Ludwig-Maximilians-Universität München
LFE Medieninformatik, Mobile Interaction Group
Prof. Dr. Michael Rohs
Dipl.-Inform. Sven Kratz

Sommersemester 2012
Übungsblatt 4
30.05.2012

Übung Computergrafik 2

Abgabetermin:
Die Lösung zu diesem Übungsblatt ist bis zum **06.06.2012** abzugeben.

Form der Abgabe:

- Die Übungen können in Gruppen von 2–3 Studenten bearbeitet werden.
- Die Abgaben bestehen aus den Python-Quelltexten samt aller Bilder, die für die Bearbeitung der Übung verwendet wurden. Idealerweise wird das gesamte PyDev-Projektverzeichnis für die jeweilige Übung über Eclipse als .zip Archiv exportiert. Wichtig: Zur Vermeidung von Namenskonflikten in Eclipse, euren Nachnamen bitte als Präfix vor die Namen der die PyDev-Projekte setzen, z.B.: „müller-übung-x“.
- Teilaufgaben, deren Python-Dateien wegen Syntaxfehlern nicht ausführbar sind, werden nicht weiter korrigiert.
- Textaufgaben sollten in Form einer .doc, .odt oder (bevorzugt) als PDF-Datei abgegeben werden, und sollten (falls erforderlich) die benötigten Ausgabebilder der Aufgaben enthalten.
- Alle Übungsabgaben erfolgen über UniWorX[1].

Inhalt:

Konvolution, Unscharfmaskierung, Medianfilter

[1] https://uniworx.ifi.lmu.de

Aufgabe 1 Konvolution (⋆⋆⋆)

Bisher haben Sie Methoden zur Bildverbesserung kennengelernt, die auf einzelnen Pixeln basieren. Anhand von nur einem Pixel lässt sich jedoch keinerlei Aussage treffen, wie sich Grauwert, Helligkeit und Kontrast räumlich über ein Bild verteilt und verändert. Um räumliche Veränderungen in Helligkeit und Farbwerten von Bildern zu beschreiben und zu verändern, müssen Regionen anstatt von einzelnen Pixeln betrachtet werden. D.h. dass der neue Grauwert eines Pixels aus seinem alten Wert und dem Wert der benachbarten Pixel berechnet wird. In der Vorlesung haben Sie zwei Methoden kennengelernt: Konvolution und Korrelation. Konvolution wird meistens zum Filtern von Bildern verwendet, um z.B. Unschärfe zu entfernen oder Kanten zu erkennen. Korrelation wird meist verwendet, um bestimmte Features in Bildern zu finden.

Bei der Konvolution handelt es sich um eine gewichtete Summe, bestehend aus dem Grauwert eines Pixels und dem seiner Nachbarn. Die Koeffizienten zur Gewichtung werden in einer Matrix mit ungerader Dimension angegeben. Diese Matrix heißt **Konvolutionskern**.

Während der Konvolution werden die Einträge des Kerns der Reihe nach betrachtet und mit einem Pixel der darunterliegenden Region multipliziert, dann werden die Ergebnisse aufsummiert. Dabei ist zu beachten, dass der Wert links oben im Kern mit dem Pixel rechts unten in der darunterliegenden Region multipliziert wird. Danach werden die Einträge des Kern von links oben nach rechts unten betrachtet und mit den Pixeln in umgedrehter Reihenfolge (rechts unten nach links oben) multipliziert.

Für einen Kern h mit der Breite m und der Höhe n lässt sich die Konvolution also folgendermaßen beschreiben:

$$g(x,y) = \sum_{k=-n_2}^{n_2} \sum_{j=-m_2}^{m_2} h(j,k) f(x-j, y-k)$$

wobei gilt:

$$n_2 = \left\lfloor \frac{n}{2} \right\rfloor \text{ und } m_2 = \left\lfloor \frac{m}{2} \right\rfloor$$

Bei einem $3x3$ Kern wird also von -1 bis 1 summiert.
Da obige Formel etwas umständlich ist schreibt man für gewöhnlich:

$$g(x,y) = h * f(x,y)$$

und $*$ ist dabei der Konvolutionsoperator.

