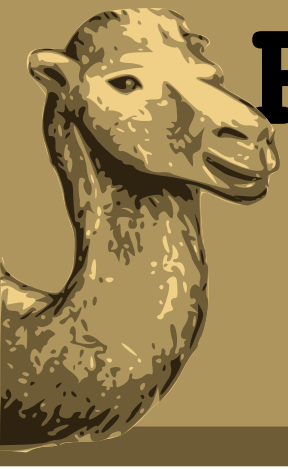
Perl Mongers im Ruhrgebiet



Ruhr.pm

Bildverarbeitung mit PerlMagick

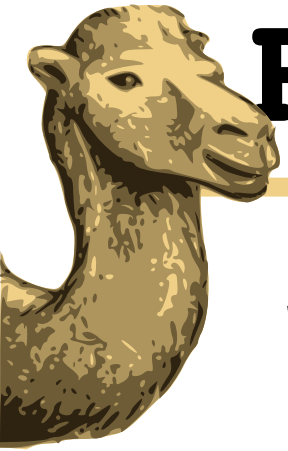
Eine praktische Einfuehrung in Image::Magick

Autor: Veit Wahlich

EMail: veit AT ruhr.pm.org

Datum: 4. Februar 2008

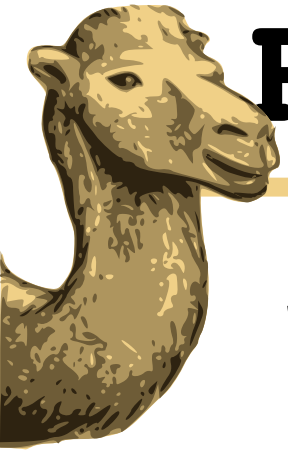
http://ruhr.pm.org/



Ruhr.pm

Was ist Image::Magick?

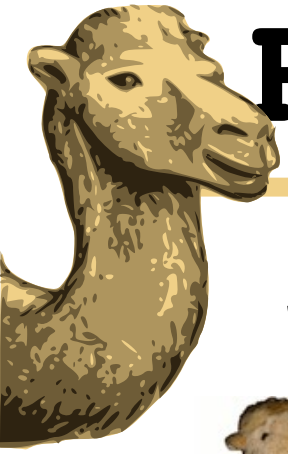
- Binding, das den enormen Funktionsumfang von ImageMagick vollstaendig abbildet
  - optimal fuer die Verarbeitung von Grafiken und Fotos
    - Transformation, Komposition, Beschriftung, Filter
    - Optimierung des Bildinhalts
      - Farben, Gamma, Kontrast, Helligkeit, Schaerfe, ...
    - Optimierung der Dateigroesse
      - Farbtiefe, Dateiformat, Kompressionsgrad, ...
  - aber auch geeignet fuer die Erstellung von Grafiken
    - Funktionen fuer das Zeichnen geometrischer Formen



Ruhr.pm

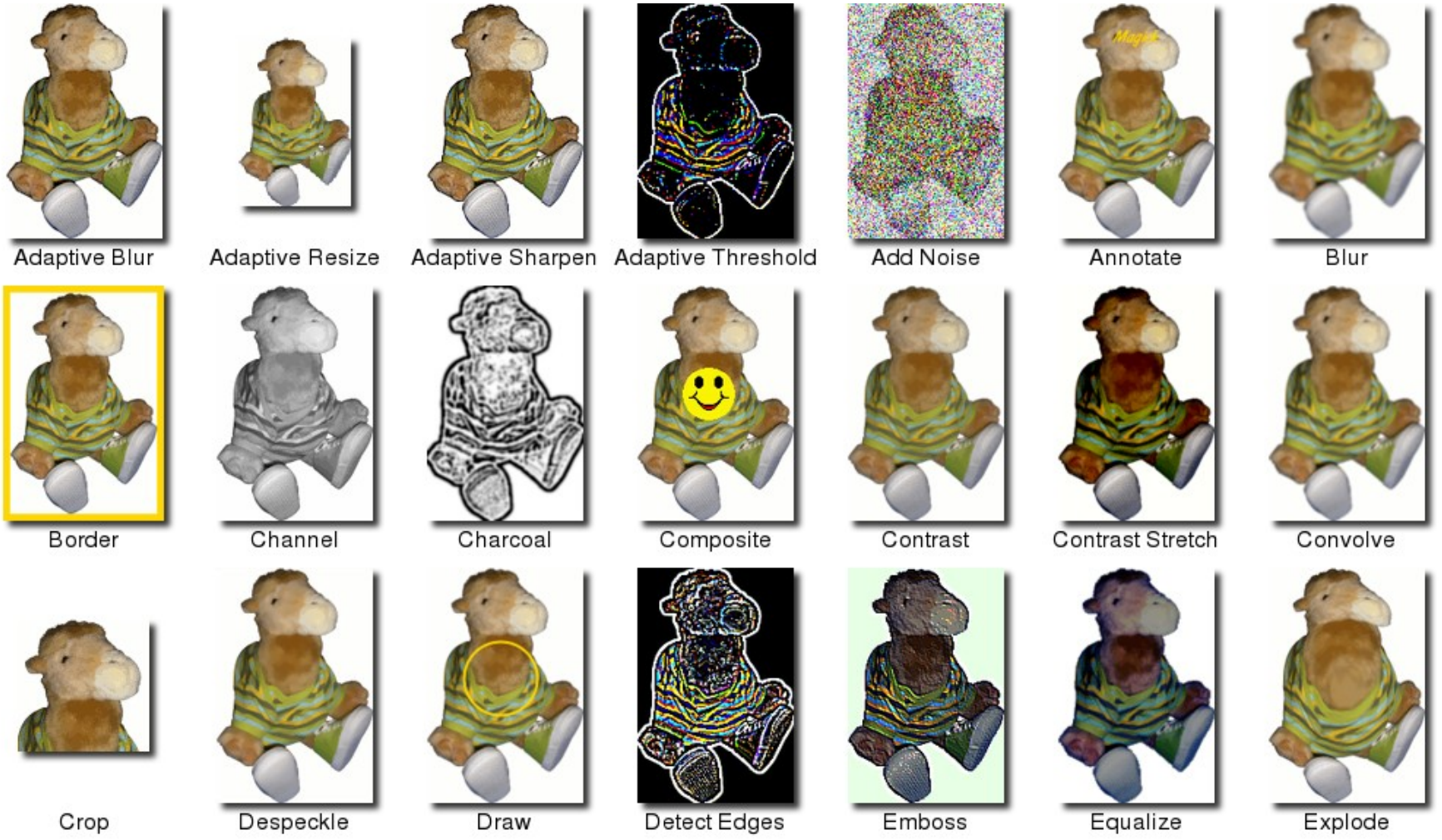
Was ist Image::Magick?

