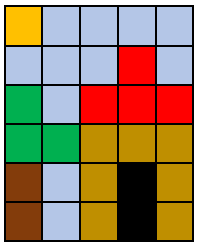
Algorithmen zur Manipulation von Bildern

Neben dem Zeichnen verschiedener Formen gehört zu den typischen Funktionen von Bildbearbeitungsprogrammen die Manipulation von Bildern, insbesondere Fotos, in Hinblick auf Helligkeit und Kontrast, das Verändern zu einem Graustufenbild, das Entfernen eines Rotstichs, das Entfernen oder Umfärben eines Bildbereichs usw. Ziel dieses Arbeitsblattes ist es, für einige dieser Funktionen geeignete Algorithmen zu entwerfen.

Codierung digitaler Fotos

Digitale Fotos sind Rastergrafiken, die aus vielen einzelnen, winzigen Quadraten, den Pixeln, bestehen. Jedes Pixel wird durch seinen RGB-Wert codiert.



(255, 192, 0)	(180, 198, 231)	(180, 198, 231)	(180, 198, 231)	(180, 198, 231)
(180, 198, 231)	(180, 198, 231)	(180, 198, 231)	(255, 0, 0)	(180, 198, 231)
(0, 176, 80)	(180, 198, 231)	(255, 0, 0)	(255, 0, 0)	(255, 0, 0)
(0, 176, 80)	(0, 176, 80)	(191, 143, 0)	(191, 143, 0)	(191, 143, 0)
(131, 60, 11)	(180, 198, 231)	(191, 143, 0)	(0, 0, 0)	(191, 143, 0)
(131, 60, 11)	(180, 198, 231)	(191, 143, 0)	(0, 0, 0)	(191, 143, 0)

Abbildung 1: Aufbau und Codierung einer Rastergrafik

Aufgabe 1: Erläutern Sie den Zusammenhang zwischen dem grob gerasterten Bild und der Zahlenmatrix in Abbildung 1.

Wie Abbildung 1 zeigt können wir uns die Codierung eines Bildes also zunächst als eine Matrix von RGB-Werten vorstellen. Als Datenstruktur zum Speichern eines Bildes würde sich daher eine zweidimensionale Reihung anbieten. Da der Speicher eines Rechners linear aufgebaut ist, stellt Processing[1] die Daten eines Bildes jedoch als eindimensionale Reihung zur Verfügung.

(255, 192, 0)	(180, 198, 231)	(180, 198, 231)	(180, 198, 231)	(180, 198, 231)	(180, 198, 231)	(180, 198, 231)	(180, 198, 231)	...

Abbildung 2: Abbilden einer zweidimensionalen Rastergrafik auf eine eindimensionale Reihung

Aufgabe 2:

a) Erläutern Sie anhand von Abbildung 2, wie das Bild aus Abbildung 1 in einer eindimensionalen Reihung codiert werden kann.

b) Berechnen Sie die Länge der eindimensionalen Reihung aus der Breite und Höhe des Bildes.

Umgang mit Bildern in Processing

Die Algorithmen vieler der eingangs aufgezählten Funktionen eines Bildbearbeitungsprogramms basieren darauf, den RGB-Wert jedes einzelnen Pixels des Bildes mithilfe geeigneter Rechenoperationen passend zu verändern. Daher müssen wir uns nicht nur anschauen, wie wir ein Bild einlesen und anzeigen, sondern auch wie wir auf die Reihung mit den RGB-Werten eines Bildes zugreifen können. Beispiel 1 zeigt ein Programm, das ein Bild im Programmfenster anzeigt und die zugehörige

[1] Die Programmierumgebung Processing wurde 2001 von Ben Fry und Casey Reas initiiert. Nähere Informationen finden Sie unter https://processing.org/

eindimensionale Reihung zur Bearbeitung bereithält. Das Bild, das geladen und bearbeitet werden soll, muss sich im Sketch-Verzeichnis in einem Unterordner namens *data* befinden. Wenn die Bilddatei über die Option *Sktech → Datei hinzufügen …* ausgewählt wird, wird dieser Unterordner automatisch angelegt und die Bilddatei dorthin kopiert. Das Bild kann in einem der Formate GIF, JEPG oder PNG verarbeitet werden.

```
1  PImage meinBild;
2
3  void setup(){
4    size(600, 800);
5    meinBild = loadImage("Foto_Meer.jpg");
6    image(meinBild, 0, 0);
7    loadPixels();
8  }
9
10 void draw(){}
```

