Formelinterpreter für die Bildverarbeitung

Autoren: Dr.-Ing. Olaf Hochmuth und Stefan Ehrhardt
Humboldt-Universität zu Berlin
Institut für Informatik

1. Einleitung

Der „Formelinterpreter für die Bildverarbeitung“ dient der Vertiefung der Kenntnisse über die Algorithmen der digitalen Bildverarbeitung. Die Studenten sollen die Möglichkeit haben, selber Programme zu schreiben und mit selbst generierten Bildern oder mit vorhandenen Bilddateien auszuprobieren. Damit die Programme kurz und überschaubar bleiben, werden hauptsächlich Punkt- und lokale Operatoren praktisch erprobt. Andere Operationen sind jedoch möglich.

2. Grundlagen

Es gibt unzählige Programme für die Verarbeitung von digitalisierten Bildern. Die wenigsten erlauben dem Nutzer einen Einblick in die Algorithmen bzw. ihre Programmierung. Mit dem „Formelinterpreter für die Bildverarbeitung“ steht ein PC-Programm zur Verfügung, das es erlaubt, Programme einzugeben, zu editieren und ihre Funktion zu überprüfen. Aus didaktischen Gründen wird die Programmiersprache PASCAL favorisiert, obgleich wohl fast jede andere Programmiersprache ebenfalls in der Lage ist, digitalisierte Bilder zu verarbeiten. Studierende, die im Grundstudium mit PASCAL, MODULA oder mit DELPHI Bekanntschaft gemacht haben, sollten kurzfristig in der Lage sein, mit dem geringen Teil der PASCAL-Sprache, die hier nur erforderlich ist, umzugehen. Ein PASCAL-Programm besteht aus einer oder mehreren Anweisungen mit folgender Syntax:

Anweisung:
- **if** Bedingung **then** Anweisung;
- **if** Bedingung **then** Anweisung **else** Anweisung;
- **while** Bedingung **do** Anweisung;
- **for** Variable:=Ausdruck **to** Ausdruck **do** Anweisung;
- **for** Variable:=Ausdruck **downto** Ausdruck **do** Anweisung;
- **begin** Anweisung; Anweisung; ... **end**;
- Zuweisung;

Bedingung:
- beliebige logische Ausdrücke mit and, or und xor, beispielsweise (a<=0) and (b=c) or (d>99.999)

Zuweisung:
- Variable:=Ausdruck; . . . . . einfache numerische Variable
- Variable[Ausdruck]:=Ausdruck; . . . . . numerische Feldvariable, Vektor
- Variable[Ausdruck, Ausdruck]:=Ausdruck; . . . . . Bildmatrix [Zeile, Spalte]

Ausdruck:
- +, -, *, /, ( ), Variablen, numerische Konstanten (123.4567), einige ausgewählte PASCAL-Funktionen

Funktionen:
- sin, cos, atan . . . . . Kreisfunktionen Sinus, Kosinus und Arcustangens
- sqr, sqrt . . . . . Quadrat und Quadratwurzel
- exp, log . . . . . Exponentialfunktion und natürlicher Logarithmus
- random . . . . . Zufallszahl zwischen 0 und 1
- random(num) . . . . . Zufallszahl zwischen 0 und num

bitweise Verknüpfung:
- and, or, xor

Zur Erläuterung der einzelnen Programme und Programmschritte können an beliebiger Stelle Kommentare eingefügt werden. Diese müssen mit den Klammern { } gekennzeichnet werden.

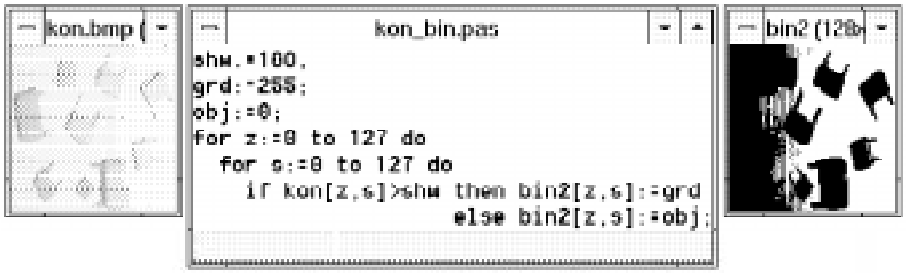
Alle Variablen im Programm, also einfache numerische Variable, Vektoren und Bildmatrizen, brauchen nicht deklariert zu werden. Diese Arbeit erledigt der Formelinterpreter. Der Studierende „deklariert“ seine Variablen, indem er sie ins Programm schreibt. Einfache numerische Variable und Vektoren werden vom Formelinterpreter nicht visualisert. Hingegen werden Bildmatrizen sofort zur Anzeige gebracht. Dies erledigt der Formelinterpreter, indem er im Programmtext alle 2fach indizierten Feldnamen ★[z,s] sucht:

