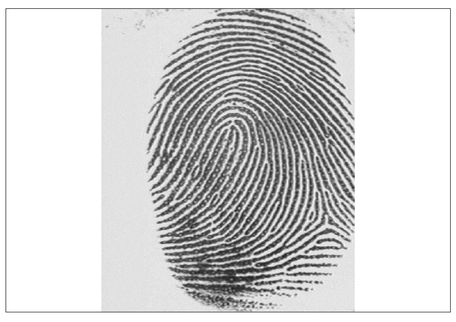
Fachhochschule Brandenburg
Fachbereich Informatik und Medien



Ausgangsbild

Funktionale Programmierung mit F# und Microsoft Accelerator am Beispiel Bildverarbeitung

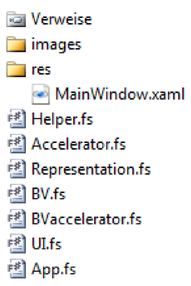
Bachelorarbeit, vorgelegt von Tobias Schacht

Kantenbild

Zielstellung

Es soll die Funktionale Programmierung im Allgemeinen und konkret anhand der Sprache F# präsentiert werden. Am Anwendungsbereich der Bildverarbeitung sollen grundlegende Bildverarbeitungsoperationen in einer grafischen Anwendung umgesetzt werden. Da sich Operationen der Bildverarbeitung durch hohe Datenmenge als aufwändig, aber auch stark iterativ gestallten, indem gleichartige Transformationen auf alle Pixel eines digitalen Bildes angewendet werden, soll die Berechnung unter Ausnutzung von Datenparallelität auf die Grafikkarte ausgelagert werden, um eine schnelle Verarbeitung auch bei großen Bildern zu erreichen. Dies wird mittels der Bibliothek Accelerator umgesetzt.

Softwarearchitektur



Die Software ist in sieben Module aufgegliedert. Der Entwurf der grafischen Oberfläche ist in einer XAML-Datei definiert. Diese wird im Modul *App* beim Start der Anwendung geladen und mit der entsprechenden Funktionalität hinterlegt. Dieses Modul dient als Einstiegspunkt der Anwendung.

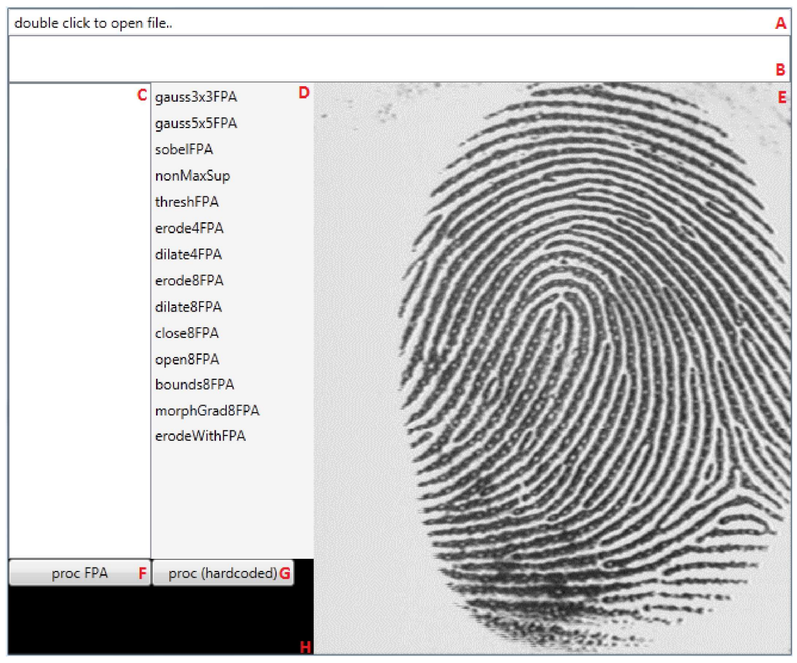
Die restlichen Module umfassen spezifische Funktionen: *Helper* beinhaltet allgemeine Hilfsfunktionen. *UI* und *Representation* umfassen Funktionalität der Oberfläche und nötigen Transformationsfunktionen zwischen verschiedenen Repräsentation von digitalen Bildern (z.B. das Einlesen eines Bitmaps und Umwandlung in ein Format, welches die Verarbeitung auf der Grafikkarte erlaubt und umgekehrt).

BVaccelerator und *BV* umfassen die Umsetzung von Bildverarbeitungsoperationen mit und ohne Auslagerung auf die Grafikkarte mit Hilfe der Bibliothek Accelerator. Allgemeine Hilfsfunktionen zum Umgang mit der Bibliothek sind im Modul *Accelerator* definiert.

Oberfläche & Bedienung

Die Oberfläche ermöglicht die Anwendung der verfügbaren Bildverarbeitungsoperationen (dargestellt im Bereich D) auf ein ausgewähltes Bild. Durch Ziehen und Ablegen im Bereich C können die Operationen kombiniert werden und werden zusammen als Ganzes auf der Grafikkarte berechnet (Schaltfläche F). Zum Vergleich wurde der *Sobel*-Operator als Variante ohne Ausnutzung von Datenparallelität und Auslagerung umgesetzt (Schaltfläche G).

Das Ergebnisbild wird in Originalgröße in E ausgegeben sowie als Miniaturbild mit zugehörigem Histogramm im Bereich H. In B erfolgt die Ausgabe der benötigten Zeit. Testbilder können durch Aufruf eines entsprechenden Auswahldialoges über den Bereich A geöffnet werden.



