VigraPLT Referenz-Handbuch

Version 0.5.0

Benjamin Seppke

8. April 2010

Inhaltsverzeichnis

Einleitung

Der Name VIGRA steht für "Vision with Generic Algorithms". Es handelt sich hierbei um eine, hauptsächlich von Ullrich Köthe entwickelte, neuartige C++ Bibliothek zur Bildverarbeitung, die das Hauptaugenmerk auf Generizität und Wiederverwendbarkeit der Algorithmen und Datenstrukturen legt. Dies wird mit mit Hilfe des in C++ enthaltenen Template-Mechanismus' erreicht, der bereits in der wohl bekannten C++ Standard Template Library (STL) zum Einsatz kommt. Die STL - so könnte man sagen - stand Pate für diese Bibliothek.

Dies alles führt schlussendlich dazu, dass sich VIGRA-Komponenten einfach an die eigenen Bedürfnisse anpassen lassen, ohne dabei auf die hohe Geschwindigkeit von C++ verzichten zu müssen. Um noch mehr zu erfahren, bietet sich ein Besuch der VIGRA Homepage an!

Die VIGRA ist eine reine C++ Bibliothek. Dies hat sicherlich einige sehr positive Aspekte, wie z.B. die Geschwindigkeit der Algorithmen, die hervorragende Ausnutzung der Template-Mechanismen und des objektorientierten Programmierstils, der die VIGRA recht verständlich macht.

Dennoch gibt es auch Gründe, die gegen eine Wahl der Sprache C++ sprechen, wie z.B. die fehlende Interaktivität durch die nötigen Code-Compile-Zyklen oder die mäßige Unterstützung von anderen Paradigmen außer dem imperativen und dem OO-Modell. Zudem ist C++ nicht gerade als High-Level-Sprache berühmt, weshalb nach wie vor viele höhere Wissenskonzepte in anderen Sprachen beschrieben werden.

Kurz und knapp zusammengefasst ergibt sich, dass alleine schon der erste Punkt (die fehlende Interaktivität) die Anbindung an eine andere Sprache rechtfertigen kann. Daher wird auch zurzeit an einer verbesserten Python-Schnittstelle der VIGRA gearbeitet.

1 Voraussetzungen und Installation

Da es mit Scheme seit der Version 4.0 recht leicht möglich ist, sogenannten "Foreign-Code", also Code der nicht mit Scheme selbst geschrieben wurde, zu laden, wird diese Schnittstelle (das Foreign-Funktion-Interface: FFI) im Rahmen der VigraPLT ausgenutzt. Sie ermöglicht es, C-konforme Shared Libraries (unter Windows: DLLs, unter MacOSX: dylibs und unter Linux: so) dynamisch und zur Laufzeit an Scheme zu binden, so dass die darin enthaltenen Funktionen benutzt werden können.

Da die VIGRA allerdings nur für den Import- bzw. Export von Bildern eine Shared Library anbietet, musste als erste Zwischenstufe ein C-Wrapper für die VIGRA-Funktionen geschrieben werden (VIGRA_C), die innerhalb der Scheme Umgebung zur Verfügung stehen sollen. An diese dynamische C-Bibliothek werden dann die entsprechenden Scheme Funktionen gebunden, die den Aufruf der eigentlichen C-Routinen vor dem Benutzer verstecken.

Zusammengefasst müssen also folgende Schritte durchgeführt werden, um die VigraPLT zu installieren:

1. Herunterladen und Entpacken von VigraPLT (inkl. VIGRA_C)
2. VIGRA_C kompilieren (make und make install)
3. Die dynamische Bibliothek der VIGRA_C in das Verzeichnis von VigraPLT ablegen
4. VigraPLT in das „collects“ Verzeichnis von PLT Scheme linken

   Unter MacOS X oder Linux z.B. mit

   ```
   > ln -s /Path/to/vigraplt
            /path/to/scheme/collects/vigraplt
   ```

Darüber hinaus gibt es für die Installation auf den Pool-iMacs und für Windows (allgemein) fertige Binaries zum Herunterladen.

Getestet werden kann die erfolgreiche Installation z.B. mit einem Durchlauf der „examples.ss“ Datei, die im VigraPLT Verzeichnis liegt.

2 Verwendete Datenstrukturen

Durch die Anbindung der Datenstrukturen an die C-Schnittstelle, sollte möglichst wenig Overhead entstehen. So sollte ein Bild auch nur ein Mal im Speicher vorhanden sein, und nicht etwa doppelt, um es an die C-Schnittstelle in einer geeigneten Form (kopiert) weiterzugeben. Daher werden in diesem Kapitel die neuen Datenstrukturen beschrieben, die mit der VigraPLT in Scheme eingeführt werden.