Beispiel 1: Laden eines Bildes und der zugehörigen Reihung der Pixel in Processing

Betrachten wir den Quelltext etwas genauer. In Zeile 1 wird eine globale Variable `meinBild` vom Typ ***PImage*** definiert, in der wir später das Bild speichern, das wir anzeigen und bearbeiten möchten. Das Laden und Zuweisen des Bildes an die Variable erfolgt in der Methode *setup()* in Zeile 5 mit der Operation ***loadImage()***. Als Parameter muss der vollständige Dateiname des Bildes angegeben werden.

Wenn das Bild das gesamte Fenster ausfüllen soll, sollten für die Abmessungen des Fensters die Breite und die Höhe des Bildes gewählt werden. Im Beispiel handelt es sich um ein Bild, das 600 Pixel breit und 800 Pixel hoch ist. Die Operation ***size()*** erhält daher in Zeile 4 die entsprechenden Werte als Parameter.

In Zeile 6 sorgt die Operation ***image()*** dafür, dass das Bild im Programmfenster angezeigt wird. Als Parameter erhält die Operation die Variable, die das Bild enthält, sowie die Koordinaten der linken, oberen Ecke. Das Bild wird dann in der Originalgröße angezeigt. Wenn das Bild skaliert werden soll, können Breite und Höhe als weitere Parameter angegeben werden.

Processing stellt uns eine eindimensionale Reihung namens `pixels` zur Verfügung, in der die RGB-Werte aller Pixel eines Bildes vorliegen. Damit die Reihung mit den RGB-Werten des im Programmfenster angezeigten Bildes gefüllt wird, muss in Zeile 7 die Operation ***loadPixels()*** ausgeführt werden. Man könnte meinen, dass jeder RGB-Wert selbst als Zahlen-Reihung vorliegt. Tatsächlich verfügt Processing aber über einen speziellen Datentyp *color*, in dem der RGB-Wert als Ganzzahl gespeichert wird. Bei der Reihung `pixels` handelt es sich daher um eine Reihung vom Typ *color*. Es stehen Operationen zur Verfügung, um den Rot-, Grün, Blau- und den Alphawert auszulesen. Diese schauen wir uns im nächsten Abschnitt noch genauer an.

Die Werte in der Reihung `pixels` können nach Belieben verändert werden. Damit das Bild im Anzeigefenster entsprechend aktualisiert wird, muss die Operation *updatePixels()* ausgeführt und die Methode *draw()* im Programm enthalten sein.

Veränderungen im Fenster, z. B. das Zeichnen von Formen, werden nicht automatisch in die Reihung `pixels` übernommen, sondern erst, wenn erneut die Operation *loadPixels()* ausgeführt wird.

Aufgabe 3: Erstellen Sie nach der Vorlage in Beispiel 1 ein Processing-Programm, das ein Foto Ihrer Wahl im Programmfenster ausgibt. Kopieren Sie die Bilddatei mithilfe der Option *Sktech → Datei hinzufügen …* zunächst in das Sketch-Verzeichnis.

Aufgabe 4: Ordnen Sie den Operationen in Abbildung 3 ihre Bedeutung zu, indem Sie die Kästen links jeweils mit dem passenden Kasten rechts verbinden.

`meinBild = loadImage("pic.jpg");`	Objektvariable vom Typ *PImage* erzeugen
`updatePixels();`	Das Bild aus der Datei *pic.jpg* laden und in der Variablen mein Bild speichern
`image(meinBild, 0, 0);`	Das Bild aus der Variablen `meinBild` in Originalgröße anzeigen.
`pixels[0] = color(255, 0, 0);`	Das Bild aus der Variablen `meinBild` in der Größe 200 mal 400 Pixel anzeigen.
`image(meinBild, 0, 0, 200, 400);`	Die Farbwerte der Pixel aus dem Programmfenster in die Reihung `pixels` schreiben.
`PImage meinBild;`	Dem ersten Bildpunkt die Farbe Rot zuweisen.
`loadPixels();`	Die Daten aus der Reihung `pixels` im Programmfenster als Bild anzeigen.