- steht ein solcher Feldname rechts von einem Zuweisungsoperator :=, so werden Bilddaten erwartet,
- steht ein solcher Feldname links von einem Zuweisungsoperator :=, so werden Bilddaten erzeugt.

Damit Bilddaten vom jeweiligen Algorithmus gelesen werden können, kümmert sich der Formelinterpreter um eine Zuweisung eines Feldnamens ★[z,s] mit dem dazugehörigen gleichlautenden Dateinamen ★.BMP. Die Bilddateien sollten grauwertig sein und im BMP-Windows-Format (256 Grauwerte, unkomprimiert) vorliegen. Damit ist klar, der Formelinterpreter verarbeitet 8-Bit-Grauwertbilder, die Grauwerte liegen im Bereich von 0 bis 255. Kommt es im Verlauf der Grauwertverarbeitung zum Über- oder Unterschreiten dieser Grenzwerte, so wird dieses dem Studierenden wie folgt signalisiert. Grauwerte gößer als 255 stellen eine Übersteuerung dar und werden im Bild rot markiert. Untersteuerungen, gemeint sind alle Grauwerte kleiner als Null, werden vom Formelinterpreter blau markiert.

2.1. Punktoperatoren

Die Programmierung einer Punktoperation soll an Hand eines bekannten Beispiels aus der Literatur erfolgen [2]. Die Kondensatorszene soll beispielsweise für einen Zählvorgang binarisiert werden. Um diese Aufgabe zu lösen, wird als erstes die Bilddatei KON.BMP geladen. Die Szene ist grauwertig und liegt im BMP-Format vor (256 Grauwerte, unkomprimiert). Das Bild wird in einem Bildfenster angezeigt. Gleichzeitig wird im Titel dieses Fensters die Anzahl der Zeilen und Spalten der Bildmatrix ausgegeben.

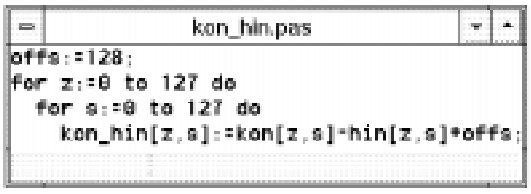


Szene mit Bauelementen auf einem scheinbar grauen Hintergrund (links), Programmfenster mit einem Algorithmus zum Binarisieren (Mitte) und Ergebnisbild der Binarisierung (rechts) [2]

In ein Programmierfenster kann der Studierende das Programm für den jeweiligen Algorithmus eingeben. Hier steht beispielsweise in der ersten Programmzeile eine Wertzuweisung: der Variablen *shw* wird ein Binarisierungsschwellwert von *180* zugewiesen. In den nächsten beiden Programmzeilen werden die im zu berechnenden Binärbild verwendeten Grauwerte vereinbart: die Hintergrundfarbe *grd* ist weiß (Grauwert 255) und die Objektfarbe *obj* ist schwarz (Grauwert 0). Dann folgen 2 Schleifen. Zum einen die Anweisung, etwas für alle Zeilen des Bildes zu tun, wobei *z* der Zeilenindex im Bild ist. Zum anderen die Anweisung, etwas für alle Spalten des Bildes zu tun, wobei *s* der Spaltenindex im Bild ist. Die Anweisung lautet: wenn der Grauwert der Bildmatrix *kon* an der Zeilenposition *z* und Spaltenposition *s* größer ist als die Variable *shw*, dann wird in die Bildmatrix *bin2* an derselben Zeilen- und Spaltenposition der Grauwert für den Bildhintergrund eingetragen. Andernfalls wird der vereinbarte Objektgrauwert eingetragen. Die so berechnete Bildmatrix *bin2* wird noch während der Berechnung angezeigt. Nun ist ein Experimentieren mit dem Wert für *shw* möglich. Jedesmal wird die Bildmatrix *bin2* aktualisiert, und der Erfolg kann sofort beobachtet werden. Es ist aber zu sehen, das die Beleuchtung der Szene so ungünstig erfolgte, daß es keine für die gesamte Szene einheitliche Binarisierungsschwelle gibt. Offensichtlich wurde die Szene inhomogen von rechts beleuchtet.

