generation weiter, wenn es *genau 2 oder 3 lebende* Nachbarn hat: Bei mehr stirbt es an Überbevölkerung, bei weniger an Einsamkeit.
- Da auf diese Weise berechnete Bevölkerungen normalerweise nach ein paar hundert Generationen stabil werden (d.h. es ändert sich nichts mehr), lassen wir noch neues Leben durch *zufällige Mutation* entstehen: In jeder Generation entsteht in einer toten Zelle mit einer Wahrscheinlichkeit von 1/*n* *neues Leben*. Schon eine Mutationsrate von 1/100000 reicht aus, um ein bisschen Bewegung in die Landschaft zu bringen.

In unserem Programm soll jeder Bildpunkt am Schirm genau einer Zelle entsprechen. Unser "Land" hat also 800 * 600 Felder. Neues Leben wird gelb angezeigt, überlebendes Leben wird rot angezeigt, tote Zellen bleiben schwarz.

Aufgerufen wird unser Programm mit dem *Mutationsfaktor* *n* (wenn *n* gleich 0 ist, gibt es keine Mutationen, sonst mit Wahrscheinlichkeit 1/*n*, d.h. im Schnitt alle *n* Elemente). Das Programm läuft endlos, bis man es abbricht.

Details:

- Das Programm enthält *zwei zweidimensionale Arrays*,
  jeweils mit **SDL_X_SIZE * SDL_Y_SIZE** Elementen
  (d.h. mit zwei Indices 0 … (SDL_X_SIZE – 1) und 0 … (SDL_Y_SIZE – 1)).
  - Ein Array zeigt das *Leben pro Feld* an: 0 für tot und 1 für lebendig.
  - Im anderen Array berechnen wir in jeder Generation für jedes Feld die *Anzahl der lebenden Nachbarn*.

  Da die Variablen aller laufenden Funktionen incl. **main** in Windows maximal 0,5 bis 2 MB belegen dürfen (je nach Compiler, Einstellung, …), diese Arrays aber rund 4 MB groß sind, dürfen wir die beiden Arrays *nicht in* **main** deklarieren: Mach die Arrays global, d.h. schreibe die beiden Array-Deklarationen *vor* **main** !
- Du brauchst wieder **rand()** für die Zufallszahlen
  und am Programmanfang einmal "**srand(time(NULL));**",
  damit bei jedem Lauf andere Zufallszahlen herauskommen.
- *Zu Beginn erzeugen wir zufällig Leben im Lebens-Array*: Wir gehen dazu einzeln über alle Elemente (geschachtelte Schleifen für den x- und y-Index!). Liegt ein Element *ganz am Rand* (x oder y gleich 0 oder gleich (SDL_X_SIZE – 1) bzw. (SDL_Y_SIZE – 1)), wird es auf 0 (*tot*) gesetzt, sonst mit *50 % Wahrscheinlichkeit zufällig auf 0 oder 1*.
- Dann kommt die *Endlos-Schleife über die Generationen*:
  Pro Durchlauf wird eine Generation berechnet und angezeigt.
  Im Detail passiert jedesmal folgendes:
  - Der interne Grafikspeicher wird auf "alles schwarz" gesetzt.
  - Dann wird Element für Element (wieder zwei geschachtelte Schleifen für x und y, *ohne* die Rand-Elemente!) die *Anzahl der lebenden Nachbarn* berechnet (*Summe der 8 benachbarten Elemente* im Lebens-Array)
    und im entsprechenden Element des Anzahl-Arrays *gespeichert*.
  - Schließlich geht man das *Lebens-Array* nochmals Element für Element durch (nochmals zwei geschachtelte Schleifen für x und y, *ohne* Rand-Elemente!).
    Bei jedem Element muss man 3 Fälle unterscheiden:
    - Das Element *lebt* bisher (ist 1):
      - Es hat laut Anzahl-Array *2 oder 3 lebende* Nachbarn.
        Dann lebt es weiter: Am Lebens-Wert ändert sich nichts,
        aber wir zeigen an dieser Stelle einen *roten Punkt* an.
      - Sonst: Es *stirbt*. Der *Lebens-Wert* wird *auf 0* gesetzt,
        und angezeigt wird nichts.
    - Das Element ist *tot*, und es hat laut Anzahl-Array *3 lebende Nachbarn*, oder es hat Glück und wird (falls der Mutationsfaktor *n* größer 0 ist) *zufällig* mit einer *Wahrscheinlichkeit 1/n* durch Mutation *neu geboren*.

      Dann zeichnen wir einen *gelben Punkt* (farbtechnisch entsteht gelb aus rot plus grün) und setzen den *Lebens-Wert auf 1*.

      (Sonst müssen wir für tote Elemente nichts tun.)

- Wenn wir alle Elemente neu berechnet haben,
  aktualisieren wir die Grafik-Anzeige.

Hinweise:

- Alle Schleifen zur Berechnung der neuen Generation gehen nur von 1 bis (SDL_X_SIZE – 2) bzw. (SDL_Y_SIZE – 2): Für die Zellen ganz am Rand berechnen wir keine Summe (weil sie nicht 8 Nachbarn haben), und sie bleiben immer tot und am Bildschirm schwarz.
- Die Anzahl-der-Nachbarn-Berechnung muss für alle Elemente *vor* der Prüfung und Änderung der Lebens-Werte gemacht werden, denn die Anzahl muss noch für die *alte* Generation gerechnet werden, bevor Lebens-Werte der neuen Generation gespeichert werden. Man kann daher das Zusammenzählen der Nachbarn und das Berechnen des neuen Lebenswertes nicht in einem Schritt machen, d.h. die beiden Schleifen zum Durchlaufen der Anzahl-Elemente und der Lebens-Elemente *nicht* zu einer einzigen Schleife zusammenfassen.
- Das Berechnen der Anzahl der Nachbarn eines einzelnen Elementes ist keine Schleife, sondern eine einzige lange Rechnung: Die Lebens-Werte der 8 Nachbarn werden direkt addiert (wenn x und y die Index-Werte des aktuellen Elementes sind, mit welchen Index-Werten greift man auf die acht Nachbarn zu?).
- Wie sieht das **`if`** für die Mutationen aus? Es muss prüfen,
  ob die Mutationswahrscheinlichkeit *n* überhaupt größer 0 ist,
  und wenn ja, eine Zufallszahl erzeugen und diese so prüfen,
  dass sich mit Wahrscheinlichkeit 1/*n* "wahr" ergibt.
- Die **`if`** werden deutlich leicher und kürzer, wenn du mehrere Bedingungen mit "**`&&`**" für "und" oder "**`||`**" für "oder" in ein **`if`** schreibst!