Projekt 2

Prof. Dr. Sebastian Hack
Julian Rosemann, B. Sc.
Tina Jung, B. Sc.
Maximilian Fickert, M. Sc.

Kantenerkennung in C (12 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.

Sie erhalten das Programmierprojekt mit

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

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

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.

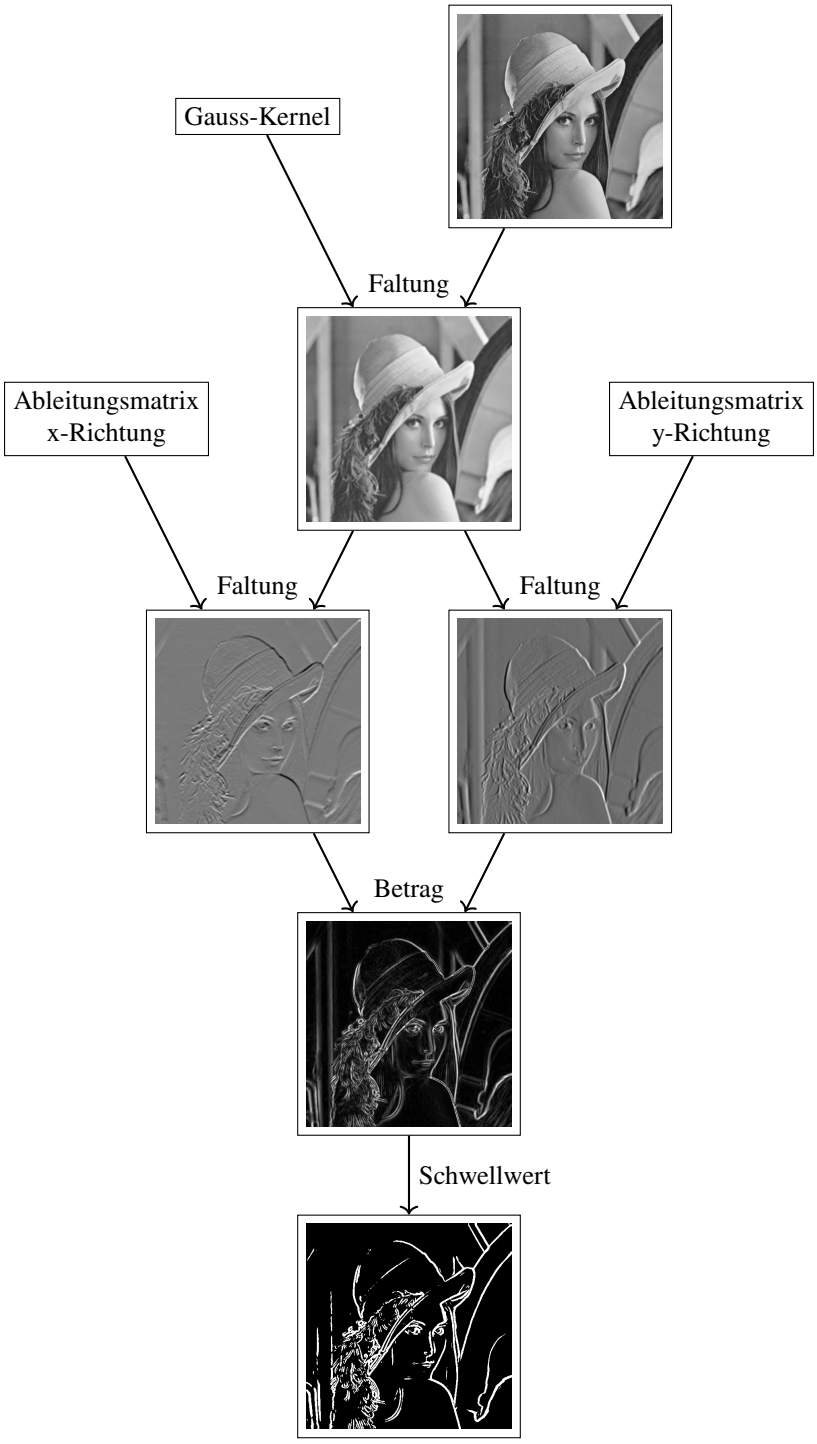


Abbildung 2: Struktur des Algorithmus zur Kantenerkennung

2 Allgemeine Hinweise zur Bearbeitung des Projekts

2.1 Kompilieren des Projekts

Das Projekt kann mit Hilfe des Befehls `make` einfach auf der Befehlszeile kompiliert werden. Die erzeugten Anwendung `edgedetection` finden Sie im `bin` Verzeichnis.

2.2 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

Wollen Sie das Programm beispielsweise vom Wurzelverzeichnis des Projekts mit einem Schnellwert von 100 und dem Testbild `img_P.pgm` ausführen, so entspricht dies dem Befehl

```
./bin/edgedetection -T 100 test/data/input/img_P.pgm
```

Das Einlesen der Befehlszeilenargumente ist bereits implementiert und muss von Ihnen nicht weiter beachtet werden. Die eingelesenen Werte werden in den globalen Variablen `image_file_name` und `threshold` gespeichert (siehe `argparser.h`).

2.3 Bildformat

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

Die pgm-Dateien sind wie folgt aufgebaut:

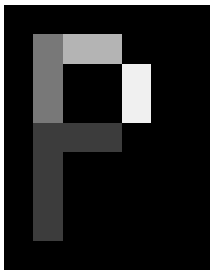
Kopfdaten

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 ganzzahligen Grauwerte der Pixel. Diese liegen alle zwischen 0 und 255. Die einzelnen Werte sind durch beliebig langen Leerraum (mind. ein Leerzeichen, Tabulator, Wagenrücklauf oder Zeilenvorschub) voneinander getrennt. 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

2.4 Repräsentation eines Bildes im C-Programm