Beispiel: Sobel-Algorithmus

Der Sobeloperator erstellt ein Kantenbild durch Berechnung von Gradientenbildern. Hier wird eine Maske (z.B. 3x3) über alle Bildpunkte geschoben. Diese enthält Koeffizienten mit der die korrespondierenden Nachbarpixel des aktuellen Referenzpixels in der Mitte multipliziert werden. Der neue Wert des Referenzpixels ergibt sich aus der Summe aller mit den Koeffizienten gewichteten Nachbarpixel.

Mit den dargestellten Sobel-Filtermasken für das Gradientenbild in x- und y-Richtung wird eine diskrete Faltung vorgenommen. Die beiden entstehenden Gradientenbilder gx und gy werden mit der Gradienten-Formel zu einem Kantenintensitätsbild (Gradientenbild) kombiniert.

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

(a) x-Richtung

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

(b) y-Richtung

Sobel-Filtermasken

$$M(x,y) = \sqrt{g_x^2 + g_y^2}$$

Gradienten-Formel

Datenparallele Umsetzung

Die Anwendung von Filtermasken kann datenparallel erfolgen, indem der Zugriff auf Nachbarpixel durch Schiebeoperationen des gesamten Bildes erfolgt. So wird nicht über jeden Pixel des Bildes in einer Schleife iteriert, sondern man erhält bspw. alle linken Nachbarpixel durch Schieben des gesamten Bildes nach links.

Die Berechnung von gx und gy erfolgt mit spezialisierten Funktionen. Diese nutzen die lineare Separierbarkeit der Masken aus. Das Bild wird mit dem Vektor (1 2 1) bzw. (1 2 1)T multipliziert, d.h. Multiplikation der beiden Nachbarbilder sowie des Originalbildes mit den entsprechenden Koeffizienten.

Danach wird linkes und rechts bzw. oberes und unteres Nachbarbild des Ergebnisbildes voneinander subtrahiert (Vektorkomponente (−1 0 1)T bzw. (−1 0 1) der Filtermasken).

Anstatt 8 werden auf diese Weise nur 4 Schiebeoperationen benötigt.

Funktionale Programmierung

Die Funktionale Programmierung (FP) baut auf dem mathematischen Begriff der Funktion auf. In der Mathematik wird eine Funktion als Beziehung zwischen zwei Mengen definiert. Jedem Wert aus der Menge des Definitionsbereichs wird genau ein Element aus der Menge des Wertebereichs zugeordnet. Somit besteht ein funktionales Programm aus einer Reihe von Funktionsdefinitionen. So gibt es in rein funktionalen Sprachen kein Anweisungen oder veränderbare Variablen. Stattdessen gibt es nur Ausdrücke, die zu einem Wert eines bestimmten Typs evaluieren.

Charakteristisch ist die zur mathematischen Schreibweise oft sehr ähnliche Syntax und der sehr prägnante Programmcode.

Die Theoretische Basis funktionaler Sprachen bildet das λ-Kalkül von Church. Basierend darauf wurde 1958 von John McCarthy die funktionale und heute zweitälteste Programmiersprache LISP („List Processing“) am Massachusetts Institute of Technology (MIT) entwickelt. FORTRAN, die älteste Programmiersprache, ist der erste Vertreter der imperativen Programmiersprachen. Beide unterscheidet die Herangehensweise, wie die zugrunde liegende Maschine abstrahiert wird. Imperative Sprachen bilden ausgehend vom Maschinencode, immer höhere Abstraktionsebenen, während funktionale Sprachen als Vertreter der deklarativen Sprachen von einem mathematischen Modell ausgehen und abwärts in Richtung Maschinencode wachsen.

Die Sprache F#

F# ist eine seit 2002 bei Microsoft Research entwickelte Sprache für das .NET-Framework. Mit dem Erscheinen von Visual Studio 2010 ist die Sprache offizielles Microsoft Produkt.

F# umfasst mehrere Programmierparadigmen. Der Fokus liegt vornehmlich auf der Funktionalen Programmierung. Die Sprache unterstützt als .NET-Sprache aber auch die objektorientierte Programmierung mit den üblichen Konstrukten. Die Einbindung von Modulen anderer .NET-Sprachen ist möglich. Umgekehrt ist durch Nutzung der objektorientierten Konstrukte von F# eine Erzeugung von Modulen mit entsprechend objektorientierter Schnittstelle realisierbar, sodass diese aus anderen .NET-Sprachen heraus genutzt werden können.

```
let toBinaryNumStr (d : int) : string =
    d
    |> Seq.unfold (fun x -> if x > 0 then Some (x%2, x/2) else None)
    |> Seq.toList // Liste mit Koeffizienten der Zweierpotenzen
    |> List.rev // Liste umkehren
    |> List.fold (fun i i' -> i.ToString () + i'.ToString ()) "" // String
       erzeugen

let bins = List.map toBinaryNumStr [1..10] // bins : string list = ["1"; "10";
    "11"; "100"; "101"; "110"; "111"; "1000"; "1001"; "1010"]
```

F#-Beispielprogramm: Umrechnung Dezimalzahlen zu Dualzahlen

Betreuer: Prof. Dr. sc. techn. Harald Loose, Dipl.-Inf. Ingo Boersch
Abgabe: 27.10.2010