2.4 Onebit-Operationen (1)

Kombination von Onebit-Bildern

- Onebit-Bilder gleicher Größe können *Pixel für Pixel* kombiniert werden:

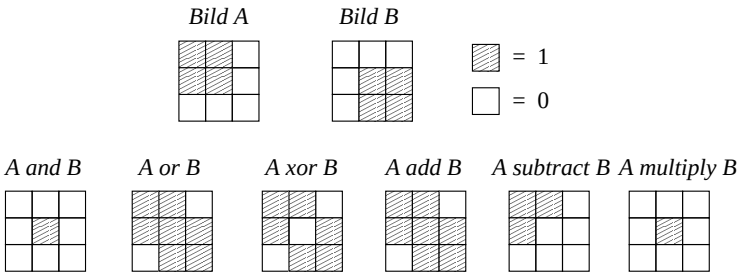
$$h(x,y) = f(x,y) \otimes g(x,y) \quad \text{für alle } x,y$$

wobei $\otimes$ eine bitweise Operation ist

- Funktionen in Gamera:
  - `Image.and_image(Image other)`
    ebenso *or_image()* sowie *xor_image()* (logische Operatoren)
  - `Image.add_images(Image other)`
    ebenso *subtract_images()* sowie *multiply_images()*
    (arithmetische Operatoren; auch auf nicht Onebit-Bildern)

2.4 Onebit-Operationen (2)

- Demonstration der bitweisen Kombinationen:

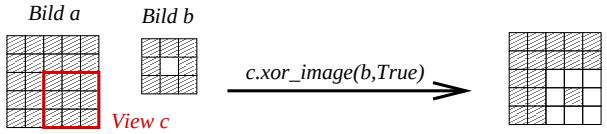


- für Onebit-Bilder gilt:
  - *A or B = A add B*
  - *A and B = A multiply B*

2.4 Onebit-Operationen (3)

- Kombination von Onebit-Bildern *unterschiedlicher* Größe über folgenden "Trick" möglich:
  - erzeuge ein Subimage vom größeren Bild an der Stelle, die mit dem kleineren Bild kombiniert werden soll
  - kombiniere das Subimage mit dem kleineren Bild, wobei der (optionale) Parameter *in_place = True*

```
# a sei 5x5 Bild und b 3x3 Bild
c = a.subimage(Point(2,2),Point(4,4))
c.xor_image(b, in_place=True)  # verändert b "in place"
```



2.4 Onebit-Operationen (4)

Farbliche Hervorhebung von Teilbildern

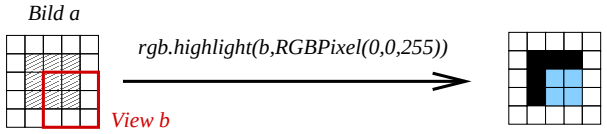
- manchmal möchte man alle Pixel eines Onebit-Bildes farblich in einem anderen Bild darstellen
  - z.B. zur Anzeige von Pixeln, die eine Operation entfernt
  - dazu braucht man ein RGB-Bild, auf das man die Methode *highlight(onebitimg, pixelvalue)* anwendet
  - Beispiel:

```
# a und b seien gleich groß
# markiere alle Pixel rot, die in a sind, aber nicht in b
c = a.subtract_images(b)
rgb = a.to_rgb()
rgb.highlight(c, RGBPixel(255,0,0))
```

2.4 Onebit-Operationen (5)

- *highlight()* geht auch mit Subimage-Views
  - Beispiel:

```
# markiere alle schwarzen Pixel in einem Unterquadrat von a blau
b = a.subimage(Point(2,2),Point(4,4))
rgb = a.to_rgb()
rgb.highlight(b, RGBPixel(0,0,255))
```



  - wichtigste Anwendung:
    hervorheben von Connected Components (s.u.)