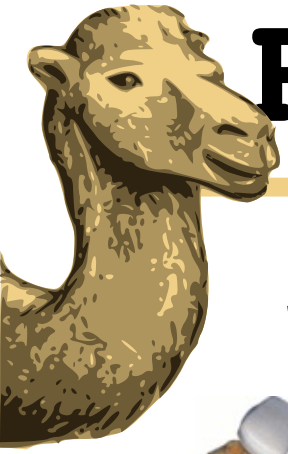
- vollstaendig objekt-orientiertes API
  - Schreibweise der Methoden allerdings nicht Perl-ueblich
  - unterstuetzt neben einfachen Bildern auch ebenenaehnliche Behandlung von Unterbildern
    - kann auch verwendet werden, um Animationen zu erstellen



Ruhr.pm

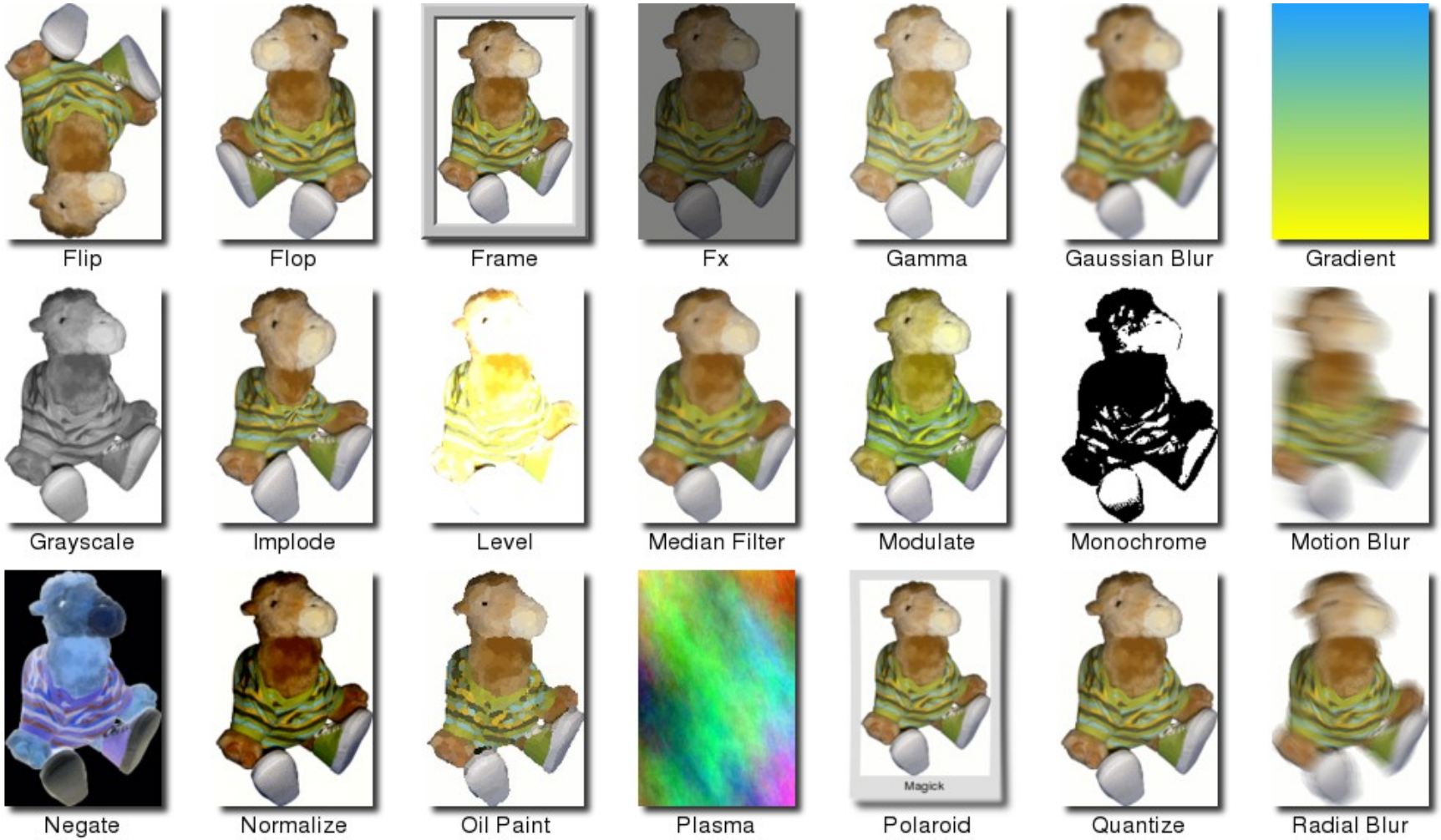
Was kann Image::Magick?