2.1 Repräsentation von Bildern

Die Repräsentation eines Bildkanals in dieser Erweiterung beruht intern auf einem „cvector“ vom Typ „float“, da dieser eine einfache Übergabe an die C-Schnittstelle bietet. Leider ist dieser aber nur eindimensional definiert, weshalb er in der Datei „vigraplt.carray.ss“ zunächst auf n Dimensionen erweitert wurde.

Diese Erweiterung ist deshalb wichtig, weil wir ein Bild als zweidimensionalen Datentyp beschreiben möchten, der in der Lage ist, für jeden Pixel eines Kanals einen float (Gleitkommazahl) Wert zu speichern. Das n-dimensionale „carray“ ist für den praktischen Umgang mit Bildern recht unwichtig, da es spezielle Funktionen gibt, welche sich in der Datei „vigraplt.helpers.ss“ finden.

Ein Bild definiert sich als Liste all seiner Kanäle: So wird ein Grauwertbild durch die Liste '(Grauwertkanal) repräsentiert, und ein RGB-Bild durch '(Rotkanal Grünkanal Blaukanal). Folgende Funktionen erlauben den einfachen Umgang mit Bildern:

(make-image width height numBands . init-val) → image

Bedingungen:

width, height, numBands: Ganzzahlen (Breite, Höhe, Anzahl der Känale)

init-val: Liste von Gleizkommazahlen

(length init-val) = numBands

Beschreibung:

Erstellt ein neues Bild mit Breite width, Höhe height und numBands Bändern. Falls angegeben, wird das neue Bild in jedem Pixel mit dem Wert init-val gefüllt.

(image-band image band_id) → carray

Bedingungen:

image: image

band_id: Ganzzahl

Beschreibung:

Gibt das zwei-dimensionale Array der Werte aller Pixel für den Kanal band_id zurück.

(image-data image band_id) → cvector

Bedingungen:

image: image

band_id: Ganzzahl

Beschreibung:

Gibt den ein-dimensionalen Vektor der Werte aller Pixel für den Kanal band_id zurück.

(image-numbands image) → Ganzzahl

Bedingungen:

image: image

Beschreibung:

Gibt die Anzahl von Kanälen des Bildes zurück.

(image-width image) → Ganzzahl

Bedingungen:

image: image

Beschreibung:

Gibt die Breite des Bildes zurück.

(image-height image) → Ganzzahl

Bedingungen:

image: mage

Beschreibung:

Gibt die Höhe des Bildes zurück.

(image-ref image x y band_id) → Gleitkommazahl

Bedingungen:

image: image

x, y, band_id: Ganzzahlen, wobei $0 \leq x <$ (image-width image)
und $0 \leq y <$ (image-height image)
und $0 \leq$ band_id $<$ (image-numbands image)

Beschreibung:

Gibt den Wert des Pixels des Bildes an der Stelle (x,y) für Kanal band_id zurück.

(image-set! image x y band_id val) → void

Bedingungen:

image: image

x, y, band_id: Ganzzahlen, wobei $0 \leq x <$ (image-width image)
und $0 \leq y <$ (image-height image)
und $0 \leq$ band_id $<$ (image-numbands image)

val: Gleitkommazahl

Beschreibung:

Setzt den Wert des Pixels des Bildes an der Stelle (x,y) für den Kanal band_id auf den übergebenen Wert.

WARNUNG:

Dieses „Umbiegen“ des Bildwertes entspricht nicht dem funktionalen Paradigma und kann zu ungewollten Seiteneffekten führen!

(copy-image image) → image

Bedingungen:

image: image

Beschreibung:

Gibt eine Kopie des übergebenen Bildes zurück.

(image->{red | green | blue}-band image) → carray

Bedingungen:

image: image

Beschreibung:

Gibt die zwei-dimensionalen Arrays der jeweiligen Bänder eines Bildes zurück.

(image->{ red | green | blue } image) → image

Bedingungen:

image: image

Beschreibung:

Gibt die jeweiligen Bänder eines Bildes als neues Bild zurück.

(image->list image) → list

Bedingungen:

image: image

Beschreibung:

Wandelt ein Bild in eine (Kanal)Liste von Listen von Zeilenlisten um und gibt diese zurück.

Beispiel:

```
'( [0   1    2                    '( '(  (0    1    2)
   3    4    5                           (3    4    5)
   6    7    8          →                (6    7    8)
   12   13   14] )                      (12   13   14) ) )
```

(list->image lst) → image

Bedingungen:

lst: Eine Liste von Listen von Listen gleicher Länge

Beschreibung:

Inverse Funktion zu image->list, die eine Liste von Listen wieder in ein image umwandelt.

2.2 Repräsentation von Matrizen

Die Repräsentation für Matrizen wird zunächst nur für eine einzige Funktion der VigraPLT benötigt, nämlich die affine Transformation eines Bildes. Dennoch ist diese Erweiterung ist deshalb wichtig, um genau diese Funktion benutzbar zu machen.

