Kantenerkennung in C (15 Punkte)

(a) Original Bild

(b) Kanten

Abbildung 1: Beispiel Kantenerkennung

In diesem Projekt werden Sie ein Programm zur *Kantenerkennung* implementieren. Ziel ist es die Grenzen zwischen homogenen Bereichen mit ähnlicher Farbe/Grauwert zu finden. Kanten spielen eine wichtige Rolle bei der Objekt-Erkennung (Mensch und Maschine) und werden zum Beispiel bei der Bildkompression verwendet.

Das Projekt ist in mehrere Aufgaben unterteilt die sich an dem Stoff der Vorlesung orientieren. Die Deklarationen aller benötigter Funktionen sind bereits vorgegeben. Sie müssen keine weiteren hinzufügen. Das Projekt kann mit Hilfe des Befehls `make` einfach auf der Befehlszeile kompiliert und gebunden werden. Die Ausgabedateien des Übersetzers – inklusive der erzeugten Anwendung `edgedetection` – finden Sie im `build` Verzeichnis. Sie erhalten das Programmierprojekt mit

```
git clone https://prog2scm.cdl.uni-saarland.de/git/project2/<NAME>
```

Ersetzen Sie wie gewohnt `<NAME>` durch Ihren Benutzernamen.

1 Kantenerkennung

Die Basis des Algorithmus zur Kantenerkennung bildet die Berechnung der diskreten Ableitung des Bildes. Der Betrag der Ableitung repräsentiert dabei die Größe der Änderung in den Grauwerten benachbarter Pixel und deutet somit auf eine Kante hin. Welche Pixel als Kante gezählt werden wird durch einen Schwellwert bestimmt. Um nur die *wichtigen* Kanten zu berücksichtigen wird das Bild im vorhinein unscharf gemacht. Sowohl die Berechnung der Ableitung als auch das Verwaschen des Bildes basieren auf der mathematischen Operation **Faltung**. Diese wird im Folgenden noch genauer spezifiziert.

1.1 Algorithmus

Der Algorithmus besteht aus folgenden Schritten:

1. Einlesen des Original Bildes
2. Verwaschen des Bildes durch eine Faltung mit einem Gauss-Kernel
3. Ableitung des Bildes in x und y Richtung durch Faltung des verwaschenen Bildes mit vordefinierten Matrizen zur diskreten Ableitung
4. Berechnung des Betrages des Gradienten (Vektor aus Ableitung in x und y Richtung)

5. Bestimmung der Kanten als Pixel deren Betrag des Gradienten den gegebenen Schwellwert übersteigt
6. Ausgabe der Kanten als Schwarz-Weiß Bild

Die Ergebnisse der einzelnen Schritte sind in Abbildung 2 dargestellt.