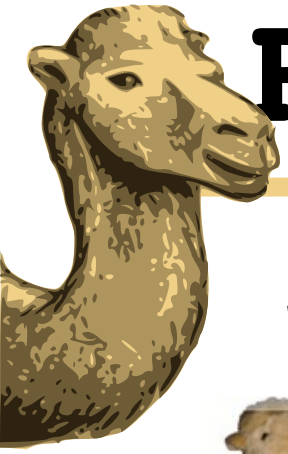




Ruhr.pm

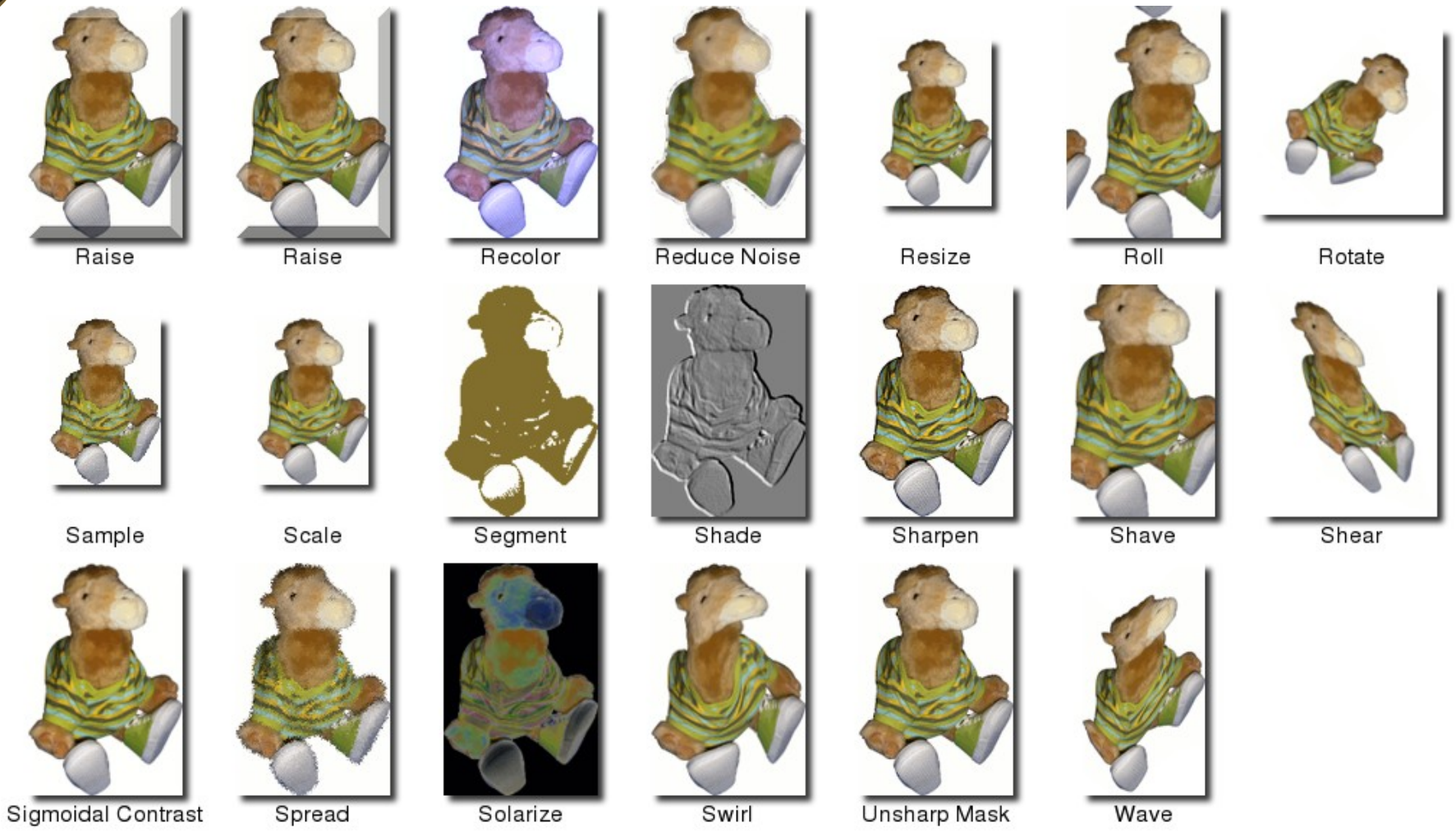
Was kann Image::Magick?