Abbildung 3: Operationen zum Umgang mit Bildern in Processing.

Aufgabe 5: Machen Sie sich mit den Operationen zum Umgang mit Bildern weiter vertraut, indem Sie Folgendes ausprobieren:

a) Lassen Sie Ihr Bild viermal (neunmal) im Programmfenster ausgeben (s. Abbildung 4).

b) Färben Sie die obere Hälfte des Programmfensters rot, indem Sie die Farbwerte in der Reihung `pixels` entsprechend verändern.

Abbildung 4: Vierfachanzeige eines Fotos im Programfenster.

Auslesen und Verändern der Farbanteile eines Pixels

In Aufgabe 5b haben Sie einzelnen Bildpunkten in der Reihung `pixels` bereits einen Wert zugewiesen. Um ein Bild gezielt zu manipulieren, müssen wir jedoch auch in der Lage sein, den ursprünglichen Farbwert vom Datentyp *color* aus der Reihung auszulesen und daraus den Rot-, Grün- und Blauanteil zu extrahieren. Dazu stehen uns die Operationen *red()*, *green()* bzw. *blue()* zur Verfügung, die auf einen Parameter vom Typ *color* angewendet werden können und den entsprechenden Farbanteil als Zahl vom Typ *float* zurückgeben. Damit haben wir die Möglichkeit die

Farbanteile rechnerisch zu verändern und anschließend mithilfe der Operation *color()* zu einem neuen Farbwert zusammenzusetzen. Die Operation *color()* erhält als Parameter den Rot-, den Grün- und den Blauanteil. Wird der Operation *color()* nur ein Parameter übergeben, so wird dieser als Rot-, Grün- und Blauanteil verwendet, so dass der entsprechende Grauton erzeugt wird.

Außerdem haben wir bereits die Operation *updatePixels()* kennengerlernt, um die Anzeige in unserem Programmfenster zu aktualisieren. Um das Ergebnis in einer Datei abzuspeichern, können wir den Befehl *save()* verwenden. Als Parameter übergeben wir den gewünschten Dateinamen und das gewünschte Dateiformat. Zur Auswahl stehen TIFF (.tif), TARGA (.tga), JPEG (.jpg), oder PNG (.png). Die entsprechende Datei wird dann im Sketch-Verzeichnis erzeugt.

Tabelle 1 gibt einen Überblick über die Operationen, die beim Erstellen von Algorithmen zur Manipulation von Bildern hilfreich sind.

Operation	Beispiel	Bedeutung
	`color col = pixels[0];`	Speichert den Farbwert des ersten Bildpunktes in der Variablen `col` vom Typ *color*.
red`(farbe: color): float`	`float rot= red(col);`	Liefert den Rotanteil eines Farbtons vom Datentyp *color*.
green`(farbe: color): float`	`float gruen = green(pixels[0]);`	Liefert den Grünanteil eines Farbtons vom Datentyp *color*.
blue`(farbe: color): float`	`float blau = 255 - blue(col);`	Liefert den Blauanteil eines Farbtons vom Datentyp *color*.
color`(grauwert: float): color`	`color(200);`	Erzeugt den Farbwert vom Datentyp *color* zu einem Grauton.
color`(red: float, green: float, blue: float): color`	`color(rot, gruen, blau);`	Erzeugt den Farbwert vom Datentyp *color* zu einem RGB-Wert.
save`(dateiname: String)`	`save("neuesBild.jpg");`	Speichert die Anzeige im Programmfenster als Bilddatei ab.

Tabelle 1: Operationen zum Umgang mit Farbwerten vom Datentyp color und Abspeichern von Bildern.

Aufgabe 6: Erstellen Sie ein Processing-Programm, das zu Beginn ein Bild lädt und anzeigt.

- Bei Drücken der Taste *i* sollen die RGB-Werte aller Bildpunkte invertiert werden (s. Abbildung 5).
- Bei Drücken der Taste *s* soll das Ergebnis abgespeichert werden.
- Bei Drücken der Taste *r* soll das Original des Bildes wieder angezeigt werden.