Bilder werden als eine Reihung aus `floats` dargestellt. Die Elemente der Reihung stellen die Grauwerte der Pixel dar. Um ein zweidimensionales Bild der Breite w und Höhe h darzustellen wird eine eindimensionale Reihung der Größe $w \cdot h$ benötigt. Die Pixel werden zeilenweise in der Reihung gespeichert: Der Pixel mit Koordinate x, y befindet sich an Position $x + w \cdot y$ in der Reihung, wobei w die Breite des Bildes in Pixel ist.

2.5 Weitere Hinweise

- Sie dürfen das mitgelieferte `Makefile` nicht verändern.
- Sie dürfen die Signaturen der Funktionen nicht verändern.
- Sie müssen weder neue Dateien noch Funktionen anlegen, lediglich die bereits angelegten implementieren.
- Die `(void)`-Casts dienen nur zur Unterdrückung der Compilerwarnung für unbenutzte Variablen und dürfen (aber müssen nicht) entfernt werden.
- Schauen Sie sich auch die Dokumentation der Funktionen in den entsprechenden `.h` Dateien an!
- Sie dürfen Funktionen aus der C Standard-Bibliothek `math.h` (insbesondere `sqrt`) 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. Alle zu implementierenden Funktionen sind in den entsprechenden `.h` Dateien dokumentiert.

3.1 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 folgende Funktion in der Datei `image.c`:

```
void apply_threshold(float *img, int w, int h, int T)
```

3.2 Betrag (1 Punkt)

Nach der diskreten Ableitung in x- und y-Richtung wird der Betrag des Gradienten benötigt. 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 definiert als:

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

Implementieren Sie dazu die folgende Funktion in der Datei `derivation.c`:

```
void gradient_magnitude(float *result, const float *d_x, const float *d_y, int w, int h)
```

Der Parameter `result` stellt das Ausgabebild dar, in welches die berechneten Beträge eingetragen werden müssen. Die Parameter `d_x` und `d_y` sind Reihungen, welche die Ableitungen D_x und D_y des Bildes in x- bzw. y-Richtung angeben.[1] Die Parameter `w` und `h` geben Breite und Höhe des Bildes an. Die Standardbibliotheksfunktion `sqrt` dürfte hier nützlich sein.