Matrizen werden genauso wie Bilder repräsentiert, mit dem einzigen Unterschied, dass sie Gleitkommazahlen mit doppelter Genauigkeit als Wertebereich verwenden. Dies bietet sich an, um die Rechengenauigkeit zu erhöhen. Außerdem sind Matrizen meist deutlich „kleiner“ als Bilder, weshalb dies für diesen Fall beim Speicherverbrauch in einem akzeptablem Rahmen bleibt.

(make-matrix r c . init-val) → image

Bedingungen:

r, c: Ganzzahlen

init-val: Gleitkommazahl (doppelte Genauigkeit)

Beschreibung:

Erstellt eine neue Matrix mit r Zeilen (= rows) und c Spalten (=columns). Falls angegeben, wird die neue Matrix mit dem Wert init-val gefüllt.

(matrix-data matrix) → cvector

Bedingungen:

matrix: matrix

Beschreibung:

Gibt den eindimensionalen Vektor der Werte aller Matrixelemente zurück.

(matrix-rows matrix) → Ganzzahl

Bedingungen:

matrix: matrix

Beschreibung:

Gibt die Anzahl der Zeilen der Matrix zurück.

(matrix-cols matrix) → Ganzzahl

Bedingungen:

matrix: matrix

Beschreibung:

Gibt die Anzahl der Spalten der Matrix zurück.

(matrix-ref matrix r c) → Gleitkommazahl (doppelte Genauigkeit)

Bedingungen:

matrix: matrix

r, c: Ganzzahlen, wobei $0 \leq r <$ (matrix-rows matrix)
und $0 \leq c <$ (matrix-cols matrix)

Beschreibung:

Gibt den Wert der Matrix in Zeile r und Spalte c zurück.

(matrix-set! matrix r c val) → void

Bedingungen:

matrix: matrix

r, c: Ganzzahlen, wobei $0 \leq r <$ (matrix-rows matrix)
und $0 \leq c <$ (matrix-cols matrix)

val: Gleitkommazahl

Beschreibung:

Setzt den Wert der Matrix in Zeile r und Spalte c auf den übergebenen Wert.

WARNUNG:

Dieses „Umbiegen“ des Matrixelements entspricht nicht dem funktionalen Paradigma und kann zu ungewollten Seiteneffekten führen!

(copy-matrix matrix) → matrix

Bedingungen:

matrix: matrix

Beschreibung:

Gibt eine Kopie der übergebenen Matrix zurück.

(matrix->list matrix) → list

Bedingungen:

matrix: matrix

Beschreibung:

Wandelt eine Matrix in eine Liste von Spaltenlisten um und gibt diese zurück.

(list->matrix lst) → matrix

Bedingungen:

lst: Eine Liste von Listen gleicher Länge

Beschreibung:

Inverse Funktion zu matrix->list, die eine Liste von Listen wieder in eine Matrix umwandelt.

(matrix-mult mat1 mat2)→ matrix

Bedingungen:

mat1, mat2: matrix

Beschreibung:

Multipliziert zwei Matrizen (gemäß Matrixmultiplikation) miteinander.

3 Laden und Speichern von Bildern

Generell können mit der VigraPLT zurzeit nur Grauwertbilder eingelesen und verarbeitet werden. Sollte es nötig werden, so müssten die Farb-Funktionalitäten in einer weiteren Version hinzugefügt werden. Daher muss das zu ladende Bild auch ein Grauwertbild sein.

Die unterstützten Formate sind daran gebunden, mit welchen Einstellungen die VIGRA auf dem System kompiliert wurde. Sind alle benötigten Bibliotheken vorhanden gewesen, wovon hier ausgegangen wird, so sind folgende Formate für das Lesen und Schreiben von Bildern geeignet:

- Windows BMP
- GIF
- JPEG
- PNG, PNM
- Sun Raster
- TIFF (including 32bit integer, float, and double)
- Khoros VIFF
- HDR (high dynamic range)

Zum Einlesen und Abspeichern bietet die VigraPLT folgende Funktionen (implementiert in der Datei „vigraplt.impex.ss“):

(loadimage filename) → image

Bedingungen:

filename: String

Beschreibung:

Liest das Bild, gegeben durch den Dateinamen, ein und gibt es als image zurück. Falls das Bild nicht eingelesen werden konnte, wird eine Fehlermeldung ausgegeben. Sowohl absolute als auch relative Pfade (zur VigraPLT) sind erlaubt.

(saveimage image filename)

Bedingungen:

image: image

filename: String

Beschreibung:

Speichert das Bild auf der Festplatte ab.

WARNUNG:

Ist es bereits vorhanden, wird es ohne Vorwarnung überschrieben!