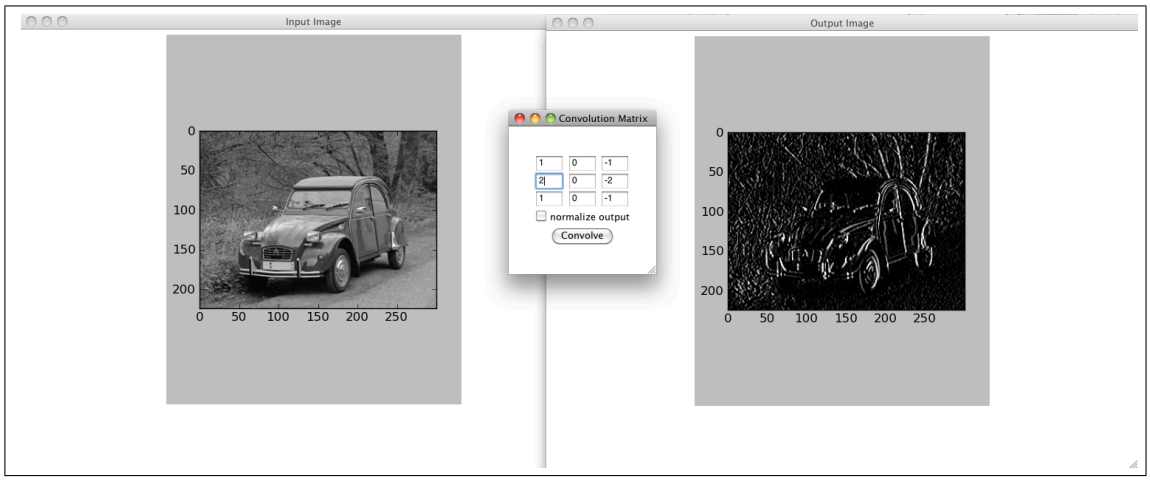

Abbildung 1: Python-Programm *convolve.py* zur Eingabe und Anwendung von Konvolutionskernen.

a) **Konvolution** Die Python-Datei *convolve.py*, erhältlich von der Vorlesungsseite, implementiert ein einfaches User-Interface, dass es erlaubt einen Konvolutionskern einzugeben, und auf ein Eingabebild anzuwenden. `ConvolveDialog` ist die Hauptklasse des Programms, in der die der Kern als Matrix eingegeben werden kann. Klickt der Benutzer den Button, wird die Funktion `self.command` aufgerufen, und das Ausgabebild, die Anwendung des Konvolutionskerns auf das Eingabebild, erzeugt (siehe Abbildung 1).

Der Klasse `ConvolveDialog` fehlt die Methode `def convolve(self, image, matrix):`, in der die eigentliche Konvolution (Folie 14) implementiert ist. `convolve` soll ein neues array zurückgeben, dass die Konvolution von `image` mit dem Kern `matrix` ist. Die Methode zur Behandlung von Bildrandpixeln bleibt Ihnen überlassen.

Implementieren Sie `convolve`, um *convolve.py* lauffähig zu machen. Probieren Sie dabei die folgenden Kerne aus und diskutieren Sie deren Wirkung.

$$(1)\begin{pmatrix}1&0&0\\0&0&0\\0&0&0\end{pmatrix} \qquad (2)\begin{pmatrix}0&0&0\\1&1&1\\0&0&0\end{pmatrix} \qquad (3)\begin{pmatrix}0&1&0\\0&1&0\\0&1&0\end{pmatrix}$$

$$(4)\begin{pmatrix}1&0&0\\0&0&0\\0&0&1\end{pmatrix} \qquad (5)\begin{pmatrix}0&0&0\\0&1&-1\\0&0&0\end{pmatrix} \qquad (6)\begin{pmatrix}0&0&0\\1&-2&1\\0&0&0\end{pmatrix}$$

b) **Separierbarkeit** Zeigen Sie, dass ein Boxcarfilter (Folien 28f, 37) separierbar ist und implementieren sie dazu eine Python-Methode. Vergleichen Sie (z.B. durch einen Zähler) die Anzahl der Operationen zwischen der separierten und nichtseparierten Version der Konvolution. *Hinweis:* Schauen Sie sich dazu an, wie die Separierbarkeit der Gaußfunktion auf Folie 46 gezeigt wird.