3.3 Skalierung der Grauwerte (1 Punkt)

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:

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

Im Sonderfall $max = min$ soll ein schwarzes Bild ausgegeben werden.

Implementieren Sie dazu die folgende Funktion in der Datei `image.c`:

```
void scale_image(float *result, const float *img, int w, int h)
```

Der Parameter `result` stellt das Ausgabebild dar, in welches die berechneten skalierten Grauwerte eingetragen werden müssen. Der Parameter `img` stellt das zu skalierende Eingabebild dar. Die Parameter `w` und `h` geben Breite und Höhe des Bildes an.

[1] Wie bei Bildern werden die Ableitungen an den Pixeln zeilenweise gespeichert: Die Ableitung für Pixel x, y befindet sich and der Stelle $x + w \cdot y$ der Reihung.

3.4 Faltung (3 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 Höhe h_M für den Pixel an der Position (x, y) ist definiert als:

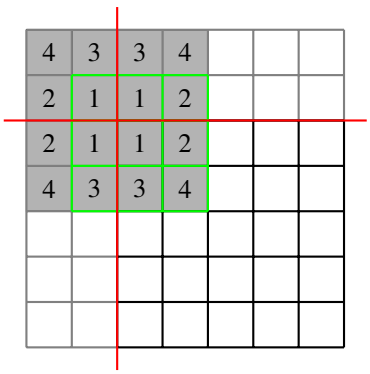
$$F(x,y) = \sum_{j=0}^{h_M-1} \sum_{i=0}^{w_M-1} M(i,j) \cdot B(x+i-a, y+j-b)$$

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 sind, sodass der Mittelpunkt eindeutig definiert ist.

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: Faltung. Das Bild ist in schwarz, die Matrix in grün dargestellt. Die roten Linien entsprechen den Spiegelachsen.

Überlegen Sie sich zunächst wie Sie eine Koordinate spiegeln, falls sie außerhalb des Bildes liegt. Implementieren Sie anschließend zuerst die Funktion

```
float get_pixel_value(const float *img, int w, int h, int x, int y)
```

in der Datei `image.c`. Diese Funktion gibt den Wert des Pixels an Koordinate (`x`, `y`) aus, falls diese innerhalb des Bildes liegt, ansonsten den Wert des Pixels an der gespiegelten Stelle. Nutzen Sie diese um die Funktion

```
void convolve(float *result, const float *img, int w, int h, ...)
```

in der Datei `convolution.c` zu implementieren.

3.5 Einlesen und Ausgeben eines Bildes (4 Punkte)

Implementieren Sie jeweils eine Funktion zum Einlesen bzw. Auslesen von `pgm` Bilddateien. Das Format ist in Abschnitt 2.3 beschrieben. Schreiben Sie allerdings Ihre Einleseroutine so, dass sie die folgenden potentiellen Fehler in der Eingabedatei abfängt und in diesem Fall einen `NULL` Zeiger zurückgibt:

- Die Datei existiert nicht
- Kaputte Kopfdaten, insbesondere Höhen-/Breitenangaben kleiner gleich null
- Zu wenige oder zu viele Bildpunkte
- Grauwerte außerhalb des Bereichs 0-255

Da die Größe des Bildes erst zur Laufzeit des Programmes bekannt ist, muss dynamisch Speicher zur Repräsentation des Bildes alloziiert werden (siehe 2.4). Zusätzlich muss dieser Speicher wieder frei gegeben werden, wenn er nicht mehr verwendet wird. Implementieren Sie dazu zuerst folgende Funktionen in der Datei `image.c`:

1. `float* array_init(int size)`
2. `void array_destroy(float *m)`

Die Funktion `array_init` alloziiert dynamisch eine Reihung an floats die das Bild als Reihung von Graustufen repräsentieren soll. Ergänzend hierzu gibt die Funktion `array_destroy` eine dynamisch alloziierte Reihung an floats wieder frei. Implementieren Sie anschließend folgende Funktionen in der Datei `image.c`:

1. `float* read_image_from_file(const char *filename, int *w, int *h)`
2. `void write_image_to_file(const float *img, int w, int h, const char *filename)`

Die Funktion `read_image_from_file` erwartet den Pfad zur Eingabedatei im `filename` Parameter. Die Parameter `w` und `h` dienen zur *Ausgabe* der Breite und Höhe des eingelesenen Bildes. Der Rückgabewert ist die dynamisch alloziierte Reihung an floats die das eingelesene Bild darstellt. Denken Sie daran alloziierten Speicher im Fehlerfall wieder freizugeben und geöffnete Dateien zu schließen bevor Sie von der Funktion zurückkehren!

Die Funktion `write_image_to_file` erwartet ein Bild im `img` Parameter. Die Parameter `w` und `h` geben Breite und Höhe des Bildes an. Der Parameter `filename` gibt den Namen der Ausgabedatei an. Die Grauwerte des ausgegebenen Bildes müssen laut Format Ganzzahlen sein. Runden Sie daher die Pixelwerte zur nächsten Ganzzahl ab bevor Sie diese in die Datei schreiben. Denken Sie daran geöffnete Dateien zu schließen bevor Sie von der Funktion zurückkehren!

Hinweis: Sie müssen für die Ausgabe keine Fehlerbehandlung implementieren.

3.6 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 Main-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: Zwischenergebnisse müssen in den Bereich 0 bis 255 skaliert werden (siehe 3.3) bevor sie als Bild ausgegeben werden können. Dies ist nur für die Ausgabe als Bild nötig. Der Algorithmus selbst rechnet mit den unskalierten Zwischenergebnissen weiter.

Korrektur: Das Zwischenergebnis für das **verwaschene** Bild ist **unskaliert** auszugeben. Der dazugehörige Kommentar in `main.c` ist inkorrekt falls Sie das Projekt *vor dem 18.05, 14:00* geklont haben.

4 Tests

Es wird nicht nur das ganze Programm sondern auch die einzelnen Funktionen getestet. Sie sollten nach jeder Aufgabe überprüfen ob Ihre Implementierung die entsprechenden public Tests 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.

Achten Sie darauf das Projekt neu zu kompilieren bevor Sie die Tests laufen lassen!

5 Bearbeitung in Visual Studio Code

Wir empfehlen Ihnen das Projekt mithilfe von Visual Studio Code zu bearbeiten. Sie dürfen natürlich auch den Editor Ihrer Wahl verwenden. Wichtig ist, dass das Projekt mit den oben aufgeführten Befehlen kompiliert und getestet werden kann. Der integrierte Terminal von Visual Studio Code kann für Befehle verwendet werden. Darüber hinaus stellen wir Ihnen einige Konfigurationsdateien bereit um Ihnen die effiziente Arbeit im Editor zu ermöglichen. Sie können das Projekt dadurch direkt mit der Tastenkombination `Strg + Shift + B` kompilieren. Für die gängigen Befehle zum Testen und Kompilieren existieren Tasks die Sie über *Terminal* → *Run Tasks...* ausführen können.

Um das Projekt in der virtuellen Maschine im Visual Studio Code Editor zu debuggen müssen Sie die nötigen Erweiterungen erst installieren. Rufen Sie dazu einfach das von uns bereitgestellte Installationsskript mit `./scripts/install_cpptools.sh` auf und starten Sie Visual Studio Code neu. Anschließend können Sie die Main-Funktion des Projekts debuggen. Wechseln Sie dazu zum Debug-Panel (`Strg + Shift + D`) und führen Sie die voreingestellte Startkonfiguration "Debug Main" aus. Das Programm wird am Anfang der Main-Funktion angehalten.

6 Abgabe

Zur Bewertung ziehen wir den Zustand Ihres git-Depots im `master` Zweig *vom 25.05.2021 um 23:59* heran.

Viel Erfolg!