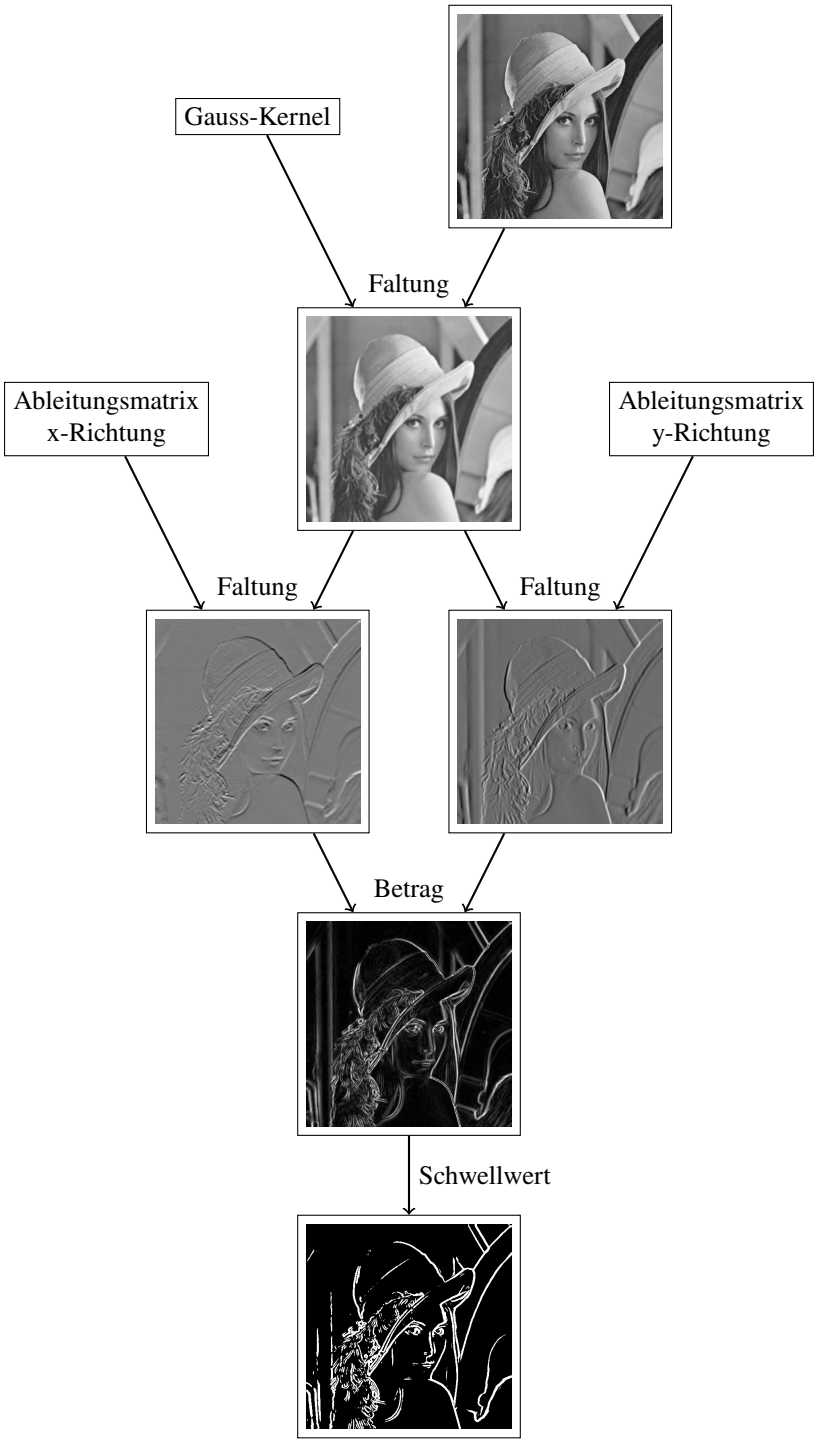


Abbildung 2: Struktur des Algorithmus zur Kantenerkennung

2 Allgemeine Hinweise zur Bearbeitung des Projekts

2.1 Befehlszeilenargumente

Die Anwendung implementiert folgende Befehlszeilensyntax:

```
edgedetection -T <threshold>  <image file>
```

Es folgt eine Beschreibung der Befehlszeilenargumente im Einzelnen:

- `-T <threshold>` Schwellwert für Schritt 5 des Algorithmus.
- `<image file>` Pfad zum Eingabebild

Das Einlesen der Befehlszeilenargumente ist bereits implementiert. Die eingelesenen Werte werden in den globalen Variablen in der Datei `argparser.h` gespeichert.

2.2 Bildformat

Als Ein- und Ausgabeformat verwenden wir das *portable graymap format*. Es hat die Endung `pgm`.

