

# AIK Programmieren 1 Übung: Überraschungsbeispiel

## Zweidimensionale iterierte Funktionssysteme

### *Klaus Kusche*

Zum Abschluss schreiben wir wieder ein grafisches Programm (ich verrate noch nicht, was dabei herauskommt). Installiere dir dazu wieder die SDL laut Anleitung und schau dir in **sdlinterf.h** an, welche Funktionen du wie aufrufen musst:

Am Anfang des Programms **sdlInit**, am Ende des Programms **sdlExit**, zum Löschen des Inhaltes des Grafikfensters **sdlSetBlack**, zum Zeichnen eines Punktes **sdlDrawPoint**, zum Anzeigen der bisher gezeichneten Punkte **sdlUpdate** und zum Warten **sdlMilliSleep**. Vergiss nicht auf das **#include "sdlinterf.h"** !

### *Teil 1: Grundberechnung*

Das Programm soll eine bestimmte Anzahl von einzelnen Bildpunkten der Reihe nach mit einer mathematischen Formel berechnen und anschließend anzeigen.

Die Idee dabei ist Folgende:

- Der erste Punkt hat immer die Koordinaten 0/0.
- Ab dem zweiten Punkt werden die Koordinaten des Punktes Nummer i aus den Koordinaten des vorigen Punktes Nummer i-1 wie folgt berechnet:  
$$x_i = c_0 * x_{i-1} + c_1 * y_{i-1} + c_2 \quad \text{und} \quad y_i = c_3 * x_{i-1} + c_4 * y_{i-1} + c_5$$
- Der Trick an der Sache ist, dass die Konstanten  $c_0$  bis  $c_5$  dabei nicht fix sind, sondern für jeden Punkt zufällig mit gewissen Wahrscheinlichkeiten aus mehreren verschiedenen Sätzen von je 6 Konstanten gewählt werden: Je nach Beispiel gibt es zwischen 2 und 8 verschiedene Möglichkeiten bzw. Gleichungen zur Berechnung des nächsten Punktes, d.h. 2 bis 8 Mal je 6 Konstanten.

Zuerst brauchen wir ein paar Strukturtypen (mach für alle **typedef**'s!).

Halte dich an die angegebene Reihenfolge der Member,

denn sonst funktioniert die Initialisierung der Daten in meinen Beispielfiles nicht!

- Einen Typ **punkt** zum Speichern eines Punktes:

Er enthält zwei double-Member **x** und **y** für die Koordinaten des Punktes:

Die Punkt-Koordinaten müssen exakt und ohne Platz-Begrenzung berechnet werden, erst beim Anzeigen werden die mathematischen Koordinaten auf **int**-Pixel-Koordinaten umgerechnet.

In Teil 3 kommen zu diesem Typ noch 3 **int**'s für die Farbe des Punktes dazu (Rot-, Grün- und Blauwert).

- Einen Typ **koeff** für die Konstanten (Koeffizienten in den Gleichungen):

Er hat vier double-Member: **wert**, **min\_wert**, **max\_wert** und **schritt**.

**wert** ist der eigentliche Wert der Konstante, die anderen drei Member

brauchen wir erst in Teil 2 zur Berechnung bewegter Bilder

(in welchem Bereich und mit welcher Schrittweite ändert sich die Konstante?).

- Einen Typ **farbinfo** zum Berechnen der Farbe des nächsten Punktes. Er wird erst in Teil 3 verwendet, muss aber jetzt schon jetzt deklariert werden, damit sich der Headerfile mit den Berechnungsdaten ohne Fehler inkludieren lässt.

Er enthält sechs int-Member: **proz\_rot**, **proz\_gruen** und **proz\_blau** (wieviel Prozent der Farbe des vorigen Punktes wird übernommen?) sowie **dazu\_rot**, **dazu\_gruen** und **dazu\_blau** (wieviel wird dann zum Farbwert noch dazu- oder weggerechnet?).

Nach den Typdeklarationen inkludierst du einen meiner Headerfiles mit den Berechnungsdaten (ich stelle einige Files für verschiedene Bilder zur Verfügung). Kopiere den File in dein Verzeichnis und inkludiere ihn mit " ", nicht mit < > !

