

Die Mathematik der Bildverarbeitung

Reinhard Oldenburg

Angaben zur Veröffentlichung / Publication details:

Oldenburg, Reinhard. 2006. "Die Mathematik der Bildverarbeitung." In Materialien für einen realitätsbezogenen Mathematikunterricht 9: Computer-Anwendungen, edited by Jörg Meyer and Reinhard Oldenburg, 23-37. Hildesheim: Franzbecker.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under the following conditions:

Deutsches Urheberrecht

Weitere Informationen finden Sie unter: / For more information see:

<https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publizieren>



Die Mathematik der Bildverarbeitung

Reinhard OLDENBURG, Heidelberg

Mit dem Boom der digitalen Fotografie wurde auch die digitale Bildbearbeitung mit Programmen wie Photoshop, Gimp, Paintshop u.ä. eine Massenaktivität. Mit der Mathematik dahinter versteht man einen Teil unserer technisierten Welt, und mit ein paar Programmzeilen kann man auch Dinge ausprobieren, die in den fertigen Programmen unmöglich sind. In diesem Sinne werden in diesem Aufsatz die mathematischen Grundlagen der Bildbearbeitung dargestellt und gezeigt, wie sie zur Grundlage eigener Handlungen werden können.

Vorbemerkungen

Mit Bildverarbeitungsprogrammen können Kinder umgehen, sobald sie das allgemeine PC-Handling beherrschen. Das Spiel mit Bildern und Farben macht auch viel Spaß, deshalb ist es erfreulich, dass die Mathematik der Sekundarstufe I für viele Algorithmen ausreicht. Aber auch anspruchsvollere mathematische Theorien finden in der Bildverarbeitung konkrete und anschauliche Anwendungen.

Es ist unmöglich, auf beschränktem Platz vollständige Unterrichtswege in diesem riesigen Gebiet zu skizzieren. Satt dessen versteht sich dieser Aufsatz als Anregung und Materialsammlung. Im Verlauf des Textes werden wir Anbindungen an folgende Standardthemen des Mathematikunterrichts finden:

- Statistik: Mittelwert, Minimum, Maximum, Standardabweichung
- Funktionen
- Geometrische Transformationen: Translation, Skalierung, Drehung, Scherung
- Gewichtete Mittelwerte
- Matrizenrechnung

Es ist denkbar, die hier vorgestellten Inhalte im Rahmen eines Anwendungsblocks von ca. 8-10 Unterrichtsstunden im 9. oder 10. Jahrgang zu behandeln (ohne die Fouriertransformation, die sinnvoll nur in Leistungskursen behandelt werden kann). Alternativ kann man nach einer kurzen allgemeinen Einführung in die Bildbearbeitung immer wieder bei passenden Themen auf die Anwendung „Bildbearbeitung“ zurück kommen.

In den Text sind immer wieder „Aufträge“ eingestreut, die in dieser Form Schülern gestellt werden können. Sie zeigen, dass in diesem Bereich offene Arbeitsaufträge gut möglich sind. Dem Aufsatz liegen die umfangreichen Erfahrungen zu Grunde, die ich im Göttinger Schülerlabor XLAB mit diesem Thema machen konnte. Die Schülerklientel dort ist sehr inhomogen. Es kommen Klassen und Kurse verschiedener Schulformen (Schwerpunkt Gymnasium) sowie zu den Ferienkursen einzelne, besonders motivierte und oft auch begabte Schüler. Das Thema Bildverarbeitung kann dabei flexibel dem Schülerprofil angepasst werden, weil es viele Variationsmöglichkeiten gibt, die Mathematiklastigkeit, den Programmieranteil oder das allgemeine Anspruchsniveau einzustellen.

Bilder als Matrizen und Algorithmen im Pseudocode

Computerbilder werden, egal ob sie gedruckt oder auf dem Monitor dargestellt werden, aus kleinen Rechtecken - den Pixeln - unterschiedlicher Farben zusammengesetzt. Bei Graustufenbildern reicht pro Pixel eine Zahl aus, die die Helligkeit des Pixels angibt. Üblich sind die Wertebereiche von 0 (schwarz) bis 255 (weiß) o-

der von 0 (schwarz) bis 1 (weiß). Die rechteckige Anordnung dieser Werte ist den Informatikern als Feld (array), den Mathematikern als Matrix vertraut. In mathematischen Kontexten schreiben wir $B_{i,j}$ (aus dem Intervall von 0 bis 1) für den Helligkeitswert an den Pixelkoordinaten (i,j) , in Pseudocode zur Darstellung von Algorithmen steht dagegen $B[i,j]$ als Zahl zwischen 0 bis 255. Generell sind die Algorithmen so aufgebaut, dass aus einem Bild B ein neues Bild B' berechnet wird. Bildbearbeitungsalgorithmen können in praktisch allen Programmiersprachen implementiert werden. Hilfreich ist natürlich, wenn eine Bibliothek zur Verfügung steht, die übliche Bilddateien lesen kann. In Java ist dies der Fall, allerdings ist die Handhabung der relevanten Klassen für Schüler nicht ganz einfach. Ich habe deshalb den Zugriff auf Pixel-Felder in einer Klasse gekapselt, die vom Laden aus JPG-Dateien bis zur Modifikation der Pixelwerte alle wichtigen Operationen in einfacher Form anbietet. Die Dateien sind auf Wunsch per Email erhältlich.

Statt einer allgemeinen Programmiersprache kann auch ein programmierbares Computeralgebrasystem eingesetzt werden. Moderne CAS wie Maple, Mathematica und MuPAD (leider aber nicht Derive) können Bilddateien direkt als Matrizen laden. Die Algorithmen können im CAS interaktiv verändert und die berechneten Bilder sofort betrachtet werden. Die turn-around-Zeit von der Veränderung des Verfahrens bis zur Begutachtung des Ergebnisses wird damit extrem kurz. Außerdem stehen einem eine Vielzahl von zuverlässigen Operationen von Funktionen und Matrizenrechnung über Standardabweichung bis hin zur Fouriertransformation zur Verfügung.