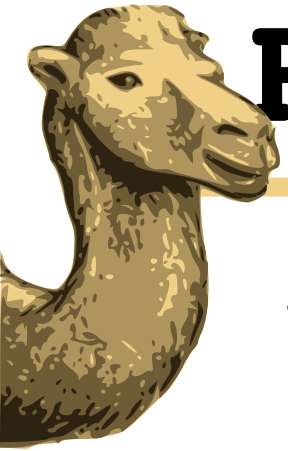




Ruhr.pm

Was kann Image::Magick?





Ruhr.pm

Was kann Image::Magick?

- dieser Vortrag kann nur wenige dieser Methoden behandeln und deren Parameter nicht vollstaendig abdecken
- Dokumentation aller Manipulationsmethoden inkl. aller moeglichen Parameter:
  - http://www.imagemagick.org/script/perl-magick.php#manipulate

Ruhr.pm

Beispiele

- die Funktionsweise von Image::Magick soll hier mit einigen praxisnahen Beispielen demonstriert werden:
  - Kontaktabzug-Indexprint
  - TV-Optimierung
  - Psycho-Lama!

Ruhr.pm

Kontaktabzug-Indexprint

- erzeugt einen Indexprint, wie er von Kontaktabzuegen in der Fotoentwicklung bekannt ist
- stellt wichtige Image::Magick-Methoden vor
  - Read, Montage, Write, Display
- beschreibt interessante Perl-Funktionen, die evtl. nicht jeder regelmaessig verwendet
  - glob, map (Void-Kontext)

Ruhr.pm

Kontaktabzug-Indexprint

Ruhr.pm

Kontaktabzug-Indexprint

```
use strict;
use warnings;
use Image::Magick;


# Dateimaske fuer Eingabe-Bilder.
my $maske   = 'bilder/*.jpg';


# Dateiname fuer die Ausgabedatei.
my $index   = 'indexprint.jpg';


# Max. Groesse der Thumbnails.
my $groesse = '120x120';


# Anzahl Bilder pro Zeile.
my $spalten = 4;
```

Ruhr.pm

Kontaktabzug-Indexprint

```
# Lese anhand der Dateimaske die Dateinamen
# der Bilder ein.

my @bilder = glob($maske);
```

ARRAY = **glob** *STRING*

glob liest alle (nicht versteckten) Verzeichniseintraege, die auf die Maske in *STRING* passen, und speichert sie in *ARRAY*.

```
# Jede Zeile soll $spalten Bilder enthalten,
# bestimme die Anzahl der Zeilen.

my $zeilen = int(@bilder / $spalten)
             + (@bilder % $spalten ? 1 : 0);
```

Berechnet den ganzzahligen Anteil des Quotienten aus der Anzahl der Bilder durch die Anzahl der Bilder pro Spalte.
Addiere ausserdem 1 hinzu, falls die Ganzzahldivision der Anzahl der Bilder durch die Anzahl pro Spalte einen Restwert ergibt.

Ruhr.pm

Kontaktabzug-Indexprint

```
# Erzeuge in $i ein neues Image::Magick-
# Objekt.

my $i = new Image::Magick;



# Lese alle Bilder als Ebenen in $i ein.

$i->Read(@bilder);
```

STRING = $i->**Read** *ARRAY*

Die Methode **Read** liest alle Grafiken aus *ARRAY* als neue Ebenen in das Image::Magick-Objekt `$i` ein.
In *STRING* werden eventuelle Fehlermeldungen gespeichert.

Hier werden alle Dateien aus dem Array `@bilder` eingeladen, eventuelle Fehler werden ignoriert.

Ruhr.pm

Kontaktabzug-Indexprint

```
# Entferne alle Pfadangaben aus den
# Dateinamen.

map{s|^.*/||}@bilder;



# Durchlaufe alle Bilder und gebe jedem
# Bild seinen Dateinamen als Label.

foreach my $j (@{$i}){

  $j->Label(shift(@bilder));

}
```

map *BLOCK ARRAY*

map im Void-Kontext verarbeitet jedes Element aus *ARRAY* ueber `$_` in *BLOCK* und ersetzt das Element in *ARRAY* durch den Rueckgabewert.

Hier wird `$_` durch einen Regexp modifiziert, der alles vom Zeilenanfang bis zum letzten Slash (“/”) durch einen leeren String ersetzt.