Die pgm-Dateien sind wie folgt aufgebaut:

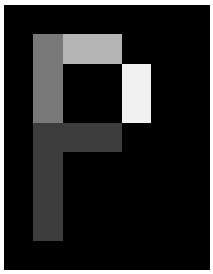
Kopfdaten

1. `P2`
2. Leerraum
3. Breite des Bildes
4. Leerraum
5. Höhe des Bildes
6. Leerraum
7. Maximalwert für die Helligkeit (hier immer `255`)

Bilddaten Auf den Kopfbereich folgen die Grauwerte der Pixel. Diese liegen alle zwischen 0 und 255. Die einzelnen Werte sind durch Leerraum getrennt. Leerräume sind Leerzeichen, Tabulator, Wagenrücklauf und Zeilenvorschub. In Abbildung 3 ist ein Beispiel dargestellt. Zur Veranschaulichung wurde nach je 7 Werten ein Zeilenumbruch eingefügt. Dies muss nicht der Fall sein.

```
P2
7 9
255
0  0   0   0   0 0 0
0 120 180 180   0 0 0
0 120   0   0 240 0 0
0 120   0   0 240 0 0
0 60   60  60   0 0 0
0 60    0   0   0 0 0
0 60    0   0   0 0 0
0 60    0   0   0 0 0
0  0    0   0   0 0 0
```

(a) PGM Format



(b) Dekodiertes Bild

Abbildung 3: Beispiel PGM Format

2.3 Weitere Hinweise

- Sie dürfen das mitgelieferte `Makefile` nicht verändern, da wir auf unseren Testservern diese Datei durch unsere eigene ersetzen.
- Sie müssen weder neue Dateien noch neuen Funktionen anlegen.
- Sie dürfen die Signaturen der Funktionen nicht ändern, da sie sonst nicht getestet werden können.
- Schauen Sie sich auch die Dokumentation der Funktionen in den entsprechenden `.h` Dateien an.
- Bilder werden als eine Reihung aus `floats` dargestellt. Der Pixel mit Koordinate x, y befindet sich an Position $x + w \cdot y$ in der Reihung.
- Sie können Funktionen aus der C Standard-Bibliothek `math.h` z.B. `sqrt` und `round` verwenden.

3 Aufgaben

Die Aufgaben sind vorrangig nach ihrer Schwierigkeit und dem Vorlesungsstoff geordnet und nicht nach der Reihenfolge im Algorithmus. Wir empfehlen Ihnen sich an die gegebene Reihenfolge zu halten.

Aufgabe 1: Schwellwert, 1 Punkt

Setzen Sie jeden Pixel, dessen Grauwert größer als der Schwellwert T ist auf rein weiß (255) und dessen Grauwert kleiner gleich T ist auf rein schwarz (0).

Implementieren Sie dazu die Funktion:

1. `void apply_threshold(float *img, int w, int h, int T)` in der Datei `image.c`.

Aufgabe 2: Betrag, 1 Punkt

Die diskrete Ableitung in x- bzw. y-Richtung wird durch die Faltung des Bildes mit dem *Sobel*-Operator in x- bzw. y-Richtung umgesetzt. Die beiden 3×3 Matrizen sind in der Datei `derivation.c` vordefiniert. Nach der diskreten Ableitung in x- und y-Richtung wird der Betrag des Gradienten benötigt. Dieser ist wie folgt definiert:

Seien D_x bzw. D_y die diskreten Ableitungen in x- bzw. y-Richtung. Der Betrag des Gradienten GM an der Stelle (x,y) ist dann definiert als:

$$GM(x, y) = \sqrt{D_x(x, y)^2 + D_y(x, y)^2}$$

Implementieren Sie dazu die Funktion:

1. `void gradient_magnitude(float *result, float *d_x, float *d_y, int w, int h)` in der Datei `derivation.c`.