Ein fertiges Bildverarbeitungsprogramm wie Gimp (www.gimp.org, eine weit verbreitete Freeware-Alternative zu Photoshop, Paintshop und ähnlichen Programmen) kann im Mathematikunterricht sinnvoll eingesetzt werden, z.B. um ...

- die Behandlung von Bildbearbeitungsalgorithmen zu motivieren,
- einen leicht zugänglichen Zugriff auf bestimmte Fragen der Bildbearbeitung zu geben (insbesondere Faltungen),
- die Anwendung der behandelten Themen in realen Bildverarbeitungsprogrammen zu zeigen,
- einen ungewöhnlichen Zugriff auf mathematische Objekte, z.B. Funktionen, zu ermöglichen.

Mit der Wahl eines (oder mehrerer) dieser Werkzeuge kann der Lehrer steuern, wie stark der Informatikanteil in der Unterrichtseinheit ist. Das bestimmt auch wesentlich, wie lange eine Einheit sinnvollerweise anzusetzen ist. Minimal reicht eine Stunde aus, in der man die Veränderung von Bildhelligkeit und Kontrast mit Funktionsgraphen in Gimp durchspielt. Im maximalen Fall kann das Thema als fächerübergreifendes Leitthema ein ganzes Halbjahr dominieren. Angesichts dieser Variationsmöglichkeiten stellt der Aufsatz Bausteine zusammen, die jeder Lehrer individuell kombinieren sollte.

Helligkeit und Kontrast

Es gibt viele Wege, ein Bild aufzuhellen, sprich die Pixelwerte größer zu machen. Wenn das ganze Bild gleichmäßig behandelt werden soll, unterwirft man jeden Helligkeitswert der gleichen Funktion. Beispielsweise kann man zu jedem alten Helligkeitswert einen bestimmten Wert addieren oder ihn mit einem bestimmten Faktor multiplizieren. Dabei ist darauf zu achten, dass man den zulässigen Bereich von Helligkeitswerten nicht verlässt.

Der Pseudocode für dieses Verfahren ist denkbar einfach:

```
Eingabe: Bildmatrix B, Transformationsfunktion f
for i=1..Breite von B, j=1..Hoehe von B
  if f(B[i,j])<0 then B'[i,j]:=0
  else if f(B[i,j])>255 then B'[i,j]:=255
  else B'[i,j]:=f(B[i,j])
```

Auftrag: Erstellen Sie eine Tabelle mit Funktionstermen und der zugehörigen Beschreibung der Änderungen (mit Erklärungen!).

Eine Lösung könnte z.B. so anfangen:

Funktionsterm in w	Wirkung
120	alles wird grau
$w + 20$	Aufhellung
$64 + w / 2$	Kontrastverminderung
$1,2 \cdot w - 20$	Kontrasterhöhung
$w^2 / 255$	Grautöne werden dunkler
$16 \cdot \sqrt{w}$	Grautöne werden heller

Damit haben wir Operationen, die die Veränderung von Helligkeit und Kontrast bewirken. Kann man diese Begriffe selbst aber auch inhaltlich füllen? Das ist natürlich sehr leicht: Der Mittelwert der Pixelwerte ist ein Maß für die Helligkeit und die Standardabweichung für den Kontrast (in der professionellen Bildbearbeitung wird allerdings die Varianz als Definition des Kontrasts verwendet). Wenn diese statistischen Größen für ein gegebenes Bild tatsächlich berechnet werden sollen, zeigt sich der Vorteil der Verwendung eines CAS: Die nötigen Rechnungen sind in wenigen Sekunden eingegeben und durchgeführt. Gimp allerdings liefert diese statistischen Kennzahlen auch direkt, allerdings kann man dann keine eigenen Variationen ausprobieren, zu denen es durchaus Anlass gibt: Es gibt viele verschiedene Mittelwerte, welcher stimmt mit dem subjektiven Gefühl der Helligkeit am besten überein - und warum? Das Thema lädt zu eigenem Forschen ein.

Wenn die Schüler ganz ohne Programmierung arbeiten sollen, können Anwendungen wie die folgende, vom Autor erhältliche, nützlich sein:



Abbildung 1: Das Java-Programm TermBild erlaubt die Eingabe einer Transformationsfunktion für Helligkeitswerte als Term.

Diese Anwendung zeigt Urbild und Bild (man beachte die schöne Passung der mathematischen Sprache) unter der Funktion, deren Term im Eingabefeld oben spezifiziert wird. Der Simultanaspekt der Funktion (Malle 1993) wird so sehr augenfällig.

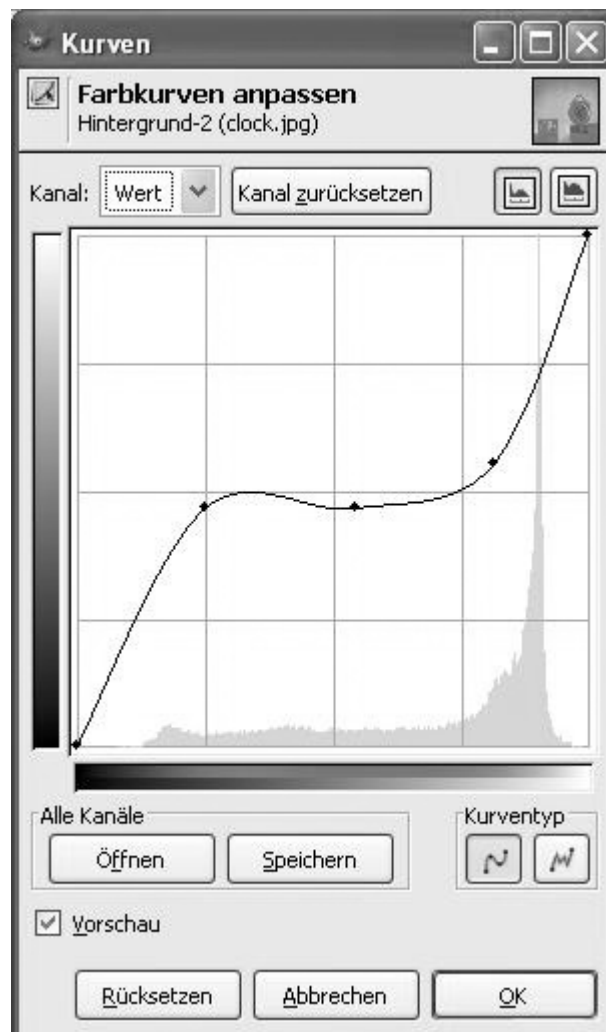


Abbildung 2: Der Kurven-Dialog in Gimp: Funktionsgraphen zwischen Koordinatenachsen aus Graustufen.