Ruhr.pm

Kontaktabzug-Indexprint

```
# Erstelle aus den Ebenen in $i einen
# Indexprint in einem neuen Image::Magick-
# Objekt $k.

my $k = $i->Montage(

    # Ueberschrift fuer den Indexprint.
    title      => $maske,

    # Anordnung der Einzelbilder.
    tile       => $spalten.'x'.$zeilen,

    # Max. Groesse und Abstand der Bilder.
    geometry   => $groesse.'+10+10',

    # Einen 2px breiten Rand um die Bilder.
    border     => 2,

    # Schatten fuer die Bilder.
    shadow     => 1,

  );
```

`$k = $i->`**`Montage`** *`HASH`*

Die Methode **`Montage`** erzeugt aus den Ebenen eines Image::Magick-Objekts ein neues Objekt mit einer Tabelle von Thumbnails.

`Montage` kennt viele Parameter in Form von *`HASH`*, die wichtigsten, `title`, `tile`, `geometry`, `border` und `shadow` werden hier angewendet, das Beispielscript indexprint.pl beschreibt noch einige weitere.

Ruhr.pm

Kontaktabzug-Indexprint

```
# Schreibe den Indexprint in eine
# Grafikdatei.

$k->Write(

    # Dateiname fuer die Ausgabedatei.
    filename => $index,

    # Qualitaets-/Kompressionsstufe.
    quality  => 90

  );
```

STRING = $k->**Write** *HASH*

Die Methode **Write** schreibt den Inhalt von `$k` als Grafikdatei auf den Datentraeger oder auf ein Handle, *STRING* speichert eventuelle Fehler.

Write kann mit vielen Parameter in Form von *HASH* aufgerufen werden, hier werden mit `filename` ein Dateiname und mit `quality` ein Kompressionsgrad angegeben. Das Grafikformat der Ausgabe wird wahlweise durch die Dateiendung oder ein Praefix wie “`png:`” definiert.

Ruhr.pm

Kontaktabzug-Indexprint

```
# Zeige den Indexprint auf dem Bildschirm
# an.

$k->Display();
```

STRING = $k->**Display**

Mit der Methode **Display** wird ein Fenster mit dem (ersten) Bild aus dem Image::Magick-Objekt `$k` auf dem Bildschirm angezeigt. *STRING* kann eventuelle Fehlermeldungen enthalten.

Das Programm wird erst fortgesetzt, wenn das Fenster geschlossen wurde.

Ruhr.pm

TV-Optimierung

- erzeugt aus Bildern in einem Verzeichnis skalierte Versionen, die einen schwarzen Mindestrand besitzen, damit bei der Ausgabe auf Fernsehern (ueber DVD-Player o.ae.) keine Bildteile abgeschnitten werden
- zeigt weitere wichtige Image::Magick-Methoden
  - Set, Scale, Composite
- zeigt eine weitere interessante Perl-Funktion
  - grep

Ruhr.pm

TV-Optimierung

Ruhr.pm

TV-Optimierung

```
use strict;
use warnings;
use Image::Magick;


# Verzeichnis mit Eingabe-Bildern.
my $dir_in           = 'input';


# Verzwichnis fuer Ausgabe.
my $dir_out          = 'output';


# Groesse der Ausgabebilder.
my $groesse_leinwand = '800x600';


# Groesse der eingebetteten Bilder.
my $groesse_bild     = '768x576';


# Ausgabequalitaet/-kompression.
my $qualitaet        = 85;
```

Ruhr.pm

TV-Optimierung

```
# Oeffne das Eingabeverzeichnis in $dh.

opendir(my $dh, $dir_in);


# Lese alle Eintraege im Verzeichnis-Handle
# $dh und filtere unpassende Eintraege aus.

my @bilder = grep{

    # Nur Eintraege, die Dateien sind...
    -f $dir_in.'/'.$_

    # ... und deren Name nur aus alphanum.
    # Zeichen besteht und auf .jpg, .jpeg
    # oder .png endet (case insensitive).
    && /^\w+\.(?:jpe?g|png)$/i

  }readdir($dh);


# Schliesse das Verzeichnis-Handle $dh.

closedir($dh);
```

ARRAY2 = **grep** *BLOCK ARRAY1*

grep im Array-Kontext verarbeitet jedes Element aus *ARRAY1* und speichert nur die Elemente in *ARRAY2*, fuer die *BLOCK* mit der Laufvariable `$_` zu true evaluiert.

Ruhr.pm

TV-Optimierung

```
# Durchlaufe alle Bilder und verarbeite sie.
foreach my $datei (@bilder){

  printf("Verarbeite %s...\n", $datei);


  # Erzeuge ein Image::Magick-Objekt $i.
  my $i = new Image::Magick;


  # $i, setze die Leinwandgroesse und
  $i->Set(size => $groesse_leinwand);


  # Lese eine schwarze Flaeche als Bild ein.
  $i->Read('xc:black');
```

`$i->`**`Set`** *`HASH`*

Die Methode **`Set`** setzt im Image::Magick-Objekt `$i` die Werte der Eigenschaften in *`HASH`*.