2.2. Punktoperatoren mit zwei Bildern

Ist bei einer so ungünstigen Situation eine sogenannte Leeraufnahme vorhanden, dann kann der Bildhintergrund von der Kondensatorszene subtrahiert werden. Dazu wird die Bilddatei HIN.BMP geladen, das Bild wird angezeigt und es werden wieder die Anzahl der Zeilen und Spalten dieser Bildmatrix ausgegeben. Bilder können natürlich nur dann voneinander subtrahiert werden, wenn sie die gleiche Bildgröße haben. In das Programmierfenster muß nun das Programm für die Bildsubtraktion eingeben werden. Jetzt steht in der ersten Programmzeile eine Wertzuweisung für die Variable *offs*. Ihr wird der Grauwert *128* zugewiesen. Dann folgen die 2 Schleifen: für alle Zeilen und für alle Spalten. Die eigentliche Anweisung lautet: vom Grauwert der Bildmatrix *kon* an der Zeilenposition *z* und Spaltenposition *s* soll der Grauwert der Bildmatrix *hin* bei denselben Bildkoordinaten subtrahiert werden. Zusätzlich soll die Konstante *offs* addiert werden. Das macht man immer dann, wenn bei den Algorithmen mit negativen Grauwerten zu rechnen ist - bei der Bildsubtraktion sehr wohl möglich. Das Rechenergebnis wird dann in die neue Bildmatrix *kon_hin* an derselben Zeilen- und Spaltenposition eingetragen. Die nun berechnete Bildmatrix *kon_hin* wird angezeigt. Wieder ist ein Experimentieren mit dem Wert für *offs* möglich. Jedesmal wird die berechnete Bildmatrix aktualisiert, und das Ergebnis kann sofort beobachtet werden. Nun sollte versucht werden, das Subtraktionsbild zu binarisieren. Mit den Einstellungen *schw=120* und *offs=128* gelingt es jetzt, die 9 Bauelemente gut aus der Szene heraus zu lösen. Selbstverständlich können alle Schritte dieses Algorithmus in ein einziges Programm geschrieben werden.