Wenn die Funktion nicht über ihren Term, sondern über ihren Graphen definiert werden soll, bietet sich die Verwendung von Gimp an, da es eine interaktive Veränderung des Funktionsgraphen erlaubt. Über die Menüpunkte Ebene \Rightarrow Farben \Rightarrow Kurven erreicht man ein Dialogfeld (Abbildung 2), in dem die Helligkeitstransformationsfunktion eingestellt werden kann. Dieses Menü kann auch zum Einstieg in die Helligkeitsmanipulation gewählt werden. Die Helligkeitswerte aller Pixel werden der gleichen Funktion unterworfen, die hier grafisch bestimmt wird. Die Rechtsachse steht für den alten Helligkeitswert eines Pixels und die Hochachse für den neuen Wert dieses Pixels. Der Funktionsgraph kann mit der Maus verschoben werden, indem man Stützpunkte auf ihn setzt und daran zieht (dazu muss der Kurventyp „weich“ gewählt sein - die Standardvorgabe). Alternativ schaltet man unten den Kurventyp auf „Freihand“, dann kann man den Graphen direkt mit der Maus zeichnen.

In jedem Falle machen diese Aktivitäten den Schülern viel Spaß, allerdings muss man darauf achten, dass die Dokumentation (z.B. durch die oben angesprochene Tabelle) nicht zu kurz kommt, denn allzu leicht geht der Spieltrieb mit einigen Schülern durch, so dass sehr schnell Bilder erzeugt werden, ohne den Zusammen-

hang mit dem Graphen noch zu verstehen. Abhilfe schaffen z.B. Aufträge, bei denen die Schüler ein vorgelegtes, verfremdetes Bild möglichst exakt reproduzieren sollen.

Transformationen

In den Bildverarbeitungsprogramme gibt es vielfältige Möglichkeiten zum Skalieren, Drehen, Verschieben und Scheren von Bildern oder Bildteilen. Eine schöne Ergänzung bilden die Möglichkeiten, Abstände und Winkel zu messen (bei Gimp im Menü Werkzeuge \Rightarrow Messen bzw. in Gimp Version 2.2.x: Werkzeuge \Rightarrow Maßband). Dies eröffnet die Möglichkeit, Bild und Urbild nicht nur in Augenschein zu nehmen, sondern auch messend zu vergleichen und damit den Eigenschaften der Transformationen „experimentell“ nachzuspüren. Ein Arbeitsauftrag könnte sein, herauszufinden, bei welchen Transformationen Winkel geändert werden und bei welchen nicht. Es ist aber auch reizvoll, einige dieser Transformationen selbst zu programmieren. Recht einfach zu realisieren sind folgende Fälle:

- Translationen
- Spiegelungen
- Drehung um 90°
- Streckung (Skalierung) um Faktor 2 (dazu wird jedes Pixel durch vier gleich helle Pixel ersetzt)
- Streckung um Faktor $\frac{1}{2}$ (je vier Pixel werden durch ihren Mittelwert ersetzt)
- Scherung

In der Regel gibt es in jeder Schülergruppe etliche, die selbst herausfinden, wie man diese Abbildungen mit Schleifen und Zuweisungen umsetzt.

Auch andere Skalierungen sind nicht sehr schwierig, werden aber von den Schülern erfahrungsgemäß weniger leicht selbsttätig entdeckt, zumal es mehrere Lösungsansätze gibt. Als Beispiel sollen zwei Skalierungsalgorithmen genügen:

```
Eingabe: Bildmatrix B, Skalierungsfaktor s
for i=1..Breite von B', j=1..Hoehe von B'
  B'[i,j]:=B[s*i,s*j]
```

Zweiter Algorithmus:

```
Eingabe: Bildmatrix B, Skalierungsfaktor S
for i=1..Breite von B, j=1..Hoehe von B
  B'[i*S,j*S]:=B[i,j]
```

Schüler können diese Algorithmen vergleichen und Defizite aufdecken. Wenn man perfekt verkleinern will, muss man geeignete Mittelwerte der Pixel berechnen, die zu einem Pixel im Ergebnis beitragen - das ist gar nicht so einfach!

Bilder verzerren

Bisher wurden Pixel entweder an Ort und Stelle verändert oder mit einer Bewegung des ganzen Bildes verschoben. Der Wunsch nach Bildkorrektur führt aber auch auf andere Problemstellungen: Wenn man etwa eine Nase größer oder kleiner machen will, muss man einige Pixel lokal verschieben. Dazu gibt es extrem viele unterschiedliche Modelle zur Realisierung. Leistungsstarke Lerngruppen können hier eigene Varianten erfinden.

Wir betrachten hier die Möglichkeit, Teile des Bildes zu verschieben und zu verzerren, als wäre das Bild auf eine Gummihaut gemalt. Graphisch kann man das durch lokal variierende Verschiebungspfeile veranschaulichen:

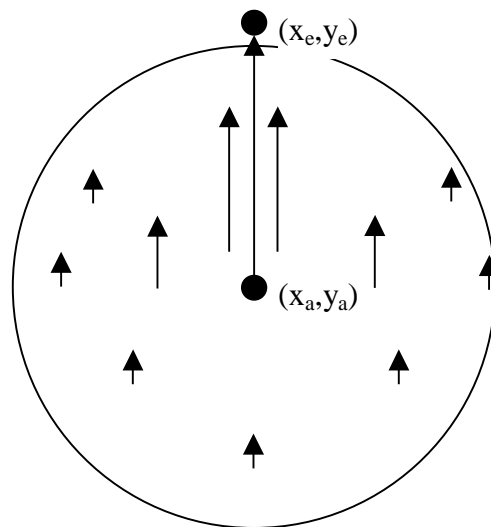


Abbildung 3: Verschiebungspfeile illustrieren die Verzerrung des Bildes.