Diese Files enthalten folgende Deklarationen:

- Konstante **GL\_ANZ**:  
Die Anzahl der Gleichungen, d.h. der verschiedenen Möglichkeiten (2 bis 8).
- Konstante **ANZAHL**:  
Die Anzahl der zu berechnenden Punkte (einige 100000).
- Konstante **ERSTER**:  
Die Nummer des ersten anzuzeigenden Punktes (0 bis 100).  
Bei einigen Bildern liegen die ersten paar Punkte außerhalb des eigentlichen Musters. Man muss sie zwar berechnen, aber lässt sie dann beim Anzeigen weg.
- Ein befülltes **int**-Array **prozent**:  
Es enthält **GL\_ANZ** viele Elemente, die bestimmen, mit welcher Wahrscheinlichkeit welche Gleichung zufällig gewählt wird (siehe unten).
- Ein zweidimensionales Array **c**, dessen Elemente die Konstanten zur Punktberechnung, also befüllte **koeff**-Strukturen sind:  
Der erste Index geht fix von 0 bis 5 und ist die Nummer der Konstante ( $c_0$  bis  $c_5$ ).  
Der zweite Index geht von 0 bis **GL\_ANZ-1** und ist die Nummer der Möglichkeit (der Gleichung), zu der diese Konstante gehört.
- Für Teil 3: Drei Konstanten **ROT**, **GRUEN** und **BLAU** für die Anfangs-Farbwerte.
- Für Teil 3: Ein Array **farbe** von **GL\_ANZ** vielen befüllten **farbinfo**-Strukturen. Wird zur Berechnung der Position eines Punktes die n-te Gleichung verwendet, so wird das n-te Element dieses Arrays zur Berechnung seiner Farbe verwendet.

Der „klassische“ Datensatz für iterierte Funktionssysteme ist in **farn.h** enthalten.

Deklariere als nächstes ein globales Array mit **ANZAHL** vielen **punkt-Strukturen** für die zu berechnenden Punkte (da dieses Array einige MB groß ist, der Platz für lokale Variablen aber auf wenige MB beschränkt ist, muss das Array global sein).

Dann kommt das Hauptprogramm:

- Initialisiere den Zufallszahlen-Generator so, dass bei jedem Programmablauf andere Zufallszahlen berechnet werden! (die Zufallszahlen beeinflussen allerdings nicht die Form des Ergebnisses: Die Form hängt nur von unseren Konstanten  $c_{i,n}$  ab, die Zufallszahlen verändern nur die Streuung der Punkte innerhalb der Form).
- Öffne das Grafikfenster (**sdInit**).

- Um die mathematischen Koordinaten der Punkte beim Anzeigen richtig auf Pixel-Koordinaten umrechnen zu können, müssen wir Buch führen, innerhalb welchen Koordinatenbereiches die berechneten Punkte liegen.

Dazu brauchen wir vier **double**-Werte  $x_{\min}$ ,  $x_{\max}$ ,  $y_{\min}$  und  $y_{\max}$ :

Jeweils ein Minimum und ein Maximum für die x-Koordinaten und die y-Koordinaten. Initialisiere die beiden Minima auf eine sehr große Zahl und die beiden Maxima auf eine sehr kleine Zahl (ich habe  $10^{300}$  und  $-10^{300}$  (**1e300** und **-1e300**) verwendet).

- Initialisiere die x- und y-Koordinate des Punktes Nummer 0 auf 0/0.
- Dann werden die restlichen Punkte (von Nummer 1 bis **ANZAHL-1**) in einer Schleife der Reihe nach berechnet und in unserem Array gespeichert.
  - Zuerst einmal müssen wir zur Berechnung des nächsten Punktes zufällig eine Gleichung auswählen, und zwar mit den durch das Array **prozent** aus dem Headerfile vorgegebenen Wahrscheinlichkeiten.
    - Berechne dazu eine Zufallszahl zwischen 0 und 99.
    - Das Array **prozent** ist aufsteigend sortiert und enthält am Ende immer 100. Suche in diesem Array das erste Element, das größer als die Zufallszahl ist.
    - Der Index n dieses Elementes ist die Nummer der Gleichung. Er wird weiter unten als Array-Index verwendet.

**Beispiel:** Der Header-File enthält 4 Gleichungen

und initialisiert das Prozent-Array mit den Werten 85, 92, 99 und 100.