4 Bildverarbeitung mit VigraPLT

Die in diesem Abschnitt beschriebenen Funktionen sind direkt für die Bildverarbeitung anzuwenden. Die Kapitel 4.1 – 4.4 beschreiben Algorithmen die direkt mit Hilfe der VIGRA_C Funktionen und damit sehr schnell ausgeführt werden. In Abschnitt 4.5 sind Scheme-basierte Algorithmen beschrieben, die beim Implementieren eigener funktionaler Algorithmen helfen können, aber von der Geschwindigkeit nicht mit den C-Methoden mithalten können. Alle hier vorgestellten Funktionen f existieren auch mit Namen f-band, und lassen sich so auch auf einzelne Bildkanäle anwenden.

4.1 Filtern

Die Operationen, die in diesem Kapitel zusammengefasst sind, finden sich in der Datei „vigraplt.filters.ss“

(convolveimage image kernel_matrix) → image

Bedingungen:

image: image

kernel_matrix:Faltungskern (als zwei-dimensionale Matrix)

Beschreibung:

Führt eine diskrete Faltung des Bildes mit einem Kern durch.

(separableconvolveimage image kernel_matrix_h kernel_matrix_v) → image

Bedingungen:

image: image

kernel_matrix_h: Faltungskern für horizontale Faltung (Matrix d. Größe [*,1])

kernel_matrix_v: Faltungskern für vertikale Faltung (Matrix d. Größe [1,*])

Beschreibung:

Führt eine diskrete separierbare Faltung des Bildes mit zwei Kernen durch.

(gsmooth image sigma) → image

Bedingungen:

image: image

sigma: Gleitkommazahl

Beschreibung:

Führt eine Gaußsche Glättung des Bildes mit einer Stadardabweichung von sigma durch.

(gaussiangradient image sigma) → '(image_x image_y)

Bedingungen:

image: image

sigma: Gleitkommazahl

Beschreibung:

Berechnet den Gauß'schen Gradienten des Bildes. Dafür wird das Bild mit einer einfach abgeleiteten Gausskurve mit einer Stadardabweichung von sigma gefaltet. Die partiellen Ableitungen in x- und y-Richtung werden als Liste von zwei Bildern zurückgegeben.

(ggradient image sigma) → image

Bedingungen:

image: image

sigma: Gleitkommazahl

Beschreibung:

Erstellt ein Gradientenbetragsbild des Bildes. Dafür wird das Bild mit einer einfach abgeleiteten Gausskurve mit einer Stadardabweichung von sigma gefaltet.

(laplacianofgaussian image scale) → image

Bedingungen:

image: image

scale: Gleitkommazahl

Beschreibung:

Führt eine Laplacian of Gaussian (LoG) Transformation des Bildes auf der Skala scale durch

(gsharpening image sharpening_factor scale) → image

Bedingungen:

image:	image
sharpening_factor, scale:	Gleitkommazahl

Beschreibung:

Führt eine Gaußsche Schärfung des Bildes durch

(sharpening image sharpening_factor) → image

Bedingungen:

image:	image
sharpening_factor:	Gleitkommazahl

Beschreibung:

Führt eine einfache Schärfung des Bildes durch

(nonlineardiffusion image edge_threshold scale) → image

Bedingungen:

image:	image
edge_threshold, scale:	Gleitkommazahl

Beschreibung:

Führt eine nichtlineare Filterung des Bildes aus, um z.B. nicht-additives Rauschen zu entfernen.

(distancetransform image background_label norm) → image

Bedingungen:

image:	image
background_label:	Gleitkommazahl
norm:	Ganzzahl im Intervall [0,1,2]

Beschreibung:

Führt die Distanztransformation eines Bild durch. Mit „norm“ kann die zu verwendende Distanz-Metrik für die Transformation angegeben werden (0=Schachbrett-, 1=Manhattan- und 2=Euklidische-Distanz).

(hessianmatrixofgaussian image scale) → '(image_xx image_xy image_yy)

Bedingungen:

image: image

scale: Gleitkommazahl

Beschreibung:

Berechnet die Komponenten der Hessischen Matrix auf der Gauß'schen Skala scale und gibt diese als Liste von drei Bildern zurück.

(structuretensor image inner_scale outer_scale) → '(image_xx image_xy image_yy)

Bedingungen:

image: image

inner_scale, outer_scale: Gleitkommazahl

Beschreibung:

Berechnet die Komponenten des Strukturtensors auf innerer Skala inner_scale und äußerer Skala outer_scale und gibt diese als Liste von drei Bildern zurück.

(boundarytensor image scale) → '(image_xx image_xy image_yy)

Bedingungen:

image: image

scale: Gleitkommazahl

Beschreibung:

Berechnet den Boundary Tensor auf der Gauß'schen Skala scale und gibt diesen als Liste von drei Bildern zurück.

(boundarytensor1 image scale) → '(image_xx image_xy image_yy)

Bedingungen:

image: image

scale: Gleitkommazahl

Beschreibung:

Berechnet den Boundary Tensor ohne Antwort nullter Ordnung auf der Gauß'schen Skala scale und gibt diesen als Liste von drei Bildern zurück.