Es wird angenommen, dass der Benutzer mit der Maus das Pixel (x_a, y_a) , den Startpunkt, nach (x_e, y_e) zieht (a und e stehen für Anfang und Ende). Es sollen alle Pixel außerhalb des gepunkteten Kreises um den Startpunkt des Zuges an Ort und Stelle bleiben. Innerhalb des Kreises werden alle in die gleiche Richtung verschoben, und zwar um so weiter, je näher sie am Startpunkt liegen. Eine mögliche Realisierung zeigt der folgende Pseudocode.

```

Eingabe: Bildmatrix B,  $x_a, y_a, x_e, y_e$ 
for i=1..Breite von B, j=1..Höhe von B
     $d := \sqrt{(i-x_a)^2 + (j-y_a)^2}$ 
     $x := i - (x_e - x_a) * h(d)$ 
     $y := j - (y_e - y_a) * h(d)$ 
     $B'[i, j] := B[x, y]$ 
Ausgabe: B'

```

Dabei ist h eine Funktion, die die Reichweite der Verschiebung bestimmt, z.B.

$$h(d) = \left\{ \begin{array}{ll} 1 - \frac{|d|}{40} & \text{falls } d < 40 \\ 0 & \text{sonst} \end{array} \right\}.$$

Kernidee des Algorithmus ist eine Schleife, die über das Bild (nicht etwa über das Urbild - sonst läuft man nämlich Gefahr, bestimmte Pixel im Bild gar nicht zu setzen) läuft. Zu jedem Pixel (i, j) im Bild wird ein passendes Pixel (x, y) im Urbild bestimmt, von dem der Helligkeitswert geholt wird. Die Verschiebung, also die Abweichung von (x, y) von (i, j) , ist umso größer, je näher man am Zentrum der Verschiebung ist (Abstand d), weil die Funktion h dort große Werte liefert. Die Ergebnisse können zwar nicht mit Dalí mithalten, aber trotzdem macht das Programm viel Spaß.



Abbildung 4: Eine Uhr mit Maus und Mathematik verzerrt.

Bilder kombinieren

In der Werbefotografie begegnen uns ständig Kombinationen von Bildern, die getrennt aufgenommen wurden, und zu einem Kunstwerk verschmolzen wurden. Das ganze Gebiet der Photomontagen hat viel mit Kunst und auch etwas mit Mathematik zu tun. Hier sollen nur einige einfache, aber trotzdem schöne Verfahren zur Sprache kommen.

Wenn man mehrere Bilder der gleichen Größe hat, kann man damit rechnen: Alle Operationen werden für alle Helligkeitswerte an den gleichen Stellen der verschiedenen Bilder durchgeführt. Dies entspricht der komponentenweisen Addition und Subtraktion von Vektoren und Matrizen. Überhaupt kann man alle Matrixoperationen wie z.B. $A + B$, $A - B$, $s \cdot A + t \cdot B$, $A \cdot B$ usw. in bestimmten Situationen sinnvoll anwenden. Damit wird die Matrizenrechnung zu einer Rechnung mit Bildern.

Der Pseudocode zur gewichteten Überlagerung zweier Bilder lautet:

```
Eingabe: Bildmatrizen A,B gleicher Größe, Gewichte s,t
for i=1..Breite von B, j=1..Höhe von B
    B'[i,j]:=s*A[i,j]+t*B[i,j]
Ausgabe: B'
```

Die Helligkeit bleibt im Rahmen der alten Bilder, wenn man $s + t = 1$ wählt, also einen gewichteten Mittelwert bildet.

Einige Anwendungen:

- In Gimp und Co. kann man Ebenen transparent übereinander legen, um Motive zu kombinieren.
- Zum Verständnis der Transformationen des letzten Abschnitts ist es geschickt, Urbild und Bild übereinander zu legen, so dass man die Wirkung der Transformation im direkten Vergleich sehen kann. Dies erreicht man durch die obige Überlagerung mit $s = t = \frac{1}{2}$.
- Die Differenz zweier Bilder zeigt Unterschiede: Man nimmt (mit Stativ) eine Szene auf, verändert eine Kleinigkeit und nimmt erneut auf. Die Differenz der Bilder zeigt deutlich, wo die Änderung lag. Mit diesem Verfahren suchen Astronomen nach Supernova-Explosionen. Eine gängige Technik in der Strömungsphysik besteht darin, kleine Testpartikel in die strömende Flüssigkeit zu bringen und dann aus der Differenz zweier kurz nacheinander aufgenommener Bilder die Strömungsgeschwindigkeit zu ermitteln. Abbildung 5 zeigt das Prinzip am Beispiel zweier strömender Passanten.

- Es gibt viele Verfahren, die Bilder in Teilbilder aufteilen. Dabei wird ein gegebenes Bild B in zwei Summanden zerlegt: $B = C + D$. Beispielsweise könnte C alle Rotanteile enthalten und D den Rest. In allen gängigen Bildbearbeitungsprogrammen liefert die Auswahl mit dem „Zauberstab“ eine solche Zerlegung des Bildes. Auf einen Teil kann dann eine Funktion f angewendet werden und danach werden die Bilder wieder zusammen gesetzt: $B' = C + f(D)$.
- Bildzusammensetzungen und Transformationen kombiniert bilden die Grundlage der iterierten Funktionssysteme IFS - einer beliebten Technik zur Erzeugung (nicht nur) von Fraktalen (siehe Peitgen et al. 1998).
- Schließlich eröffnet diese Betrachtungsweise die Übertragung vieler Arbeitsweisen der linearen Algebra auf die Bildbearbeitung. Wenn etwa e_{ij} das Bild darstellt, das an der Stelle $(i; j)$ weiß und sonst überall schwarz ist, kann jedes Bild in dieser Basis entwickelt werden: $B = \sum_{i,j} B_{ij} \cdot e_{ij}$. Abstrakt aber nützlich, wie der Abschnitt zur Fouriertransformation zeigen wird.