- Für Zufallszahlen von 0 bis 84 wird **n** auf 0 gesetzt, d.h. es wird die erste Gleichung verwendet.

- Für Zufallszahlen von 85 bis 91 wird **n** gleich 1, für 92 bis 98 wird **n** gleich 2.

- Nur wenn die Zufallszahl 99 ist (Wahrscheinlichkeit nur 1 %), wird **n** gleich 3, d.h. die vierte Gleichung wird am seltensten verwendet.

- Dann werden die Koordinaten des Punktes Nummer i aus den Koordinaten des vorigen Punktes i-1 wie folgt berechnet:

$$x_i = c_{0,n} * x_{i-1} + c_{1,n} * y_{i-1} + c_{2,n} \quad \text{und} \quad y_i = c_{3,n} * x_{i-1} + c_{4,n} * y_{i-1} + c_{5,n}$$

**n** ist dabei die Nummer der Gleichung aus dem vorigen Punkt,

die  $c_{i,n}$  sind die **wert**-Member des Elementes [**i**][**n**] aus dem Array **c** .

- Wenn die Nummer des soeben berechneten Punktes größergleich ERSTER ist (wenn es sich also nicht um einen der unsichtbaren Punkte am Anfang handelt), dann prüfe mit 4 **if**'s, ob seine Koordinaten ein neues Minimum oder Maximum darstellen, und aktualisiere wenn nötig  $x_{\min}$ ,  $x_{\max}$ ,  $y_{\min}$  oder  $y_{\max}$  .
- Nachdem alle Punkte berechnet sind und  $x_{\min}$ ,  $x_{\max}$ ,  $y_{\min}$  und  $y_{\max}$  für das ganze Bild feststehen, müssen wir die Punkte der Reihe nach mit **sdlDrawPoint** anzeigen, und zwar nicht ab Punkt Nummer 0, sondern erst ab Punkt Nummer **ERSTER**.

Dabei müssen wir die Koordinaten so dehnen oder stauchen, dass alle Punkte genau in unser Fenster passen. Dafür sind in **sdlinterf.h** die Konstanten **SDL\_X\_SIZE** und **SDL\_Y\_SIZE** (Breite und Höhe des Grafikfensters in Pixel) definiert (du kannst sie dort auch an deinen PC oder an dein Notebook anpassen).

Die Umrechnung erfolgt so:  $x_{\text{pixel}} = ((x - x_{\min}) / (x_{\max} - x_{\min})) * (\text{SDL\_X\_SIZE} - 1)$

y wird zusätzlich gespiegelt:  $y_{\text{pixel}} = (1 - (y - y_{\min}) / (y_{\max} - y_{\min})) * (\text{SDL\_Y\_SIZE} - 1)$

Nimm als Farbe der Punkte im **sdlDrawPoint** grün (0 / 255 / 0).

- Ruf **sdlUpdate** erst auf, nachdem du alle Punkte gezeichnet hast, nicht nach jedem Punkt (jeden Punkt einzeln anzuzeigen wäre viel zu langsam)!  
Warte dann mit **sdlMilliSleep** ein paar Sekunden, bevor du **sdlExit** aufrufst und das Programm beendest.

## Teil 2: Bewegung

Das Aussehen des Ergebnisses ändert sich, wenn man die Konstanten der Gleichungen geringfügig ändert.

Wir machen daher Folgendes, um ein bewegtes Bild zu erhalten:

- Packe das gesamte Berechnen und das Anzeigen der Punkte aus Teil 1 in eine Endlosschleife.  
Verkürze die Wartezeit nach dem Anzeigen (oder lass sie ganz weg).  
Vergiss nicht, vor jedem Berechnen  $x_{\min}$ ,  $x_{\max}$ ,  $y_{\min}$  und  $y_{\max}$  wieder auf eine sehr große bzw. sehr kleine Zahl zu setzen und vor jedem Anzeigen das vorige Bild zu löschen (**sdlSetBlack**).
- Ändere nach dem Anzeigen eines jeden Bildes alle Gleichungs-Konstanten wie folgt:
  - Mach zwei geschachtelte Schleifen über alle 6 mal **GL\_ANZ** Konstanten-Strukturen unseres zweidimensionalen Gleichungs-Arrays.
  - Zähl zu jedem Konstanten-Wert **wert** seine Schrittweite **schritt** dazu.
  - Wenn **wert** > **max\_wert** oder **wert** < **min\_wert** ist (d.h. die Konstante **wert** aus dem Bereich **max\_wert** ... **min\_wert** herausgefallen ist), dann dreh das Vorzeichen von **schritt** um, damit die Konstante beim nächsten Durchlauf in die entgegengesetzte Richtung wandert.

