A2 Bildfilter

Ein wenig Einleitung

Bildfilter sind aus der Bildnachbearbeitung gut bekannt und dienen dazu, Effekte wie Kontrast, Schärfung, Weichzeichnen, Kantenextraktion etc. zu erzeugen. Diese Filter finden z.B. in Gimp Anwendung. Häufig werden Sie im Zusammenhang mit der Verarbeitung von Kameradaten angewendet, um Bildinformation zu reduzieren.

Bildfilter sind ein- oder zweidimensionale Arrays, die Pixel für Pixel über ein Orginalbild gelegt werden. Dabei werden für jedes Pixel des Originalbilds die Werte der benachbarten Pixel mit den Werten in der Filter Matrix multipliziert aufaddiert und anschließend die Ergebnissumme mit einem Faktor multipliziert. Das Ergebnis wird in eine neue Bildmatrix an der Position des Originalpixels eingetragen.

Durch Berücksichtigung von benachbarten Pixeln lassen sich Eigenschaften von Bildern verändern. So wird mit einem 3x3 Bildfilter, der nur die Werte 1/9 enthält, der Durchschnitt über alle benachbarten Pixel gebildet und damit die Schärfe des Bildes reduziert (Weichzeichner).

Bildfilter sollten immer eine ungerade Anzahl von Zeilen und Spalten haben, damit der Mittelpunkt eindeutig ist. Die Summe aller Werte des Filters sollte 1 ergeben, damit die Helligkeit des Bildes erhalten bleibt.

Aufgabe

Ihre Aufgabe wird es sein, eine Implementierung der abstrakten Klasse ***AbstractImageFilter*** zu schreiben, die Filtermatrizen beliebiger Breite und Höhe auf Bilder anwenden kann.

Dazu sollen Sie für die in dieser Anleitung vorgestellten Filter im ***enum Filter*** die Methoden ***getContent*** überschreiben, in denen die Filtermatrizen als int-Matrix erzeugt werden, sowie eine Implementierung für die Klasse ***AbstractFilterMatrix*** liefern, die den Zugriff auf die Inhalte und Eigenschaften der Filter ermöglicht.

Erweitern Sie das mitgelieferte und vorbereitete Programmpaket (***A2-WS1213Bildfilter***). Sie benötigen dazu ***Processing*** core Bibliothek (das Einrichten der Umgebung ist im Anhang beschrieben).

A2-WS1213Bildfilter enthält u.A. ein Verzeichnis ***data***, in dem sich das Beispielbild befindet. Wenn Sie mit anderen Bildern experimentieren wollen, dann müssen Sie diese in das ***data*** Verzeichnis kopieren.

Erläuterung des Verfahrens für die Methode *doFilter* der Klasse *AbstractImageFilter*

Im nachfolgenden Schaubild haben wir einen 3x3 Bildfilter (Filtermatrix), der nur in der Mitte ein 1 hat. Diese Matrix bildet eine Originalbild identisch in ein Ergebnisbild ab. ***fW*** und ***fH*** beschreibt Breite und Höhe dieser Matrix.

Das Originalbild hat die Breite w und die Höhe h. Um Schwierigkeiten an den Randbereichen zu umgehen, legen wir den Filter zu Beginn an der oberen linken Ecke des Originalbildes an und stoppen die Bearbeitung, wenn ***x = w-vx*** und ***y = h-vy*** sind.

Das erste, für die Ergebnismatrix zu filternde Pixel steht auf der Position ***(vx, vy)= (fW/2,fH/2)***. Wenn die Position des zu filternden Pixel ***(x,y)*** ist und die Pixel der Filtermatrix durch ***(fx,fy)*** gegeben sind, dann berechnen sich die Nachbarpixel ***(npixX, npixY)*** wie folgt***: npixX= x-vx+fx; npixY = y-vy+fy***.

Filtermatrix (float[][] filter)

fy \ fx			
	0	0	0
	0	1	0
	0	0	0

fW = Breite der Filtermatrix
fH = Höhe der Filtermatrix
faktor = Multiplikationsfaktor = 1.0

Bildmatrix original

y \ x									
	99	66	45	97	146	100	...	...	...
	55	**70**	151	160	110	70	...	...	...
	66	105	106	107	75	37	...	...	...
	45	84	114	79	56	97	...	...	...
	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...

w = Breite der Bildmatrix
h = Höhe der Bildmatrix
vx (vy) = offset für das Iterieren über die x (y) Koordinate
vx = fW/2 (ganzzahlige Division)
vy = fH/2

Wir legen die Filtermatrix über die Bildmatrix und berechnen für das Pixel, das unter dem Mittelpunkt der Filtermatrix liegt, einen neuen Wert, den wir an der Position des Pixels im Original in die Ergebnismatrix eintragen.

Berechnungsvorschrift:
wir multiplizieren die Werte der Filtermatrix mit den darunter liegenden Pixelwert der Originalmatrix,
addieren alle Teilprodukte auf und
multiplizieren diese Summe mit der Größe faktor.
Das Ergebnis tragen wir in die Ergebnismatrix ein.

Im Bsp.: **(0*99 + 0*55 + 0*66 + 0*66 + 1*70 +**
0*105 + 0*45 + 0*151 + 0*106) * 1.0 = 70

Bildmatrix ergebnis

y \ x									
	0	0	0	0	0	0	...	...	...
	0	**70**	...	...	...	...	...	...	...
	0	...	...	...	...	...	...	...	...
	0	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...