Abbildung 5: Ich habe kurz nacheinander zwei Bilder A und B aus meinem Bürofenster aufgenommen. Abgebildet ist der Betrag von $2 \cdot (A - B)$. Der Radfahrer unten im Bild ist erkennbar schneller als die Fußgängerin rechts im Bild. Die Kanten längs des Bordsteins deuten auf ein leichtes Verwackeln der Kamera hin. Die unscharfen Dinge links und rechts oben sind hohes Gras und Büsche: Es war ein windiger Tag.

Faltungen

Das Konzept der Faltung entwickelt man am einfachsten aus dem Wunsch, ein Bild weich zu zeichnen, also unschärfer zu machen. Wer (z.B. als Kurz- oder Weitsichtiger) unscharf sieht, sieht alles ganz weich: Harte Kanten werden verwaschen. Wenn man an den Strahlengang bei einer Linse denkt, wird klar, warum das so ist:

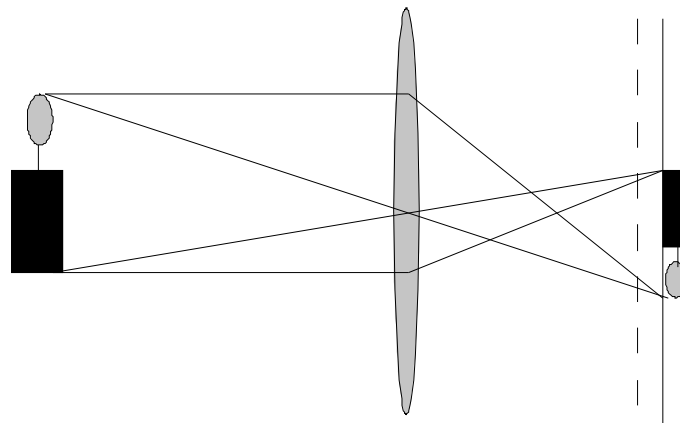


Abbildung 6: Weichzeichnung bei einer unscharfen Abbildung.

Wenn das Bild nicht scharf ist, liegt das z.B. daran, dass der Schirm (vertikale durchgezogene Linie in der Abbildung), auf dem das Bild aufgefangen wird, an der falschen Position steht. Wenn er etwas näher an der Linse steht (etwa an der durch eine vertikale gestrichelte Linie gezeigten Position), beleuchten die Lichtstrahlen, die von einem Punkt des Objekts ausgehen, einen ganzen Fleck im Bild. Das kann man - grob angenähert - auch in Software machen: Man errechnet die Helligkeit eines Punktes im Bild aus einem geeigneten Mittelwert der Helligkeit des entsprechenden Urbildpunktes und seiner Nachbarn.

Diese Idee der Einbeziehung der Nachbarpixel kann in verschiedenen Formen mathematisch gefasst und verallgemeinert werden. Die verbreitetste Form ist die sogenannte Faltung mit einer Matrix. Diese Faltungsmatrix beschreibt das Gewicht, mit dem Nachbarpixel das Ergebnis beeinflussen. Nehmen wir einmal eine konkrete

3×3-Faltungsmatrix, z.B. $A = \frac{1}{12} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 1 \end{pmatrix}$. Das Ausgangsbild sei in der folgenden

Matrix B gespeichert:

Zeile \ Spalte	0	1	2	3	4
0	23	56	122	155	160
1	56	67	127	111	188
2	55	78	144	145	217
3	65	89	177	188	234
4	67	213	211	233	233

Um jetzt z.B. den Wert des Pixels bei $(x = 3, y = 2)$ des neuen, gefalteten Bildes zu bekommen, legt man die Matrix A als Maske „über“ das Urbild, und zwar so, dass ihre mittlere Stelle über dem Pixel $(3, 2)$ liegt. Die dann überdeckten Pixel sind in der Tabelle grau unterlegt. Dann werden die an entsprechenden Stellen liegenden Zahlen aus der Faltungsmatrix und dem Ausgangsbild multipliziert und alle so erhaltenen Werte addiert:

$$\frac{1}{12} \cdot (127 \cdot 1 + 111 \cdot 1 + 188 \cdot 1 + 144 \cdot 1 + 145 \cdot 4 + 217 \cdot 1 + 177 \cdot 1 + 188 \cdot 1 + 234 \cdot 1)$$

Das Ergebnis (gerundet) ist der Wert des entsprechenden Pixels im neuen Bild. Diese Prozedur muss also für alle Pixel im Ausgangsbild wiederholt werden. Das Ergeb-

nis ist ein weichgezeichnetes Bild. Frage: Durch welche Matrix wird es stärker bzw. schwächer weichgezeichnet?

Der Faktor $1/12$ vor der Faltungsmatrix soll dafür sorgen, dass das errechnete Bild nicht zu hell wird. 12 ist gerade die Summe der Einträge der Matrix. Wenn man ein einheitlich graues Bild mit dieser Matrix faltet, erhält man durch diese Normierung genau das gleiche Bild zurück.

Als Pseudocode sieht die Anwendung einer 3×3 -Faltungsmatrix A folgendermaßen aus:

```
Eingabe: Bildmatrix B, Faltungsmatrix A
for i=1..Breite von B, j=1..Hoehe von B
  B'[i,j]:=0
  for s=-1..1, t=-1..1
    B'[i,j]:=B'[i,j]+A[s+2,t+2]*B[i+s,j+t]
Ausgabe: B'
```

Damit alle Zugriffe auf das alte Bild definiert sind, kann man dieses z.B. durch 0 fortsetzen oder periodisch schließen. Das Verfahren lässt sich analog mit größeren Faltungsmatrizen durchführen.