Hier wird die Groesse des Bildes bzw. der Leinwand definiert.

`Read` erzeugt mit einem Pseudo-dateinamen aus dem Prefix “`xc:`” und einer Farbdefinition ein Bild mit einer farbigen Flaeche in Bild-groesse.

Ruhr.pm

TV-Optimierung

```
# Erzeuge ein weiteres Image::Magick-
# Objekt $j.

my $j = new Image::Magick;



# Lade lade die zu verarbeitende Datei
# aus dem Eingabeverzeichnis hinein.

$j->Read($dir_in.'/'.$datei);



# Das Bild wird auf die maximal darstell-
# bare Groesse des Fernsehers skaliert.

$j->Scale(geometry => $groesse_bild);
```

`$j->`**`Scale`** *`HASH`*

Die Methode **`Scale`** skaliert das Image::Magick-Objekt `$j` mit den in *`HASH`* uebergebenen Parametern auf eine neue Groesse.

`HASH` enthaelt wahlweise ein Hash-Element mit dem Schluessel `geometry` oder mindestens eines der Elemente `width` und `height`.

Entgegen der Doku sind `geometry` und `width`/`height` nicht aequivalent, da nur `geometry` das Seitenverhaeltnis beruecksichtigt.

Ruhr.pm

TV-Optimierung

```
    # Lege das Bild $j zentriert ueber die
    # schwarze Leinwand in $i.

    $i->Composite(

        # Das einzufuegende Bild.
        image   => $j,

        # Position von $j auf $i zentrieren.
        gravity => 'Center'

      );



    # Schreibe das zusammengefuegte Bild in
    # das Ausgabe-Verzeichnis.

    $i->Write(
        filename => $dir_out.'/'.$datei,
        quality  => $qualitaet
      );


} # Ende foreach $datei (@bilder)
```

`$i->`**`Composite`** *`HASH`*

Die Methode **`Composite`** legt das in *`HASH`* unter dem Schluessel `image` uebergebene Image::Magick-Objekt ueber das Bild im Objekt `$i`.

Weitere wichtige Parameter in *`HASH`* koennen Angaben zur Position mit `x` und `y` sein, oder, wie hier, die Positionierung per `gravity`.

Es koennen aber auch komplexe weitere Eigenschaften definiert werden, wie z.B. betroffene Farbkanaele oder Rotation.

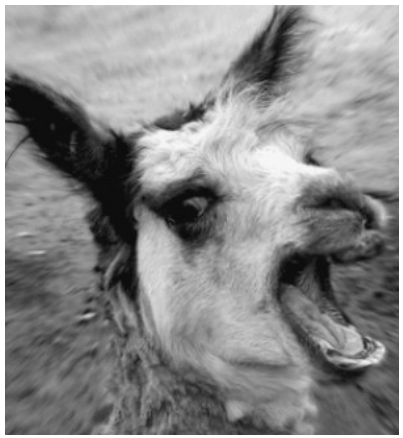
Ruhr.pm

Psycho-Lama!

- generiert aus fiesen Bildern noch fiesere Animationen
- stellt wichtige Image::Magick-Methoden vor
  - Get, Rotate, Scale (alternative Anwendung), Negate, Crop, Write (alternative Anwendung), Animate
- beschreibt Perl-Funktionen, die evtl. nicht jedem bekannt sind
  - map (Array-Kontext)

Ruhr.pm

Psycho-Lama!



Ruhr.pm

Psycho-Lama!

```
use strict;
use warnings;
use Image::Magick;

# Festlegen der Parameter fuer die
# Verarbeitung.

my $conf = {

    # Dateiname der Eingabe-Grafik.
    input  => 'lama.png',

    # Dateiname der auszugebenden Animation.
    output => 'lama.gif',

    # Einzelne Ebenen um -2.5° drehen...
    rotate => -2.5,

    # ... und um 10% vergroessern.
    zoom   => 1.10,

    # Zwischen den Bildwechseln 50ms Pause.
    delay  => 5

  };
```

Ruhr.pm

Psycho-Lama!

```
# Ein neues Image::Magick-Objekt erzeugen.
my $i = new Image::Magick;



# Die Eingabe-Grafik wird 4x geladen.
$i->Read(map{$conf->{input}}(0..3));
```

ARRAY2 = **map** *BLOCK ARRAY1*

map im Array-Kontext verarbeitet jedes Element aus *ARRAY1* ueber `$_` in *BLOCK* und speichert den Rueckgabewert in *ARRAY2*.

Hier wird fuer jedes Element aus dem Array `(0,1,2,3)` lediglich der Inhalt von `$conf->{input}` zurueck gegeben.