Danach verschieben wir die Filtermatrix auf der Originalmatrix um 1 Pixel nach rechts wiederholen die Berechnung solange x < w-vx.
Danach verschieben wir die Pixelmatrix um ein Pixel nach unten und wiederholen den gesamten Berechnungsvorgang solange x < w-vx und y < h-vy

Über alle diese Nachbarpixel ***(npixX,npixY)*** berechnen wir die Summe der gewichteten Farbwerte (***pixFarbe***) getrennt nach Rot-, Grün- und Blauanteil von ***pixFarbe***. (Wie sie aus einem ***PImage*** die Pixelfarbwerte extrahieren und daraus die Farbeanteile extrahieren war Gegenstand der Vorlesung).

In Pseudocode für den Rotanteil:

sumRot += rot(pixFarbe)*filter[fy][fx];

Anschließend multiplizieren wir den akkumulierten gefilterten Farbwert mit einem Faktor (***faktor***) (***sumRot= sumRot*faktor***), der die Eigenschaft, Summe der Elemente der Filtermatrix = 1 sicherstellt.

Um nicht über den Wertebereich ***[0,255]*** hinauszulaufen, und um negative Zahlen zu vermeiden, bilden wir den absoluten Wert von ***sumRot*** und bilden das Minimum aus dem absoluten Wert von ***sumRot*** und 255.

Dann setzen wir die Farbanteile wieder zu einem Farbwert zusammen (***ergebnisFarbe***)

int ergebnisFarbe = 0xff000000 | (rot<<16) | (gruen<<8) | blau;

und übertragen diesen an der Position ***(x,y)*** in die Ergebnismatrix.

Überprüfen Ihrer Implementierung

Überprüfen Sie die Korrektheit Ihrer Implementierung mit dem Bild (blumenwiese.jpg) und den nachfolgenden Filtern.

Identitätsabbildung (3x3 Filtermatrix siehe oben):

Weichzeichner (7x7 Filtermatrix) ***Filter.BLUR***

0,0, 0, 1, 0, 0,0,
0,0, 1, 1, 1, 0,0,
0,1, 1, 1, 1, 1,0,
1,1, 1, 1, 1, 1,1,
0,1, 1, 1, 1, 1,0,
0,0, 1, 1, 1, 0,0,
0,0, 0, 1, 0, 0,0, faktor = 1.0 / 25.0;

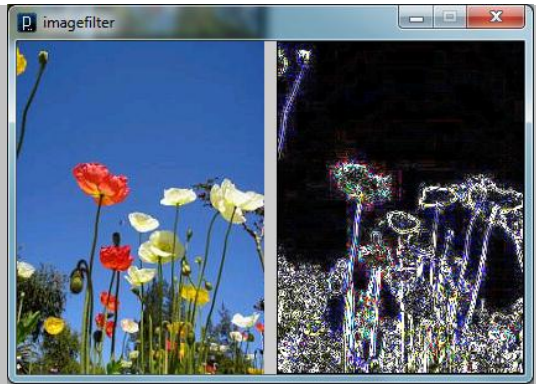
Bewegungs-Weichzeichners (9x9 Filtermatrix) ***Filter.MOTION_BLUR***

1, **0**, 0, 0, 0, 0, 0, 0, 0,
0, **1**, 0, 0, 0, 0, 0, 0, 0,
0, 0, **1**, 0, 0, 0, 0, 0, 0,
0, 0, 0, **1**, 0, 0, 0, 0, 0,
0, 0, 0, 0, **1**, 0, 0, 0, 0,
0, 0, 0, 0, 0, **1**, 0, 0, 0,
0, 0, 0, 0, 0, 0, **1**, 0, 0,
0, 0, 0, 0, 0, 0, 0, **1**, 0,
0, 0, 0, 0, 0, 0, 0, 0, **1**, faktor = 1.0 / 9.0;

Kantenfinden ***Filter.EDGE_DETECT***

-1, -1, -1
-1, 8, -1
-1, -1, -1 faktor = 1.0

1.



Anhang

2. Fügen Sie das mitgelieferte ***core.jar*** als externes jar dem Buildpath des Projektes hinzu.
3. Stellen Sie sicher, dass Sie als JRE eine 32-bit JDK Version (<= Java 1.6 eingestellt haben).
4. Starten Sie Ihr Programm immer als Applikation: Starten Sie die Klasse ***ImageFilterUIStarter***.
5. Das letzte Argument in ***PApplet.main(new String[] { "imagefilter.ImageFilterUI", "BLUR" });*** ist der Name des Filters, der auf das Orginalbild angewendet werden soll.
6. Sie können auch alle Filterergebnisse gleichzeitig darstellen, dazu müssen Sie in ***ImageFilterUI*** die Parameter im Aufruf von ***size(original.width * 2 + 10, original.height)*** anpassen und den Aufruf von ***image(aif.doFilter(original, m1, 1), original.width + 10, 0)*** vervielfachen sowie geeignet anpassen.
7. Erläuterungen zu Enums → in der nächsten Vorlesung.
8. Unter dem Suchbegriff Image Filter oder Convolution Filter finden Sie im Netz eine Reihe weiterer Filter. Wenn Sie wollen, experimentieren Sie damit.