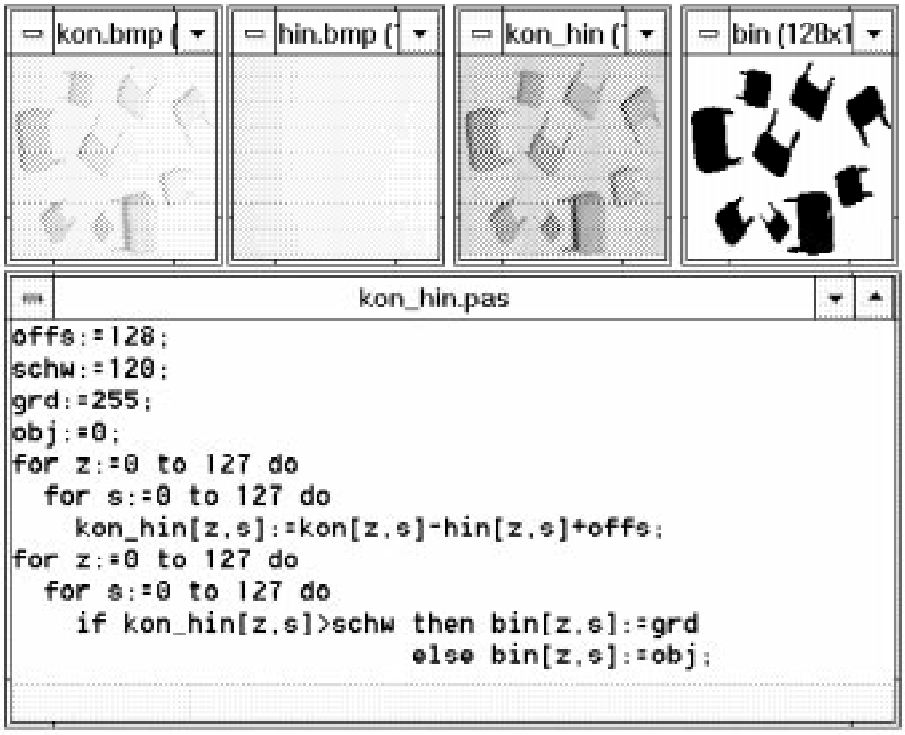
Programm für eine Bildsubtraktion

2.3. Testbilder erzeugen

Die Aufgabe, für die Demonstration lokaler Filter eine Szene mit einem Schachbrettmuster zu erzeugen, kann ebenfalls - mit einem kleinen PASCAL-Programm erledigt werden. Diesmal ist kein Bild zu laden, es soll ja erst ein Bild entstehen.

```
{Schachbrett erzeugen}
g1:=0;
g2:=255;
for z:=0 to 127 do
  for s:=0 to 127 do begin
    i:=z+s;
    while i>2 do i:=i-2;
    if i=0 then sb[z,s]:=g1;
    if i=1 then sb[z,s]:=g2;
    if i=2 then sb[z,s]:=g1;
  end;
```

Das Programm demonstriert die Verwendung der WHILE-Anweisung. Es wird ein Testbild der Größe 128x128 mit dem Namen *sb* heraus, das zwar erst einmal nur auf dem Bildschirm erscheint, dann jedoch noch als Datei SB.BMP gespeichert werden kann.



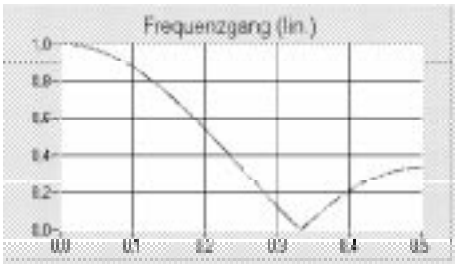
Kondensatorszene (links oben), Leeraufnahme (daneben), Subtraktionsbild (daneben) und schließlich Ergebnis der Binarisierung (rechts oben) sowie Programm im Editorfenster (unten)

2.4. Lokale Filter

Lineare lokale Operatoren werden häufig eingesetzt, um bestimmte Ortsfrequenzanteile voneinander zu trennen. Man kann zwischen Hochpässen, Tiefpässen und Kombinationen von beiden unterscheiden (Bandpässe und Bandsperren). Verfahren zum Dimensionieren der Filtervorschrift findet man in der Literatur unter den Begriffen „Entwurf nichtrekursiver 2D-Filter“ bzw. „Entwurf von 2D-FIR-Filtern“. Eine besondere Bedeutung kommt stets dem Entwurf von Tiefpässen zu, da sich der Entwurf der anderen genannten Filter auf den Tiefpaßentwurf zurückführen läßt.

Für den einfachsten Tiefpaß, das ist der, bei dem alle Grauwerte innerhalb einer 3x3-Umgebung addiert werden und dann noch durch die Anzahl der Grauwerte geteilt werden muß, läßt sich eine Übertragungfunktion bezüglich der Ortsfrequenzen angeben. Man erkennt deutlich den Tiefpaßcharakter dieses Filters - hohe Ortsfrequenzen werden abgeschwächt und tiefe Ortsfrequenzen passieren den Operator nahezu unverändert. Zum Filtern soll folgendes PASCAL-Programm dienen:

```
{ungefilterte Bildteile rot markieren}
for z:=0 to 127 do b2[z,0]:=256;  {li. Bildrand}
for z:=0 to 127 do b2[z,127]:=256;{re. Bildrand}
for s:=0 to 127 do b2[0,s]:=256;  {ob. Bildrand}
for s:=0 to 127 do b2[127,s]:=256;{un. Bildrand}
{Mittelwertfilter}
for z:=1 to 126 do
  for s:=1 to 126 do begin
    g:=b[z-1,s-1]+b[z-1,s]+b[z-1,s+1]+
       b[z  ,s-1]+b[z  ,s]+b[z  ,s+1]+
       b[z+1,s-1]+b[z+1,s]+b[z+1,s+1];
    b2[z,s]:=g/9;
  end;
```