Interpolierter Aufruf:

```
$i->Read('lama.png',
'lama.png','lama.png',
'lama.png');
```

Ruhr.pm

Psycho-Lama!

```
# Als Referenzwert fuer die spaetere
# Verarbeitung werden Breite und Hoehe der
# Ebene 0 gespeichert.

my($w,$h) = $i->[0]->Get('width','height');
```

ARRAY2 = $i->**Get** *ARRAY1*

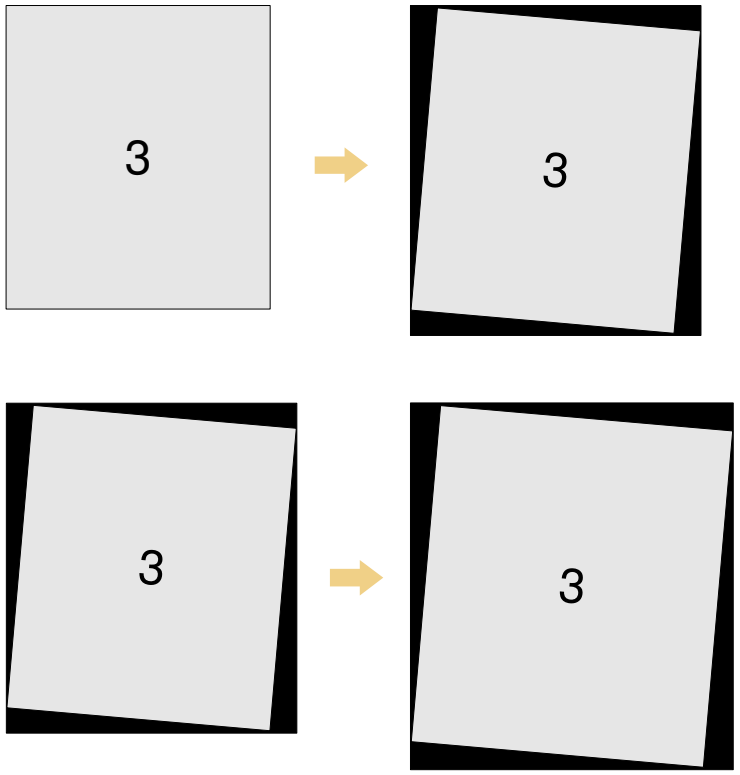
Die Methode **Get** bezieht die Werte der Eigenschaften *ARRAY1* aus dem Image::Magick-Objekt `$i` bzw. hier aus der ersten Ebene (`$i->[0]`) des Objekts und speichert sie in *ARRAY2*.

Hier wird die Breite und Hoehe der ersten Ebene von `$i` ausgelesen und in `$w` und `$h` gespeichert.

Ruhr.pm

Psycho-Lama!

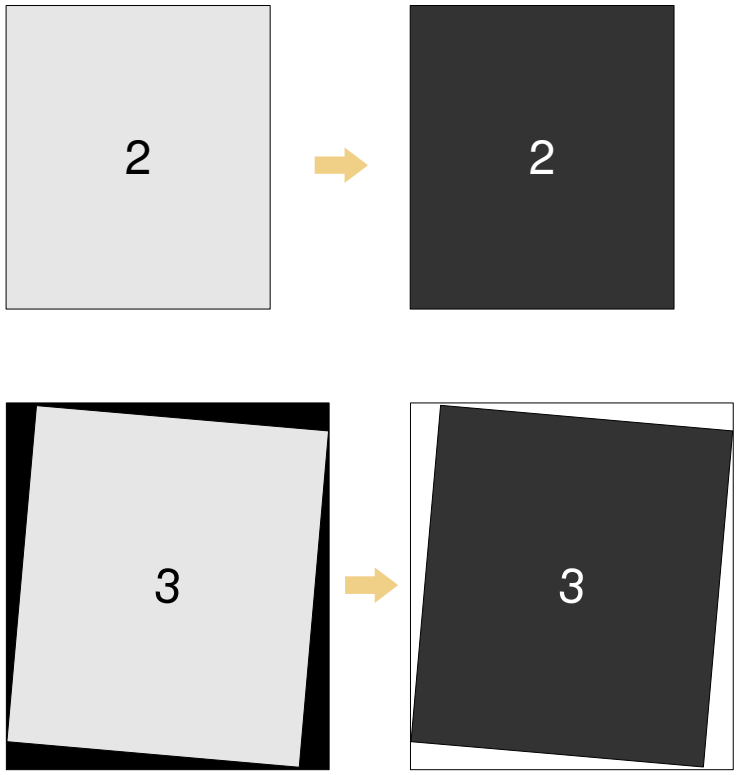
```
# Ebene 1 und 3 werden erst rotiert und
# dann hochskaliert.

for(1,3){

  $i->[$_]->Rotate(
      degrees => $conf->{rotate}
    );




  $i->[$_]->Scale(
      width  => $w*$conf->{zoom},
      height => $h*$conf->{zoom}
    );

}
```



Ruhr.pm

Psycho-Lama!