Man kann jetzt systematisch die Wirkung der folgenden Faltungsmatrizen untersuchen und erklären:

Matrix	Name	Wirkung
$A = \frac{1}{9} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$		Weichzeichnung
$A = \frac{1}{16} \cdot \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$	Gauß-Filter	Weichzeichnung
$A = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$	1. Prewitt-Operator	Vertikale Kanten auswählen
$A = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	2. Prewitt-Operator	Horizontale Kanten auswählen
$A = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$	1. Sobel-Operator	Vertikale Kanten auswählen
$A = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$	2. Sobel-Operator	Horizontale Kanten auswählen
$A = \frac{1}{4} \cdot \begin{pmatrix} -1 & -1 & -1 \\ -1 & 12 & -1 \\ -1 & -1 & -1 \end{pmatrix}$		Scharfzeichnung

Auftrag: Führen Sie die Tabelle auch mit eigenen Matrizen fort!

Der Gaußsche Filter nähert eine zweidimensionale Gaußverteilung an und führt zu einer recht angenehmen Weichzeichnung.

Grundsätzlich kann jede Matrix zur Faltung verwendet werden. Aber natürlich liefern längst nicht alle Varianten nützliche Filter-Effekte, interessant sind die entstehenden Bilder in der Regel aber schon. Bei vielen guten Filtern stehen Ideen der Analysis im Hintergrund. Beispielsweise kann die partielle Ableitung in x-Richtung genähert werden durch die Differenzen $B_{i,j} - B_{i-1,j}$. Die zugehörige Faltungsmatrix ist eigentlich „zu klein“ für eine 3x3-Matrix, man kann sie aber mit Nullen auffüllen und bekommt

$$\begin{pmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

In Gimp können bis zu 5x5 große Faltungsmatrizen unter dem Menüpunkt Filter \Rightarrow Allgemein \Rightarrow Faltungsmatrix (bei Gimp Version 2.2.x als Filter \Rightarrow Generisch \Rightarrow Faltungsmatrix) (siehe Abbildung 7) eingegeben werden. Das erlaubt ein Spielen ganz ohne Programmierung.



Abbildung 7: Gimps Dialog zur Eingabe der Faltungsmatrix

So schön und vielseitig die Faltungen sind, es bleiben doch einige Wünsche offen: Die Kanten hervorhebung etwa bevorzugt zu stark achsenparallele Kanten, und die Herstellung eines Reliefbildes scheitert, weil das Ergebnis immer zu dunkel wird. Verbesserungen versprechen aber einige Operationen, die den theoretischen Rahmen etwas erweitern, die aber nicht schwieriger zu programmieren sind. Es sind dies die folgenden

Verwandte Operationen: Die Faltungsoperatoren besitzen viele Verwandte, die nicht mehr linear sind, also nicht mehr in der obigen Form durch eine Matrix beschreiben werden können. Leider können sie nicht mit Gimp ausprobiert werden. Zwei Beispiele sind in folgender Tabelle enthalten. Viele weitere Filter findet man in der Literatur über Bildbearbeitung. Das Thema ist unerschöpflich!

Operation	Name	Wirkung
$B'_{x,y} = 128 + \frac{1}{2} \cdot (4 \cdot B_{x,y} - B_{x,y+1} - B_{x,y-1} - B_{x+1,y} - B_{x-1,y})$	Laplace-Filter	Reliefbild
$B'_{x,y} = \sqrt{(B_{x-1,y-1} - B_{x,y})^2 + (B_{x-1,y} - B_{x,y-1})^2}$	Robert-Filter	Kantenhervorhebung

Generell werden Filter oft zur Bild-Vorbereitung eingesetzt. Wenn ein Roboter selbsttätig navigieren soll, braucht er vor allem Informationen über die Grenzen der Objekte in seiner Nähe, dazu kann ein Filter verwendet werden, der Kanten hervorhebt (ist in der obigen Tabelle enthalten!). Ähnlich ist es für einen Arzt wichtig, in einem Ultraschallbild zu erkennen, wo die Grenze zwischen zwei Gewebearten verläuft: Auch das ist eine Anwendung für einen Kantenfilter!

Schärfe verbessern - ein Rückblick

Die letzte Faltungsmatrix der obigen Tabelle führt zu einer Scharfzeichnung des Bildes. Auf den ersten Blick scheint so etwas unmöglich, denn ein scharfes Bild scheint mehr Information zu enthalten als ein unscharfes. Ein genauer Blick zeigt, dass nicht die Informationsmenge erhöht wird, sondern dass die vorhandene Information pointiert wird. Wenn beispielsweise in einer monotonen Umgebung mit Helligkeit 100 ein leicht hellerer Punkt mit Helligkeit 150 sitzt, bekommt er im Ergebnisbild die Helligkeit von 250, die angrenzenden Pixel werden mit 87,5 sogar dunkler als sie waren, während weiter entfernte Pixel bei 100 bleiben. Die Wirkung besteht also effektiv in einer lokalen Kontrastvergrößerung.

Genau die gleiche Scharfrechnung passiert auch im Auge. Eine erregte Nervenzelle hemmt die benachbarten Zellen. Dies führt zu einer Betonung der Kontraste. Die globale Vergrößerung des Kontrasts kann ebenfalls zu einer besseren Erkennbarkeit des Bildes führen. Der Unterschied zwischen Kontrasterhöhung und Scharfzeichnung ist, dass bei letzterer nur lokal gearbeitet wird: Dort, wo bereits ein Helligkeitsunterschied im Bild ist, wird dieser verstärkt. Der Helligkeitsunterschied zwischen zwei nicht aneinander angrenzenden Bereichen wird aber, anders als bei globaler Kontrasterhöhung, nicht verändert.

Fouriertransformation

Die Fouriertransformation ist ein ähnlich mächtiges Werkzeug wie die Faltung. Ihre Anwendungen umfassen teilweise solche, die auch der Faltung zugänglich sind, wie z.B. das Scharfzeichnen, aber auch darüber hinausgehende Operationen wie z.B. die Bildkompression im JPEG-Dateiformat.