(tensoreigenrepresentation tensor)

→ '(tensor_largeEV tensor_smallEV tensor_angleEV)

Bedingungen:

tensor: Dreielementige Liste '(image_xx image_xy image_yy)

Beschreibung:

Berechnet den größeren Eigenwert tensor_largeEV, den kleineren Eigenwert tensor_smallEV und den Winkel der Hauptachse des Tensors tensor_angleEV.

(tensortrace tensor) → tensor_trace_img

Bedingungen:

tensor: Dreielementige Liste '(image_xx image_xy image_yy)

Beschreibung:

Berechnet die Spur der Tensor-Matrix (entspricht image_xx + image_yy).

(tensortoedgecorner tensor)

→ '(tensor_edge_x tensor_edge_y tensor_corners)

Bedingungen:

tensor: Dreielementige Liste '(image_xx image_xy image_yy)

Beschreibung:

Berechnet die Kantenstärke der Tensor-Matrix in x- und y-Richtung (tensor_edge_x, tensor_edge_y) sowie die Eckenstärke tensor_corners.

(hourglassfilter tensor sigma rho) → tensor

Bedingungen:

tensor: Dreielementige Liste '(image_xx image_xy image_yy)

sigma, rho: Gleitkommazahlen

Beschreibung:

Filtert einen Tensor mittels des Hourglass-Filters auf Gauß'scher Skala sigma und unter Berücksichtigung des Öffnungswinkels rho.

4.2 Bildtransformationen

Die Operationen, die in diesem Kapitel zusammengefasst sind, finden sich in der Datei „vigraplt.imgproc.ss“

(rotateimage image angle resize_mode) → image

Bedingungen:

image: image

angle: Gleitkommazahl

resize_mode: Ganzzahl im Intervall [0,1,2,3,4]

Beschreibung:

Dreht ein Bild um den angegebenen Winkel gegen den Uhrzeigersinn. Dabei kann zusätzlich festgelegt werden, wie das Resampling des Bildes passieren soll:

0 = Nächster Nachbar Interpolation

1 = Bilineare Interpolation

2 = Biquadratische Interpolation

3 = Bikubische Interpolation

4 = Interpolation durch Spline der Ordnung 4.

(affinewarpimage image affinematrix resize_mode) → image

Bedingungen:

image: image

affinematrix: matrix (3x3)

resize_mode: Ganzzahl im Intervall [0,1,2,3,4]

Beschreibung:

Führt eine Affine Transformation auf ein Bild aus. Dabei kann zudem festgelegt werden, wie das Resampling des Bildes passieren soll:

0 = Nächster Nachbar Interpolation

1 = Bilineare Interpolation

2 = Biquadratische Interpolation

3 = Bikubische Interpolation

4 = Interpolation durch Spline der Ordnung 4.

(reflectimage image reflect_mode) → image

Bedingungen:

image: image

reflect_mode: Ganzzahl im Intervall [1,2,3]

Beschreibung:

Reflektiert das Bild horizontal, falls reflect_mode = 1,
vertikal, falls reflect_mode = 2,
um beide Achsen, falls reflect_mode =3.

(fouriertransform image) → '(real_part_image imaginary_part_image)

Bedingungen:

image: image

Beschreibung:

Führt eine Fourier Transformation auf dem Bild aus und gibt eine Liste bestehend aus real- und komplexwertigen Anteil zurück.

WARNUNG:

Dies erfordert das Vorhandensein der FFTW Bibliothek auf dem System, da die schnelle Fourier Transformation dieser Bibliothek verwendet wird.

4.3 Morphologische Operationen

Die Operationen, die in diesem Kapitel zusammengefasst sind, finden sich in der Datei „vigraplt.morphology.ss“. Streng genommen könnte man hierunter auch die Distanztransformation zählen, allerdings ist sie aus historischen Gründen in diesem Handbuch wie auch in der VigraPLT in dem Filter-Kapitel beschrieben.

(erodeimage image radius) → image

Bedingungen:

image: image

radius: Ganzzahl

Beschreibung:

Führt die morphologische Operation der Erosion auf dem Bild aus. Dabei kann der Radius / die Stärke der Erosion mit angegeben werden.

(dilateimage image radius) → image

Bedingungen:

image: image

radius: Ganzzahl

Beschreibung:

Führt die morphologische Operation der Dilatation auf dem Bild aus. Dabei kann der Radius / die Stärke der Dilatation mit angegeben werden.

(openingimage image radius) → image

Bedingungen:

image: image

radius: Ganzzahl

Beschreibung:

Führt die morphologische Operation des Öffnens auf dem Bild aus. Dies entspricht einer Erosion gefolgt von einer Dilatation. Dabei kann der Radius / die Stärke der einzelnen Operationen mit angegeben werden.