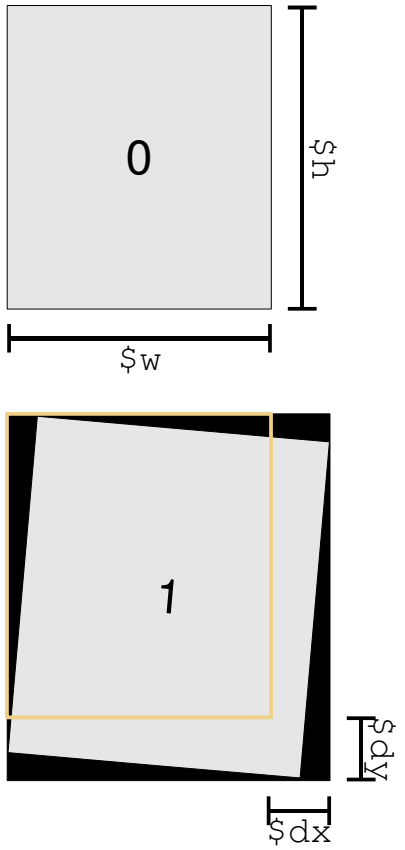
```
# Ebene 2 und 3 in ein Negativ verwandeln.
for(2,3){
  $i->[$_]->Negate();
}
```



Ruhr.pm

Psycho-Lama!

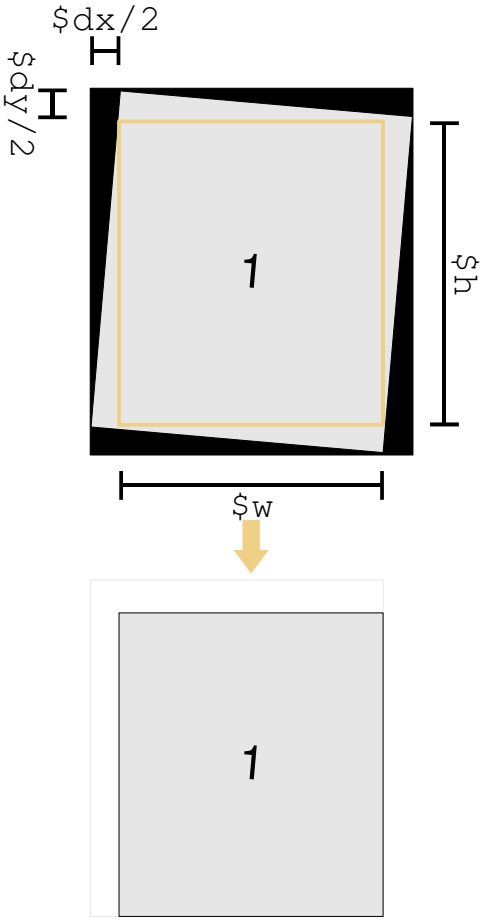
```
# Berechne das Delta zwischen Original-
# groesse (Ebene 0) und neuer Groesse nach
# Rotation und Hochskalierung (Ebenen 1, 3).

my($dx,$dy)= $i->[1]->Get('width','height');
$dx -= $w;
$dy -= $h;
```



Ruhr.pm

Psycho-Lama!

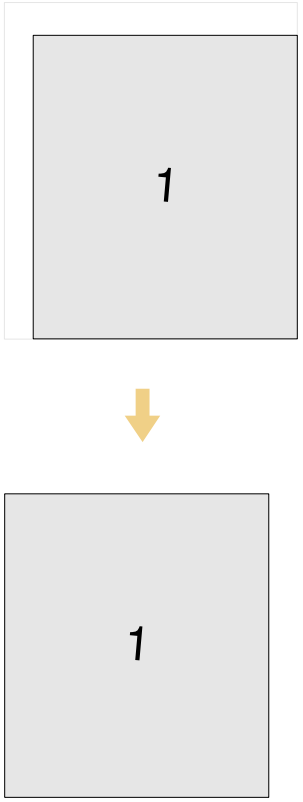
```
# Beschneide die Ebenen 1 und 3 auf
# Originalgroesse, halbiere die Deltas,
# um den Abstand zum oberen linken Rand zu
# bestimmen.

for(1,3){

  $i->[$_]->Crop(
      width  => $w,
      height => $h,
      x      => $dx/2,
      y      => $dy/2
    );

}
```



Ruhr.pm

Psycho-Lama!

```
# Setze die virtuelle Position aller Ebenen
# auf die absolute Position 0,0.
# Setze ausserdem die Zeitspanne zwischen
# den Bildwechseln der Ebenen in der
# Animation (Wert*10ms).

for(0..3){

  $i->[$_]->Set(
      page  => '+0+0',
      delay => $conf->{delay}
    );

}
```



Ruhr.pm

Psycho-Lama!

```
# Schreibe die fertige Animation in eine
# Datei...

$i->Write($conf->{output});
```

STRING2 = $i->**Write** *STRING1*

Hier wird die Methode **Write** mit einer einfacheren Syntax verwendet, in der nur der Dateiname der Ausgabedatei, *STRING1*, uebergeben wird. *STRING2* enthaelt eventuelle Fehlermeldungen.

```
# ... und zeige sie auf dem Bildschirm an.

$i->Animate();
```

STRING = $i->**Animate**

Animate ist das Aequivalent zu **Display** fuer das Anzeigen der Ebenen als Animation statt nur der ersten Ebene als Bild.

Ruhr.pm

Vielen Dank
für Eure Aufmerksamkeit

Ruhr.pm

Links

- PerlMagick (inkl. Image::Magick) und Dokumentation
  - http://www.imagemagick.org/script/perl-magick.php