Grundlegend ist die oben beschriebene Möglichkeit, Bilder zu addieren und mit einem Faktor zu multiplizieren, also Linearkombinationen von Bildern herzustellen. Dadurch bekommt die Menge aller Bilder einer bestimmten Größe die Struktur eines Vektorraums (zumindest, wenn man erlaubt, dass der Wertebereich $[0, 1]$ oder $[0, 255]$ für die Pixelwerte auch verlassen werden darf). Durch die Fouriertransformation wird ein gegebenes Bild (also ein Vektor in diesem Raum) dargestellt als Linearkombination bestimmter Basisbilder. Der Raum der Pixelbilder der Größe $N \times M$ Pixel besitzt die Dimension $N \cdot M$ und wir beginnen nun mit der Auswahl eines entsprechend großen Satzes von Basisbildern. Diese werden bei der Fouriertransformation so gewählt, dass sie die unterschiedlichen Schnelligkeiten der Hell-Dunkel-Wechsel im Bild repräsentieren. Beispielsweise sind die folgenden 4×4 -Bilder zwei der 16 Fourier-Basisbilder für diese Bildgröße. Das linke Bild enthält

eine langsame Variation in y-Richtung und keine Variation in x-Richtung. Im rechten Bild dagegen variiert die Helligkeit in beiden Richtungen stark von Pixel zu Pixel:

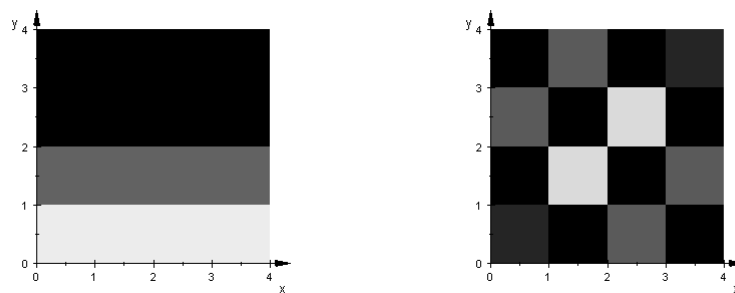


Abbildung 8: Beispiele von 4x4-Basisbildern

Man kann dem linken Bild niedrige, dem rechten Bild hohe räumliche Frequenzen zusprechen. Dies wird durch die akustische Anwendung der Fouriertransformation nahegelegt. Dort wird der Verlauf des Schalldrucks dargestellt als eine Linearkombination des Schalldrucks von reinen Tönen. Die akustische Fouriertransformation zergliedert einen Klang nach den in ihm enthaltenen reinen Tönen. Diese sind Sinus- bzw. Cosinusschwingungen und so ist es nicht erstaunlich, dass auch in unserer Anwendung die Basisbilder aus diesen Funktionen berechnet werden.

Eine genauere Inspektion zeigt sogar, dass man allein mit Cosinus-Basis-Bildern auskommen kann, wenn man diese geschickt definiert. Die daraus resultierende Variante der Fouriertransformation heißt Diskrete Cosinus-Transformation (DCT). Alle Bilder sollen im folgenden N Pixel breit und M Pixel hoch sein. Es gibt dann $N \cdot M$ reine Cosinus-Bilder $C_{u,v}$. Ihr Pixelwert an den Stellen (i, j) ist nach Definition:

$$(C_{u,v})_{i,j} = \cos\left(\frac{\pi \cdot (2i-1) \cdot v}{2N}\right) \cdot \cos\left(\frac{\pi \cdot (2j-1) \cdot u}{2M}\right), \quad \begin{array}{l} u = 0, \dots, N-1, \quad v = 0, \dots, M-1 \\ i = 1, \dots, N, \quad j = 1, \dots, M \end{array}$$

Für $M = N = 2$ ergeben sich folgende vier Basisbilder:

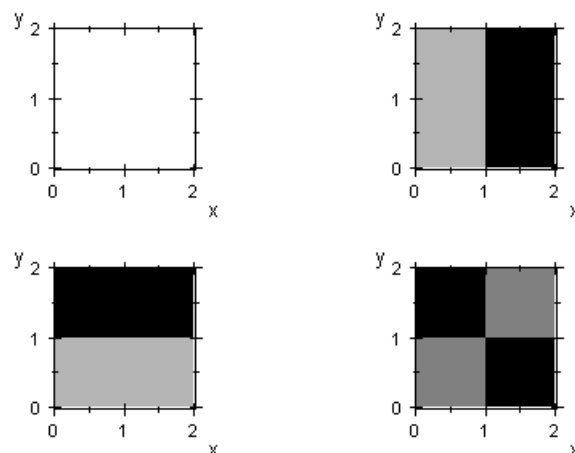


Abbildung 9: Die vier Basisbilder bei 2x2 Pixeln.

Das Basisbild C_{00} links oben ist einheitlich weiß. Es repräsentiert also die ortsunabhängige Helligkeit des Bildes, dem entspricht die Frequenz 0. Das Bild rechts daneben variiert in x-Richtung, hat dort also eine positive Frequenz und das Bild rechts unten hat in beide Richtungen positive Frequenzen.

Jedes beliebige $N \times M$ -Bild B kann als Linearkombination aus diesen Basisbildern dargestellt werden. Gemäß den Konventionen der Bildbearbeitung (siehe z.B.

Nischwitz 2004) wird diese Linearkombination in einer etwas ungewohnten Form dargestellt, die aber nur auf eine bestimmte Skalierung der Koeffizienten hinausläuft:

$$B = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} 2 \cdot k(u) \cdot k(v) \cdot b_{uv} \cdot C_{u,v}$$

Darin ist k eine fast triviale Funktion: $k(u) := \begin{cases} \frac{1}{\sqrt{2}} & u = 0 \\ 1 & u > 0 \end{cases}$.