(closingimage image radius) → image

Bedingungen:

image: image

radius: Ganzzahl

Beschreibung:

Führt die morphologische Operation des Schließens auf dem Bild aus. Dies entspricht einer Dilatation gefolgt von einer Erosion. Dabei kann der Radius / die Stärke der einzelnen Operationen mit angegeben werden.

4.4 Segmentierung

Die Operationen, die in diesem Kapitel zusammengefasst sind, finden sich in der Datei „vigraplt.segmentation.ss“. Dazu gehören sowohl Kanten-basierte als auch Regionen-ortierte Verfahren. Ausserdem bietet die VigraPLT noch die Möglichkeiten, Regionen aus Binärbildern zu extrahieren und CrackEdge-Bilder zu erstellen.

(labelimage image) → image

Bedingungen:

image: image

Beschreibung:

Diese Funktion führt ein Labeling aus. Dazu werden 4-zusammenhängende Regionen gesucht. Diese bekommen im Ergebnisbild einen eindeutigen Wert zugewiesen.

Beispiel:

0	0	0	0		0	0	0	0
1	0	0	1		1	0	0	2
0	0	0	0	→	0	0	0	0
1	0	1	1		3	0	4	4
1	0	1	1		3	0	4	4

(watersheds image) → image

Bedingungen:

image: image

Beschreibung:

Diese Funktion führt eine Wasserscheiden-Transformation auf dem Bild aus. Zurückgeliefert wird ein gelabeltes Regionen Bild, das das Ergebnis der Segmentierung darstellt.

(cannyedgeimage image scale gradient_threshold mark) → image

Bedingungen:

image: image

scale, gradient_threshold, mark: Gleitkommazahlen

Beschreibung:

Diese Funktion führt einen Canny-Kantenfinder auf geg. Skala des Bildes aus. Zurückgeliefert wird ein Bild, in dem alle Canny-Edgel mit „mark“ markiert worden sind.

(differenceofexponentialedgeimage image scale gradient_threshold mark) → image

Bedingungen:

image: image

scale, gradient_threshold, mark: Gleitkommazahl

Beschreibung:

Diese Funktion führt einen DoG-Kantenfinder auf geg. Skala des Bildes aus. Zurückgeliefert wird ein Bild, in dem alle gefundenen Kantenpixel mit „mark“ markiert worden sind.

(regionimagetocrackedgeimage image mark) → image

Bedingungen:

image: image

mark: Gleitkommazahl

Beschreibung:

Diese Funktion erstellt aus einem gegeben Regionenbild mit Größe (w, h) ein Bild der Größe (2*w-1, 2*h-1) in dem Kanten zwischen den Regionen explizit mit „mark“ eingezeichnet wurden.

Beispiel:

				1	1	1	#	2
1	1	2		1	#	#	#	2
1	3	2	→	1	#	3	#	2
4	3	3		#	#	3	#	#
				4	#	3	3	3

4.5 Funktionale Bildverarbeitung

Die Operationen, die in diesem Kapitel zusammengefasst sind, finden sich in der Datei „vigraplt.helpers.ss“. Sie sollen den Einstieg in die funktionale Bildverarbeitung erleichtern und werden sicher in Zukunft durch zahlreiche weitere Funktionen ergänzt.

(image-map func image . more images) → image

Bedingungen:

func: Funktion mit Signatur: Gleitkommazahl → Gleitkommazahl

image: image (beliebig viele)

Beschreibung:

Diese Funktion ist analog zu dem in Scheme vorhandenen „map“ für Listen. Sie wendet eine gegebene Funktion auf den Wert jedes Pixels der übergebenen Bilder an und speichert das Ergebnis in dem zurückgegebenen Bild. Dabei wird das erste ursprüngliche Bild nicht verändert. Die Funktion wird kanalunabhängig angewendet.

(image-map! func image . more images) → image

Bedingungen:

func: Funktion mit Signatur: Gleitkommazahl → Gleitkommazahl

image: image

Beschreibung:

Diese Funktion ist analog zu dem in Scheme vorhandenen „map!“ für Listen. Sie wendet eine gegebene Funktion auf den Wert jedes Pixels der übergebenen Bilder an und speichert das Ergebnis in dem ersten übergebenen Bild, also in-place!

WARNUNG:

Hierbei wird das erste ursprüngliche Bild verändert, was zu Seiteneffekten führen kann, und vermieden werden sollte!

(image-reduce func image seed) → image

Bedingungen:

func: Funktion mit Signatur: Gleitkommazahl → Gleitkommazahl

image: image

seed: Gleitkommazahl

Beschreibung:

Diese Funktion ist analog zu dem in Scheme vorhandenen „reduce“ für Listen. Sie reduziert das Bild mittels der gegebenen Funktion und eines „seed“-Wertes auf einen Wert. Dabei wird das ursprüngliche Bild nicht verändert.