Frequenzcharakteristik des einfachen Mittelwertfilters, die Frequenzachse ist normiert auf die Abtastortsfrequenz und stellt somit den Bereich von Null bis zur Nyquistfrequenz dar

In den ersten vier Programmzeilen werden im Ergebnisbild *b2* die Zeilen und Spalten markiert, die von einem lokalen Filter der Größe 3x3 nicht berechnet werden können. Diesen Matrixelementen wird der Grauwert *256* zugeordnet. Für 8-Bit-Grauwertbilder stellt dieser Wert eine Übersteuerung dar und wird daher im Bild *b2* vom Formelinterpreter automatisch rot markiert. Untersteuerungen, alle Grauwerte kleiner als Null, werden vom Formelinterpreter blau markiert. Dann wird das Mittelwertfilter programmiert. Zum Zentralpunkt *b[z,s]* werden seine 8 Nachbarn addiert, diese Summe wird der Variablen *g* zugewiesen. Die Summe muß noch durch 9 dividiert werden und stellt dann einen berechneten Grauwert für das Ergebnisbild *b2* dar.

3. Erfahrungen und Ergebnisse

Der Formelinterpreter wurde im Labor des Lehrstuhls Signalverarbeitung / Mustererkennung (Frau Prof. Dr. Beate Meffert) erprobt. Dazu erhielten die Studierenden Vorbereitungsmaterial, mußten zu Hause mit einem beliebigen Texteditor die

geforderten Programme erstellen und konnten dann im Labor ihre Programme testen. Der Laborversuch kam bei den Studierenden sehr gut an, und so ist schon eine beträchtliche Anzahl von Programmbeispielen zusammen gekommen (alphabetisch sortiert):

- arithmetischen Mittelwert und Standardabweichung berechnen
- Bild binarisieren
- Bild drehen (90° rechts und links)
- Bild spiegeln (vertikal, horizontal oder beides)
- Bild negieren
- Bildaveraging (2 bis 4 Bilder)
- Bilder subtrahieren
- Bildteile kopieren (transparent)
- Dynamik verändern
- Einebnen des Histogrammes
- Erosion und Dilatation (5x5)
- Farbraumwechsel RGB → CMY
- Farbraumwechsel RGB → CMYK[1]
- Farbraumwechsel RGB → HSI
- Farbraumwechsel RGB → YIQ
- Farbraumwechsel RGB → YUV
- Graukeile erzeugen
- häufigsten Grauwert berechnen
- Histogrammberechnung und -darstellung
- Hochpaß (lokal 3x3)
- Lauflängenkodierung und -dekodierung (zeilenorientiert)
- Medianfilter (5x5)
- Medianmittelwert berechnen
- Minimum und Maximum berechnen
- Schachbrett generieren
- Sobeloperator (3x3)
- Tiefpaß (lokal 3x3)
- 4:1:1-Subsampling

4. Erforderliche Technik

Der Formelinterpreter liegt in 2 Versionen vor. Das 16-Bit-Programm erfordert einen PC ab 286-Prozessor aufwärts, WINDOWS 3.x und eine Grafikkarte für mindestens 256 Graustufen. Die 32-Bit-Version läuft auf PC ab 386-Prozessor aufwärts, WIN 32s oder WINDOWS 95. Wieder ist eine gute Grafikkarte ratsam. Interessenten, die das Programm an ihren Schulen oder Hochschulen einsetzen möchten, wenden sich bitte schriftlich an:

Humboldt-Universität zu Berlin
Institut für Informatik
Dr. O. Hochmuth
Unter den Linden 6
D-10099 Berlin

5. Literatur

[1] Schmid, R.: Industrielle Bildverarbeitung. Braunschweig: Vieweg 1995
[2] Bässmann, H. und Besslich, P.W.: Bildverarbeitung Ad Oculos. Berlin: Springer 1991
[3] Klette, R. und Zamperoni, P.: Handbuch der Operatoren für die Bildverarbeitung. Braunschweig: Vieweg 1992

---

[1] Cyan: blaugrün, türkis
Magenta: purpur
Yellow: gelb
Black: schwarz