Die Berechnung der Koeffizienten b_{uv} aus dem Bild sieht ganz ähnlich aus:

$$b_{uv} = \sum_{i=1}^N \sum_{j=0}^M \frac{2}{M \cdot N} \cdot k(u) \cdot k(v) \cdot B_{i,j} \cdot (C_{u,v})_{ij}$$

Damit ist alles zusammen gestellt, was man zur Umsetzung braucht. Man sollte sich natürlich überzeugen, dass die Schritte von B zu b und zurück zu B tatsächlich das Ausgangsbild liefern. Dann steht einem der Weg offen für Anwendungen. Dabei wird man bemerken, dass die Transformationen vierfach geschachtelte Schleifen erfordern, was bei realen Bildgrößen einen immensen Rechenaufwand bedeutet. Außer für kleine Bilder ist das Verfahren deshalb nur praktikabel, wenn man den Algorithmus der schnellen Fouriertransformation (FFT) anwendet. Diese selbst zu programmieren ist allerdings hakelig. Entweder man benutzt eine der vielen Bibliotheken für die jeweilige Programmiersprache, oder man arbeitet in einem CAS wie MuPAD. Dort steht allerdings nicht direkt die Cosinus-Transformation, sondern die Original-Fouriertransformation zur Verfügung. Diese nutzt komplexe Basisfunktionen der Art $e^{\pi i \cdot (vx + uy)} = \cos(\pi \cdot (vx + uy)) + i \cdot \sin(\pi \cdot (vx + uy))$ (abweichend vom Bisherigen ist hier i die imaginäre Einheit). Die Interpretation der Indizes u, v als Frequenzen, also als Schnelligkeit der Hell-Dunkel-Veränderung im Bildverlauf, ist aber genau so möglich.

Egal, welche Variante der Transformation man letztlich zur Verfügung hat, die Anwendungen laufen immer nach dem gleichen Muster: Originalbild \rightarrow Fourier-Transformation \rightarrow Änderung der Fourierwerte \rightarrow Rücktransformation. Die vielen unterschiedlichen Effekte, die man mit der Fouriertransformation realisieren kann, werden also durch die jeweilige Strategie zur Änderung der Fourierwerte erzielt. Eine Betonung der hohen Frequenzen, z.B. durch die Abbildung

$b'_{uv} = b_{uv} \cdot \left(1 + \frac{u+v}{N+M}\right)$, lässt Details stärker hervortreten, das Bild wirkt schärfer.

Im folgenden, mit MuPAD berechneten Beispiel, wurden die Koeffizienten b_{uv} zu 0 gesetzt, wenn u oder v größer als $1/10$ des höchsten möglichen Wertes waren. Das Bild enthält also nur noch 1% der Informationsmenge des Ausgangsbildes! Dadurch gehen Detailinformationen natürlich verloren, aber die Grobstruktur ist noch gut erkennbar. Dies ist das Grundprinzip der verlustbehafteten Bildkompression. Im JPEG-Dateiformat wird dies zusammen mit einigen weiteren Tricks angewendet.

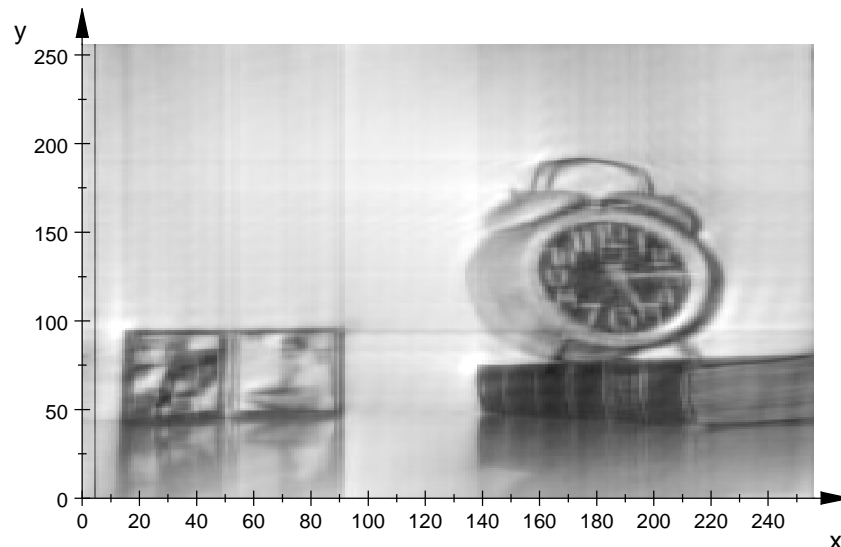


Abbildung 10: Eine einfache Fourier-Kompression auf 1% des Datenvolumens des Originals.

Was noch so alles geht

Viele interessante Themen haben wir noch gar nicht berührt. Besonders motivierend sind farbige Abbildungen. Im RGB-Modell werden diese als drei Zahlen angegeben. Nun bieten alle Bildverarbeitungsprogramme einen sogenannten Zauberstab an, mit dem man alle Pixel auswählt, die eine ähnliche Farbe haben, wie der angeklickte. Wie geht das? Nun, wenn die Farben einen dreidimensionalen Raum bilden, sollte klar sein, was der Abstand zweier Farben ist!

In Photoshop bietet der Kanalmixer die Möglichkeit, eine Abbildungsmatrix für die (r, g, b) -Vektoren anzugeben. Damit kann man Farben austauschen oder Farbstiche vermindern.

Unter dem Stichwort „Morphologische Operationen“ findet man in der Literatur einfache Algorithmen, mit denen man z.B. Kratzer aus Bildern entfernen kann. Insgesamt gesehen ist das Themenfeld noch längst nicht vollständig für den Mathematikunterricht erschlossen, aber ein Anfang ist hiermit gemacht.

Literatur

W. Burger, M. J. Burge: Digitale Bildverarbeitung, Springer 2005.

A. Janser et al.: Computergraphik und Bildverarbeitung, Vieweg 1996.

G Malle: Didaktische Probleme der elementaren Algebra, Vieweg 1993.

A. Nischwitz, P. Haberäcker: Masterkurs Computergrafik und Bildverarbeitung, Vieweg 2004.

R. Oldenburg: Funktionen, Gimp und Photoshop, MNU 2/2005.

H.O. Peitgen, H. Jürgens, D. Saupe: Bausteine des Chaos: Fraktale. Rowohlt, Hamburg 1998.

Adresse des Autors:

Prof. Dr. Reinhard Oldenburg, Pädagogische Hochschule Heidelberg, Im Neuenheimer Feld 561

E-Mail: oldenburg@ph-heidelberg.de