Aufgabe 3: Skalierung der Grauwerte, 2 Punkte

Die Berechnung der Ableitung und des Betrags des Gradienten führen zu Ergebnissen außerhalb des Bereiches 0 bis 255. Deshalb müssen die Werte bevor sie als `pgm` Bild ausgegeben werden können, zurück in diesen Bereich skaliert werden. Die Skalierung ist wie folgt definiert:

Sei *max* bzw. *min* der maximale bzw. minimale Wert innerhalb des Bildes B. Dann ist der skalierte Wert B_s an Position (x,y) definiert als:

$$B_s(x,y) = \frac{B(x,y) - min}{max - min} \times 255$$

Implementieren Sie dazu die Funktion:

1. `void scale_image(float *result, float *img, int w, int h)` in der Datei `image.c`.

Aufgabe 4: Faltung, 4 Punkte

Die zweidimensionale Faltung zwischen dem Bild B mit der Breite w_B und Höhe h_B und der Matrix M mit der Breite w_M und und Höhe h_M für den Pixel an der Position (x, y) ist definiert als:

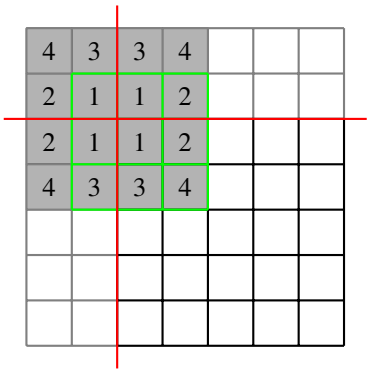
$$F(x,y) = \sum_{j=0}^{h_M-1} \sum_{i=0}^{w_M-1} M(i,j) \cdot B(x+i-a, y+j-b)$$

Hier ist a die x-Koordinate bzw. b die y-Koordinate des Mittelpunktes der Matrix. Sie können davon ausgehen, dass sowohl w_M als auch h_M ungerade und kleiner gleich als die entsprechenden Maße des Bildes sind.

Sollten die Koordinaten $B(x+i-a, y+j-b)$ nicht innerhalb des Bildes liegen wird der Grauwert des Pixels verwendet, der an der gegebenen Position liegt, wenn das Bild an seinen Außenkanten gespiegelt wird.

Beispiel Im folgenden Beispiel, basierend auf Abbildung 4, wird das Ergebnis der Faltung für den Pixel an Position $(0, 0)$ berechnet. Das Bild B ist in schwarz und die Matrix M in grün dargestellt. Die roten Linien entsprechen den Spiegelachsen. Die Funktion $mp(i, j)$ berechnet die gespiegelte Koordinate.

$$\begin{aligned}
F(0,0) =& M(0,0) \cdot B(mp(-1,-1)) + M(1,0) \cdot B(mP(0,-1)) + M(2,0) \cdot B(mp((1,-1)) + \\
& M(0,1) \cdot B(mp(-1,0)) + M(1,1) \cdot B(mp(0,0)) + M(2,1) \cdot B(mp(1,0)) + \\
& M(0,2) \cdot B(mp(0,1)) + M(1,2) \cdot B(mp(0,1)) + M(2,2) \cdot B(mp(1,1)) \\
=& M(0,0) \cdot B(0,0) + M(1,0) \cdot B(0,0) + M(2,0) \cdot B(1,0) + \\
& M(0,1) \cdot B(0,0) + M(1,1) \cdot B(0,0) + M(2,1) \cdot B(1,0) + \\
& M(0,2) \cdot B(0,1) + M(1,2) \cdot B(0,1) + M(2,2) \cdot B(1,1) \\
=& 1 \cdot 1 + 2 \cdot 1 + 1 \cdot 2 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 2 + (-1) \cdot 3 + (-2) \cdot 3 + (-1) \cdot 4 \\
=& -8
\end{aligned}$$



(a) Bild mit Spiegelung

1	2	1
0	0	0
-1	-2	-1

(b) Matrix

Abbildung 4: Beispiel Faltung: Das Bild ist schwarz die Matrix grün dargestellt. Die roten Linien entsprechen den Spiegelachsen.

