# Bildverarbeitung mit Racket – Teil I

*Bachelor-Praktikum Bildverarbeitung / Spieleprogrammierung*
*SoSe 2012*
*Universität Hamburg / FB Informatik*

## *1. Aufgabe: Mit der Arbeitsumgebung vertraut machen*

a) Ein Bild laden: Damit man ein Bild laden kann, muss man zunächst die VIGRA-Anbindung VigRacket einbinden.

```
(require vigracket)
```

Anschließend kann man ein Bild unter Angabe des Dateinamens laden.
Es sind zwei Bilder mitgeliefert, wir beginnen heute mit dem Blox-Bild:

```
(define img (load-image
          (build-path vigracket-path "images/blox.gif")))
```

b) Ein Bild betrachten: Oft ist es wichtig, ein Bild mit samt seinen Intensitätswerten zu betrachten. Hierzu bietet die VigRacket einen Viewer, den man starten kann:

```
(show-image img „Das gerade geladene Bild“)
```

Lässt man den String weg, erhält das Anzeigefenster keinen Titel.

c) Speichern von Bildern: Hierfür steht eine Funktion `save-image` zur Verfügung. Der Dateiname bestimmt dabei auch den Typ des Bildes. Unterstützt werden z.B. JPG, GIF, TIF, PNG und das BMP-Format.

```
(save-image img „...Dateiname...“)
```

## *2. Aufgabe: Einfache Filteroperationen*

Wir wollen uns nun zwei Filter anschauen, die aufbauend auf Gauß'schen Filterkernen definiert sind. Beide erwarten als Parameter ein Bild und einen weiteren Parameter `scale`, welcher als Gleitkommazahl angegeben werden muss (z.B. `1.0`):

a) Testet, wie sich das Ergebnisbild in Abhängigkeit von dem Parameter `scale` verändert für die Funktion `(gsmooth img scale)`.
Probiert z.B. folgende Werte: `0.5` , `1.0` , `2.0` , `3.0` , `5.0` und `10.0`

b) Analog zu a) allerdings mit der Funktion `(ggradient img scale)`

Tipp: Um die Untersuchung zu beschleunigen, müsst ihr keine Bilder zwischen speichern. `(show-image (gsmooth img scale))` funktioniert selbstverständlich auch...

*PAUSE*

## *3. Aufgabe: Funktionale Bildverarbeitung*

In der funktionalen Programmierung sind Listen in Zusammenhang mit Funktionen höherer Ordnung wie z.B. `map` von großer Bedeutung. So ein `map` gibt es mit VigRacket auch für Bilder! Es heißt `image-map` und verhält sich ganz ähnlich wie das normale `map`. Führt `(map + '(1 2 3) (1 2 3))` zu `'(2 4 6)`, so kann man `image-map` ebenfalls dazu verwendenden zu addieren, subtrahieren etc..

a) Benutzt `image-map` dazu, um ein geglättetes Bild vom ursprünglichen Bild abzuziehen. Probiert dabei unterschiedliche Skalen wie in Aufgabe 2, beobachtet und beschreibt den Effekt, der sich ergibt!

b) `image-map` kann man auch dazu verwenden, um Bilder zu maskieren. Benutzt es zum Schreiben einer Hilfsfunktion, die bei einem gegebenen Grauwert-Bild folgendes tut:

```
Falls der Intensitätswert < Schwelle:
     setze ihn auf 0,
sonst:
     setze ihn auf 1.
```

c) Nummeriert die Objekte, die durch die Schwellwertbildung entstanden sind, mithilfe der Funktion `labelimage`. Diese weist jeder Zusammenhangskomponente im Bild eine Ziffer zu. Verwendet anschließend `image-reduce`, um die Anzahl der Objekte zu bestimmen.

## *Zusatzaufgabe: Bildtransformationen*

Bilder können nicht nur durch Filter verändert werden, man kann auch die Geometrie von Bildern verändern. Einfachste Beispiele für solche Transformationen sind die Rotation sowie die Vergrößerung von Bildern.

a) In vielen Bildverarbeitungsprogrammen wird man beim Vergrößern von Bildern nach der zu verwendenden Interpolationsmethode gefragt. In der VigRacket ist dies ebenfalls der Fall. Probiert verschiedene Interpolationsgrade von `0` bis `2` aus, indem ihr ein Bild auf das doppelte seiner Größe vergrößert. Benutzt hierfür die Funktion:

```
(resizeimage img neue_breite neue_hoehe interpolations_grad)
```

b) Auch bei der Rotation muss interpoliert werden! Warum eigentlich? Probiert auch hier unterschiedliche Winkel (z.B. `45.0` und `90.0`) und verschiedene Interpolationsgrade von `0` bis `2` aus. Benutzt hierfür die Funktion:

```
(rotateimage img grad interpolations_grad)
```