c) **Boxcarfilter vs. Gaußfilter** Vergleichen Sie die Ausgabe einer Konvolution eines Eingabebildes ihrer Wahl (z.B. ente.png) mit einem Boxcarfilter mit der eines Gaußfilters. Beschreiben Sie die qualitativen Unterschiede, falls vorhanden.

d) **Gaußfilter** Implementieren Sie eine Python-Methode, die Gaußkerne in beliebigen Größen ($n \times n$), sowie mit beliebigem σ erzeugt. Wenden Sie die Konvolution mit drei so erzeugten Gaußkernen auf ein Eingabebild ihrer Wahl an.

Aufgabe 2 Unscharfes Maskieren ($\star\star$)

Unscharfes Maskieren (Folie 47) ist eine Methode zur Verbesserung der Bildschärfe, die auf der Idee beruht, eine unscharfe Version des Bildes vom Originalbild abzuziehen, und diese *Unscharfmaske* wieder zum Originalbild zu addieren. Die Grundidee des unscharfen Maskierens lässt sich in folgende Schritte aufteilen:

- Erzeugen einer unscharfen Version I_B des Eingabebildes I mittels eines Gaußkerns des Radius r
- Erzeugen der Unscharfmaske $I_U = I - I_B$
- Das geschärfte Bild I' erhalten Sie durch Addieren von kI_U zum Originalbild I, wobei k ein Skalierungsfaktor ist (übliche Werte für k sind 0.5-1.5).

a) **Kern** Zeigen Sie, dass unscharfes Maskieren durch Konvolution mit einem einzigen Kern durchgeführt werden kann. Leiten Sie dazu einen passenden Kern aus den Schritten des oben gennanten Algorithmus her.

Hinweis: Das neutrale Element (d.h. erzeugt keine Änderungen im Ausgebebild) bei der Konvolution ist der Kern des Dirac-Deltas, dessen Matrix im zentralen Eintrag eine 1 und bei allen anderen Einträgen eine 0 enthält:

$$\delta = \begin{pmatrix} 0 & \cdots & 0 \\ & \ddots & \\ \vdots & 1 & \vdots \\ & & \ddots \\ 0 & \cdots & 0 \end{pmatrix}$$

b) **Parameter** Zusätzlich zum Kernradius r und dem Skalierungsfaktor k ist es noch möglich eine minimale Differenz d für den Subtraktionsschritt zu definieren. D.h. die Subtraktion des unscharfen Bildes vom Originalbild wird nur durchgeführt wenn die Differenz der Grauwerte mindestens d beträgt.

Implementieren Sie eine Python-Methode `def unsharp_mask(input, r, d, k):` die die unscharfe Maskierung eines Grauwertbildes anhand der Parameter r, d, k am Eingabebild durchführt.

Probieren Sie einige Kombinationen der Parameter aus und beschreiben Sie deren Auswirkung auf das Ausgabebild.

Aufgabe 3 Medianfilter ($\star\star$)

Medianfilter sind eine effektive Methode zur Entfernung von weißem und schwarzem Bildrauschen (salt and pepper noise).

a) **Medianfilter vs. Gaußfilter** Implementieren Sie eine Python-Methode die ein 3×3 Medianfilter (Folie 50f) auf das Eingabebild (lena-saltpepper.png) anwendet. Vergleichen Sie für ein mit Pepper Noise belastetem Eingabebild ihrer Wahl die Anwendung eines Medianfilters mit der Konvolution durch ein 3×3 Gaußfilter.