Abbildung 5: Original und Bild mit invertierten Farben

Hinweis: Einen RGB-Wert zu invertieren bedeutet, jeden Farbanteil durch die Differenz zwischen 255 und dem ursprünglichen Farbanteil zu ersetzen. Aus dem RGB-Wert (30, 100, 255) wird so der invertierte RGB-Wert (225, 155, 0).

Rekonstruktion ausgewählter Funktionen eines Bildbearbeitungsprogramms

Digitalkameras erzeugen in aller Regel Farbfotos. Um eine andere Wirkung zu erzielen, möchte man die Farbinformation manchmal auf Grauwerte reduzieren.

Zu einem Farbfoto das passende Graustufenbild wie in Abbildung 6 zu erzeugen, bedeutet im Prinzip, jeden einzelnen Bildpunkt auf seine Helligkeit zu reduzieren und den zur Helligkeit passenden Grauton auszuwählen. Die Helligkeit eines Farbtons lässt sich unterschiedlich definieren.

Abbildung 6: Umwandlung eines Farbbildes in ein Graustufenbild

Aufgabe 7:

a) Diskutieren Sie unterschiedliche Möglichkeiten, die Helligkeit eines Bildpunktes anhand des RGB-Wertes zu definieren. Ergänzend können Sie anschließend die Vorschläge in der Datei *AB1_Aufg7_Hinweis* betrachten.

b) Ergänzen Sie in Ihrem Programm aus Aufgabe 6 die Möglichkeit, das Bild in ein Graustufenbild umzuwandeln. Gehen Sie dabei ggf. arbeitsteilig vor, um verschiedene Definitionen bzw. Berechnungen der Helligkeit zu implementieren und die Ergebnisse zu vergleichen.

Aufgabe 8: Je mehr Optionen das Programm zur Bildbearbeitung anbietet, desto mehr Varianten des Bildes möchte man eventuell abspeichern. Wird der Operation *save()* ein statischer Dateiname übergeben, wird das Bild jedoch bei jedem Speichervorgang überschrieben. Dieses Problem lässt sich lösen, indem in den Dateinamen eine variable Komponente eingebaut wird. Eine Möglichkeit wäre einen Zähler zu ergänzen, der als Teil des Dateinamens verwendet wird und bei jedem Speichervorgang hochgezählt wird. Eine zweite Möglichkeit wäre, den Anwender jedes Mal einen Dateinamen eingeben zu lassen.

a) Diskutieren Sie Vor- und Nachteile der beiden vorgestellten Möglichkeiten. Fallen Ihnen weitere Lösungsansätze ein?

b) Implementieren Sie eine der Möglichkeiten.

Aufgabe 9: Ergänzen Sie in Ihrem Programm weitere Funktionen zur Bearbeitung eines Bildes. Hier sind einige Ideen. Ihnen fallen aber sicher auch noch andere Möglichkeiten ein!

- Verändern der Helligkeit
- Erhöhen des Kontrastes
- Entfernen eines Rotstichs
- Sepia-Effekt
- Schwarz-Weiß-Bild
- Posterisation

Hinweise: Überlegen Sie zunächst selbst, wie die RGB-Werte rechnerisch so verändert werden können, dass sich der gewünschte Effekt ergibt.
Wenn Sie keine Idee haben, welche Rechenoperationen notwendig sind, können Sie …

- … verschiedene Rechnungen ausprobieren und sich anschauen, wie sie das Bild verändern.
- … die Hinweise in den entsprechenden Dateien *AB1_Aufg9_Hinweis_...* zur Hilfe nehmen.

Das Ergebnis muss nicht perfekt sein! Für manche Funktionen eines Bildbearbeitungsprogramms werden komplexe mathematischen Funktionen verwendet, während man einen ähnlichen Effekt schon mit einfachen Rechenoperationen erreichen kann.

Aufgabe 10: Erläutern Sie, in wieweit man bei dem Programm, das Sie in Aufgabe 6 erstellt haben, von einem Grundgerüst für Aufgabe 7 und Aufgabe 9 sprechen kann. Gehen Sie dabei darauf ein, welche Gemeinsamkeiten und welche Unterschiede die Algorithmen zur Implementierung der unterschiedlichen Funktionen eines Bildbearbeitungsprogramms aufweisen.

Lizenz