# Bildverarbeitung mit Racket – Teil II

*Bachelor-Praktikum Bildverarbeitung / Spieleprogrammierung*
*SoSe 2012*
*Universität Hamburg / FB Informatik*

## *1. Aufgabe: Von Grauwert- zu Farbbildern*

In Teil I haben wir noch mit Grauwert-Bildern gearbeitet, nun wollen wir uns den Umgang mit Farbbildern anschauen. Dazu vergleichen wir erstmal wie die Bilder in Racket repräsentiert sind, nachdem wir DrRacket gestartet haben:

a) Ein Grauwertbild laden: Damit man ein Bild laden kann, muss man zunächst die VIGRA-Anbindung VigRacket einbinden.

```
(require vigracket)
```

Wir laden heute zwei Bilder, das altbekannte Blox-Bild und ein Farbbild:

```
(define blox (load-image
          (build-path vigracket-path "images/blox.gif")))

(define lenna (load-image
     (build-path vigracket-path "images/lenna_face.png")))
```

b) Zunächst können jetzt beide Bilder betrachtet werden, mit der bekannten Funktion:

```
(show-image lenna „Das neue Bild“)
```

Wir können auch direkt `lenna` oder `blox` eingeben, um zu schauen, wie die Bilder intern repräsentiert sind.

## *2. Aufgabe: Kanalrepräsentation von Farbbildern*

Eine typische Vorgehensweise ist die Aufteilung von Farbbildern in ihre einzelnen Kanäle. Auch die VigRacket funktioniert in vielen Punkten so. Man kann aber auch selbst die enthaltenen Kanäle betrachten:

a) Probiert nun das `lenna` Bild in seine Rot, Grün und Blau-Komponenten zu zerlegen. Hierfür gibt es vorgefertigte Funktionen:

`image->red`, `image->green` und `image->blue`

b) Wie kann man aus diesen Einzelkanalbildern wieder ein Farbbild zusammensetzen? Tipp: Denkt an Listen! Stellt das Originalbild wieder her, und eins mit vertauschen Kanälen (z.B. rot ↔ grün Kanal vertauscht).

Die so erhaltenen Bilder können wie normale Grauwert-Bilder behandelt werden, insbesondere lassen sich alle Operationen von gestern darauf anwenden. Einige probieren wir nun noch einmal auf dem Bild:

c) `(gsmooth img scale)` und `(ggradient img scale)`

Probiert z.B. die Werte: `1.0` und `2.0` für den Parameter `scale`.
Was fällt euch auf?

*PAUSE*

## *3. Aufgabe: Selektive Glättung*

Ähnlich zur gestrigen Aufgabe 3 wollen wir auch für Farbbilder einen Schwellenwert-Filter definieren. Leider können wir dazu im Moment `image-map` nicht so einfach verwenden, da wir drei unterschiedliche Kanäle in unserem `lenna` Bild haben.

a) Wendet die Funktion zur Schwellenwert-Filterung auf den Rotkanal an. Erstellt verschiedene Ergebnisbilder mit Schwellenwerten von: `180.0`, `200.0` und `220.0`

b) Erstellt auf dieser Grauwert-Maske eine Farbwert-Maske, die für jeden Kanal gleich ist. Tipp: Aufgabe 2b von heute!

c) Schreibt unter Zuhilfenahme von `image-map` eine Funktion, die ein Farbbild, eine Maske und eine Glättungsskala als Parameter annimmt. Die Funktion soll Ergebnisbild ein Bild zurückgeben, dass nur an den maskierten Gebieten geglättet worden ist.

Anmerkung: So ähnlich funktionieren die einfachen Gesichtsweichzeichner in Kameras!

## *Zusatzaufgabe 1: Segmentierung von Bildern*

Siehe Referenz-Handbuch zur VigRacket, Beispiel 3, Abschnitt 6.3, Seite 26.

## *Zusatzaufgabe 2: Bildhistogramme*

a) Schreibt – vorerst nur für Grauwert-Bilder – eine Funktion, die die Anzahl der Farbwerte in ein Histogramm überführt (z.B. einen `vector` der Länge `255`).

b) Schreibt eine kleine Plotting-Funktion, der die Histogramme in einem Bild ausgibt, welches ähnlich einem Block-Diagramm die einzelnen Anzahlen auf die Höhe des Bildes skaliert einträgt.