2.4 Onebit-Operationen (6)

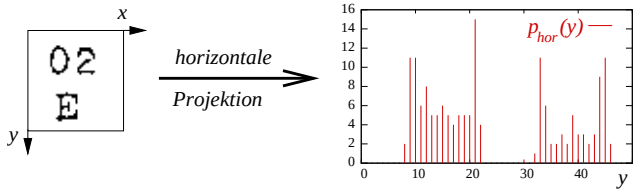
Projektionen

- Zählen schwarzer Pixel je Zeile oder Spalte
  - *horizontale* oder *vertikale* Projektionen möglich:

horizontale Projektion: $p_{hor}(y) = \sum_{x \in \mathbb{Z}} f(x,y)$

vertikale Projektion: $p_{ver}(x) = \sum_{y \in \mathbb{Z}} f(x,y)$

  - Ergebnis ist eindimensionale Liste der Projektionswerte



2.4 Onebit-Operationen (7)

- Projektionsmethoden in Gamera:
  - `Image.projection_rows()`
    berechnet Summen je Zeile und gibt sie als Int-Liste zurück
  - `Image.projection_cols()`
    berechnet Summen je Spalte und gibt sie als Int-Liste zurück
  - beides zusammen mit *projections()*, z.B.

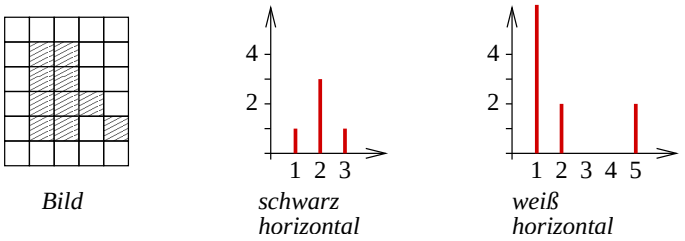
```
(prows, pcols) = img.projections()
```

    im GUI wird Ergebnis in Fenster grafisch dargestellt:

2.4 Onebit-Operationen (8)

Runlengths

- das *Runlegth-Histogramm*
  - zählt wie oft jede Lauflänge vorkommt
  - es gibt vier Histogramme, je nachdem ob schwarze/weiße oder horizontale/vertikale Lauflängen betrachtet werden



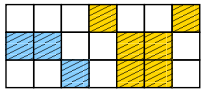
2.4 Onebit-Operationen (9)

- Image-Methoden zum Runlength-Histogramm:
  - `run_histogram(color, direction)`
    gibt Häufigkeit je Lauflängen als Liste zurück
    mögliche Werte für *color*: "black" oder "white"
    mögliche Werte für *direction*: "vertical" oder "horizontal"
  - `most_frequent_run(color, direction)`
    häufigste Lauflänge; auch die *n* häufigsten Lauflängen möglich mit *most_frequent_runs(n, color, direction)*
- Methoden zum Entfernen von Lauflängen:
  - `filter_xxx_runs(length, color)`
    wobei *xxx* angibt, welche Lauflängen entfernt werden
    - *xxx* = *narrow*: alle horizontalen kleiner *length*
    - *xxx* = *short*: alle vertikalen kleiner *length*
    - *xxx* = *wide*: alle horizontalen größer *length*
    - *xxx* = *tall*: alle vertikalen größer *length*

2.4 Onebit-Operationen (10)

Connected Components (CCs)

- CCs sind zusammenhängende schwarze Bereiche
  - Gamera verwendet dabei 8-Connectivity
  - Beispiel für ein Bild mit zwei CCs:

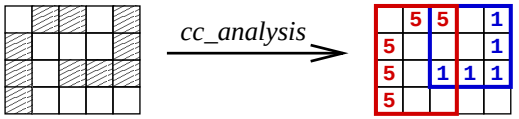


- Zerlegung in CCs mit *cc_analysis()*
  - gibt eine Liste von Images zurück; jedes Image ist vom Datentyp *Cc*, der von Onebit-ImageView abgeleitet ist

```
# erzeuge Demobild "rgb" mit allen CCs höher als 3 Pixel rot
rgb = a.to_rgb()
ccs = a.cc_analysis()
for c in ccs:
    if c.nrows > 3:
        rgb.highlight(c, RGBPixel(255,0,0))
```


2.4 Onebit-Operationen (11)

- Repräsentation von CCs in Gamera
  - bei *cc_analysis()* werden Bildwerte verändert: alle Punkte derselben CC erhalten denselben Wert ("Label")



  - bei Methoden, die auf Onebit-Bildern arbeiten, werden alle Werte ungleich 0 wie "schwarz" behandelt
  - Datentyp *Cc* hat zusätzliche Eigenschaft *label*
    ⇒ gibt an welche Punkte innerhalb Bounding-Box zu CC gehören
    (siehe Beispiel oben für überlappende Bounding-Boxen)