(image-for-each-index func image)

Bedingungen:

func: Funktion mit Signatur: Ganzzahl, Ganzzahl, Ganzzahl→ void

Bedeutung: (x y Kanal-Nr)

image: image

Beschreibung:

Diese Funktion ruft die übergebene Funktion für jede Pixelposition und jedem Kanal, die es in dem übergebenen Bild gibt, auf.

5 Anzeige von Bildern

Zur Anzeige der Ergebnisbilder ist es selten sinnvoll, diese immer und immer wieder abzuspeichern und dann in einem externen Bildbetrachter anzuschauen. Stattdessen bietet die VigraPLT einen einfachen Bildbetrachter, der die Bilder die aktuell in Scheme vorhanden sind anzeigen kann.

Dieser muss leider einigen Overhead erzeugen, um das Bild mit den GUI-Mitteln von DrScheme anzuzeigen, weshalb es bei großen Bildern vermutlich recht lange dauern wird sie zu laden. Generell stellen große Bilder aber kein Problem dar, da der Viewer Scrollbalken einblendet sobald das Bild größer als die Auflösung des Monitors ist.

Eine Zoomfunktion ist leider zur Zeit noch nicht enthalten. Dafür wird aber der Wert des aktuellen Pixels in der Statusleiste angezeigt, wenn man mit der Maus über das Bild fährt.

Zum Aufrufen des Viewers genügt eine Funktion (implementiert in „vigraplt.viewer.ss“):

(show-image image . window-title) → void

Bedingungen:

image: image

window-title: optional: String

Beschreibung:

Diese Funktion zeigt das Bild mit dem Viewer an. Falls gewünscht, kann dem Fenster ein String als Titel zugewiesen werden.

6 Beispiele

In diesem Kapitel sind einige kleine Bespiele erläutert, die den Einstieg in die Programmierung mit VigraPLT erleichtern sollen. Die Beispiele sind von einfach bis anspruchsvoller geordnet und sicherlich nur als kleiner Einstieg in das Themengebiet zu verstehen.

6.1 „Hello Image“

Dieses Beispiel soll zunächst darstellen, wie man ein Bild laden, anzeigen, das Maximum aller Intensitätswerte herausfinden und unter einem anderen Format wieder abspeichern kann. Als erstes müssen wir die VigraPLT einbinden:

```
> (require vigraplt/vigraplt)
```

Jetzt können wir ein Bild laden:

```
> (define myImage (loadimage
                (build-path vigraplt-path "images/blox.gif")))
```

Danach ist das Bild in myImage vorhanden! Wir können es uns also ansehen, zum Beispiel mit dem eingebauten Viewer:

```
> (show-image myImage „Blox-Bild“)
```

Der Viewer sollte jetzt erscheinen und das Bild anzeigen. Wenn wir uns nun für den maximalen Intensitätswert interessieren, so könnten wir nach dem hellsten Fleck im Viewer suchen, und mit der Maus darüber fahren, um den Intensitätswert zu erhalten.

Aber ist der per Hand ermittelte Wert tatsächlich der höchste aller vorkommenden Werte? Um dies zu prüfen kann z.B. die einbaute image-reduce Funktion verwendet werden:

```
> (define image-maximum (image-reduce  max myImage 0.0))
```

Wenn wir nun

```
> image-maximum
```

eingeben, erhalten wir den höchsten aller Intensitätswerte. Soll das unveränderte Bild abgespeichert werden, genügt:

```
> (saveimage myImage
                (build-path vigraplt-path "images/blox.gif"))
```

Damit wird das Bild automatisch korrekt im PNG-Format abgespeichert, obwohl wir es als GIF gelesen hatten.

6.2 Filtern mit Gauß

Dieses Beispiel soll zeigen, wie man die Bildverarbeitungs-Funktionen der VigraPLT einsetzt. Wir führen dazu einige Gaußsche Filter aus.

Als erstes müssen wir die VigraPLT einbinden:

```
> (require vigraplt/vigraplt)
```

Jetzt können wir ein Bild laden:

```
> (define myImage (loadimage
                (build-path vigraplt-path "images/blox.gif")))
```

und einen ersten Filter durchlaufen lassen, um das Bild zu glätten:

```
> (define smoothImage (gsmooth myImage 1.0))
```

Um sich das Ergebnis anzuzeigen, kann wieder der Viewer verwendet werden:

```
> (show-image smoothImage „Blox-Bild geglättet mit sigma=1.0“)
```

Falls man nicht immer ein neues Symbol definieren möchte, z.B. um sich nur kurz Zwischenergebnisse anzuschauen, dann funktioniert selbstverständlich auch:

```
> (show-image  (gsmooth myImage 1.0)
               „Blox-Bild geglättet mit sigma=1.0“)
```

Um ein Gefühl für die Ergebnisse der Algorithmen zu bekommen, bietet es sich an, die verschiedenen Verfahren einmal mit unterschiedlichen Parametern aufzurufen, und sich anschließend die Auswirkungen der Parameterwahl zu erklären.