## Teil 3: Farbe

Da wir zu jedem einzelnen Punkt seine Farbe speichern müssen, erweitern wir unsere Punkt-Struktur um drei **int**'s für den rot-, grün- und blau-Wert (0..255) des Punktes und geben diese Werte beim Zeichnen des Punktes im **sdlDrawPoint**-Aufruf statt grün als Farb-Parameter an.

Dann brauchen wir eine Hilfsfunktion, die aus einem alten Farbwert den neuen Farbwert berechnet:

- Die Funktion hat drei **int-Parameter**:  
Alten Farbwert, wieviel Prozent der alten Farbe erhalten bleiben, und um wieviel die Farbe dann noch geändert wird.  
Sie liefert einen **int** als Returnwert.
- Berechne zuerst den neuen Farbwert: (Alte Farbe \* Prozent) / 100 + Änderung  
Ist das Ergebnis größergleich 255, so ist der Returnwert 255.  
Ist es kleinergleich 0, so ist der Returnwert 0.  
Sonst wird das berechnete Ergebnis als neue Farbe zurückgegeben.

An der Stelle, wo die Koordinaten des Punktes Nummer 0 auf 0/0 gesetzt werden, setzen wir seine Farbwerte auf die Konstanten **ROT**, **GRUEN** und **BLAU**. Diese Konstanten werden im Berechnungsdaten-Headerfile deklariert.

Wir haben schon im ersten Teil der Aufgabe eine Struktur für die Farb-Berechnung deklariert, die 6 **int**-Member enthält:

Für jede der drei Farben rot, grün und blau einen Prozentwert und einen Änderungswert.

Wie ein Punkt gefärbt wird, hängt davon ab, mit welcher Gleichung er berechnet wurde. Der Daten-Headerfile legt daher ein Array namens **farbe** von solchen Strukturen an, das so viele Element hat, wie es Gleichungen gibt, und befüllt es:

Das **n**-te Element enthält die Prozent- und Änderungswerte für die Punkte, die mit der **n**-ten Gleichung gezeichnet werden.

An der Stelle, wo wir die Koordinaten des nächsten Punktes berechnen, wird auch seine Farbe berechnet. Dazu rufen wir drei Mal unsere Hilfsfunktion auf, „füttern“ sie mit dem entsprechende Rot-, Grün- oder Blauwert des vorigen Punktes und mit dem Prozentwert und dem Änderungswert für diese Farbe aus der **n**-ten Struktur des Arrays **farbe** (**n** ... Nummer der verwendeten Gleichung), und speichern das Ergebnis der Aufrufe als Rotwert, Grünwert und Blauwert des neuen Punktes.

## ***Für Interessierte***

Jede Gleichung beschreibt eine Bildtransformation: Sie bildet die gesamte Bildfläche irgendwie gedreht, gestaucht, verschoben, gespiegelt oder sonstwie verzerrt auf einen Teilbereich des Bildes ab, d.h. ist für einen Teil des Bildes bzw. der Figur verantwortlich. Das Gesamtbild wird also aus vielen verkleinerten Kopien von sich selbst zusammengesetzt, die iterierten Funktionssysteme sind daher auch eine Art Fraktale.

Am Beispiel des Farns:

- Die erste (häufigste) Gleichung zeichnet die verkleinerten Kopien oberhalb der untersten Blatt-Reihe: Sie verkleinert und neigt den Input ein bisschen und schiebt ihn ein Stückchen Richtung Farn-Spitze.
- Die zweite und dritte Gleichung zeichnen das rechte und das linke unterste Blatt: Der Input wird stark verkleinert, um knapp 90 Grad nach links oder rechts gekippt, und ganz nach unten verschoben.
- Die vierte (seltenste) Gleichung ist für den Stamm (die gerade, senkrechte Linie unten in der Mitte) zuständig: Unabhängig davon, wo der Punkt vorher war, bekommt er einen fixen x-Wert, und sein y-Wert wird stark gestaucht.