Implementieren Sie dazu folgende Funktionen:

1. `float get_pixel_value(float *img, int w, int h, int x, int y)` in der Datei `image.c`.
2. `void convolve(float *result, float *img, int w, int h, const float *kernel, int w_k, int h_k)` in der Datei `convolution.c`.

Eine genauere Beschreibung der Funktionen können Sie den jeweiligen `.h` Dateien entnehmen.

Aufgabe 5: Einlesen und Ausgeben eines Bildes, 5 Punkte

Implementieren Sie jeweils eine Funktion zum Einlesen bzw. Ausgeben von `pgm` Bilddateien. Das Format ist in Abschnitt 2.2 beschrieben. Sie können davon ausgehen, dass die Eingabedateien *keine* Kommentare enthalten. Schreiben Sie allerdings Ihre Einleseroutine robust, sodass sie Fehler in der Eingabedatei abfängt und die Anwendung ggf. beendet und einen *NULL* Zeiger zurückgibt. Zu diesen Fehlern gehören zu wenig/zu viel Bildpunkte, Höhen-/Breitenangaben kleiner gleich null und ähnliches.

Da die Größe der Bilder erst zur Laufzeit des Programmes bekannt ist, muss dynamisch Speicher alloziert werden. Zusätzlich muss dieser Speicher wieder frei gegeben werden, wenn er nicht mehr verwendet wird.

Implementieren Sie dazu folgende Funktionen:

1. `float* array_init(int size)` in der Datei `image.c`.
2. `void array_destroy(float *m)` in der Datei `image.c`.
3. `float* read_image_from_file(const char *filename, int *w, int *h)` in der Datei `image.c`.
4. `void write_image_to_file(float *img, int w, int h, const char *filename)` in der Datei `image.c`.

Aufgabe 6: Main, 2 Punkte

Die letzte Aufgabe besteht darin die implementierten Funktionen zu dem Algorithmus aus Abschnitt 1.1 zusammenzufügen. Füllen Sie dazu die angegebenen Bereiche in der Funktion `int main(int const argc, char **const argv)` in der Datei `main.c`. Die Kommentare beschreiben jeweils was genau Sie implementieren müssen. Denken Sie daran dynamisch alloziierten Speicher wieder freizugeben.

Achtung: Manche Zwischenergebnisse müssen wieder in den Bereich 0 bis 255 skaliert werden bevor sie als Bild ausgegeben werden können. Im Algorithmus selbst werden aber immer unskalierte Zwischenergebnisse verwendet.

4 Tests

Es wird nicht nur das ganze Programm sondern auch einzelnen Funktionen getestet. Deshalb können Sie nach jeder Aufgabe schon überprüfen ob Ihre Implementierung den entsprechenden public Test besteht. Sie können im Basisverzeichnis die *public* Tests mit dem Kommando `make tests` selbst durchführen. Sie müssen alle public Tests eines Aufgabenteils bestehen, um in diesem Aufgabenteil überhaupt Punkte zu erlangen. Die public Tests werden außerdem in regelmäßigen Abständen zusammen mit geheimen *daily* Tests auf unserem Testserver ausgeführt. Sie erhalten über den Ausgang der Tests eine Benachrichtigung per Email. Nach Ende des Projekts wird Ihre Abgabe mithilfe weiterer *secret* Tests ausgewertet.

4.1 Einzelne Tests

Sie können auch einzelne Tests ausführen, was insbesondere dann nützlich ist, wenn größere Teile Ihrer Implementierung noch fehlen.

- Mit `test/run-tests.py -l` können Sie sich die Namen aller Tests ausgeben lassen.
- Mit `test/run-tests.py -f <name>` lassen Sie nur den Test `<name>` laufen.

5 Abgabe

Zur Bewertung ziehen wir den Zustand Ihres git-Depots im `master` Zweig *vom 15.05.2018 um 23:59* heran.