So könnte man sich zum Beispiel auch anschauen, welches Ergebnisbild

```
> (define mysteriousImage (ggradient myImage 1.0))
```

liefert, und wie sich die Änderung des Parameters hier bemerkbar macht. Die Ergebnisse oder Zwischenbilder lassen sich jederzeit auch abspeichern, z.B. mit:

```
> (saveimage mysteriousImage
          (build-path vigraplt-path „images/myterious.png“))
```

6.3 Wasserscheiden-Segmentierung

Dieses Beispiel soll zeigen, wie die Wasserscheiden-Transformation, die in der VigraPLT enthalten ist, als Segmentierungsmethode eingesetzt werden kann.

Dafür müssen wir als erstes die VigraPLT einbinden:

```
> (require vigraplt/vigraplt)
```

Jetzt können wir ein Bild laden:

```
> (define myImage (loadimage
                (build-path vigraplt-path "images/blox.gif")))
```

Wir haben für dieses Beispiel ein Grauwert-Bild gewählt, da dieses in der weiteren Verarbeitung einfacher zu handhaben ist als ein Farbbild.

Dieses werden wir nun mithilfe der Wasserscheiden-Transformation segmentieren. Dabei gibt es jedoch zu beachten, dass die eigentliche Transformation nicht auf dem Bild selbst, sondern auf einem Gradientenbild ausgeführt werden sollte. Daher erzeugen wir das Regionenbild mit:

```
> (define regionImage (watersheds (ggradient myImage 0.7)))
```

In dem so erzeugten Regionen-Bild sind die Regionen einfach von oben nach unten durchnummeriert. Öffnet man es im Viewer mit:

```
> (show-image regionImage)
```

Dann sieht man kaum mehr als einen Farbverlauf, der durch oben beschriebene Nummerierung zustande kommt. Um das Ergebnisbild anschaulicher zu gestalten, könnte man nun zum Beispiel Kanten zwischen den Regionen explizit einzeichnen.

Dies würde einer CrackEdge-Repräsentation entsprechen, die glücklicherweise bereits in der VigraPLT eingebaut ist:

```
> (show-image (regionimagetocrackedgeimage regionImage 0.0))
```

Sehr viel schöner wäre es natürlich, wenn die jeweiligen Regionen mit dem Mittelwert aller enthaltenen Pixelintensitäten des ursprünglichen Bildes gefüllt wären. Dazu müssen die Intensitätswerte Regions-weise addiert und die Größe der Region gespeichert werden. Zunächst finden wir die Anzahl der Regionen heraus, und erstellen uns Vektoren für die Regionsstatistiken:

```
> (define region_count (inexact->exact
                    (+ (car (image-reduce max regionImage 0))
                            1)))
> (define region_sizes  (make-vector region_count))
> (define region_colors (make-vector region_count))
```

Auch wenn dies nicht funktional geschieht, so entscheiden wir uns hier aus Gründen der Effizienz für die Vektor-Manipulations-Variante. Leider muss man in der Bildverarbeitung oft Kompromisse eingehen, möchte man funktional programmieren.

Es wird jetzt für jeden Pixel im Regionsbild die Regionsnummer herausgesucht. Dann wird in beiden Vektoren die Statistik der Region aktualisiert, sprich die Größe der Region wird um einen erhöht und der Grauwert aufsummiert:

```
> (image-for-each-index
    (lambda (x y band)
       (let ((region_id (inexact->exact
                           (image-ref regionImage x y band)))
             (region_color (inexact->exact
                              (image-ref myImage x y band))))
         (begin
           (vector-set! region_sizes region_id
                (+ 1 (vector-ref region_sizes region_id)))
           (vector-set! region_colors  region_id
                (+ region_color (vector-ref region_colors
                                    region_id))))))
     regionImage)
```

In den Vektoren stehen jetzt die Intensitätswerte bzw. die Größe der Regionen. Mit einem einfachen Map erzeugen wir daraus ein Regionen-Bild, bei dem jede Region mit dem mittleren Intensitätswert derselben gefüllt ist:

```
> (define meanColorImage
    (image-map
      (lambda (col)
        (let ((color_id (inexact->exact col)))
          (exact->inexact (/ (vector-ref region_colors
                                color_id)
                             (vector-ref region_sizes
                                color_id)))))
      regionImage))
```

Nun haben wir statt der Regions-Nummer ein Bild, in dem jede Region mit dem mittleren Farbwert gefüllt ist. Betrachten wir nun

```
> (show-image
     (regionimagetocrackedgeimage meanColorImage 0.0))
```

so erhalten wir ein ansprechendes Ergebnis! Dieses können wir natürlich auch abspeichern, wenn das gewollt ist:

```
> (saveimage
     (regionimagetocrackedgeimage meanColorImage 0.0)
     (build-path vigraplt-path
          "images/nice-blox-watersheds.gif"))
```