4.2 Page-Segmentation

Fragestellung:

- Zerlegung des Bildes in *logische* Bereiche, wie z.B.
  - Spalten, Absätze, Textzeilen in Textdokumenten
  - Notensysteme in Musiknoten
- enger Bezug zur *Layout Analysis*
  - zusätzlich zur Segmentierung noch Bedeutung der Segmente ermitteln
  - Grenzen sind oft fließend ⇒ meist nicht unterschieden

Ansätze:

- *Top-Down* Vorgehen
  - Zerlegung von größeren in immer kleinere Komponenten, z.B. Spalten → Absätze → Zeilen
- *Bottom-Up* Vorgehen
  - sukzessive Gruppierung kleiner Komponenten in größere Einheiten
  - Ausgangspunkt meistens Connected Components

4.2.1 Top-Down Methoden (1)

Betrachte zwei einfache Verfahren:

- *Iterative Projection Profile Cutting*
- *Runlength Smearing*

Beides Verfahren für *Manhatten-Layout*:

- weißer Zwischenraum senkrechte und waagerechte Linien wie im Straßenplan von Manhatten
- folglich vorher Rotationskorrektur nötig

4.2.1 Top-Down Methoden (2)

Projection Profile Cutting

Beobachtung im Projektionsprofil:

- *Spaltengrenzen* entsprechen Nullwerten (oder Minima) im *vertikalen* Projektionsprofil
- *Zeilengrenzen* entsprechen Nullwerten (oder Minima) im *horizontalen* Projektionsprofil

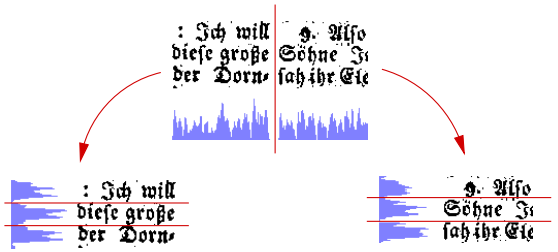
4.2.1 Top-Down Methoden (3)

Problem:

- versetzte Spalten verwischen horizontales Projektionsprofil

Lösung:

- spalte Teilbereiche abwechselnd horizontal und vertikal



4.2.1 Top-Down Methoden (4)

Algorithmus *Projection Profile Cutting*

- Kriterium zur Zerlegung:
  - splitte Projektionsprofil an "Lücken"
  - "Lücke" = alle Werte in Bereich der Breite *Tx* bzw. *Ty* unterhalb einer Schwelle *noise* (ideales Bild: *noise* = 0)
- Iteratives Vorgehen:
  - zerlege abwechselnd horizontal und vertikal
  - immer weiter iterieren
  - Abbruch, wenn beide Zerlegungen keine Splitpunkte mehr finden

4.2.1 Top-Down Methoden (5)

Plugin *projection_cutting* in Gamera

- Inputparameter
  - wenn *Tx* = *Ty* = 0 (default), werden sie so geraten, dass das Ergebnis Textzeilen sein sollten
  - "raten" erfolgt auf Basis des Medians der CC-Höhe
- Rückgabewert
  - wie bei *cc_analysis*
    - Rückgabewert ist Imagelist der Segmente
    - das Bild wird verändert durch Segmentlabels
    - Segmente entsprechen der letzten Zerlegung (typischerweise Textzeilen)

Bemerkung:

- die iterative Zerlegung liefert eigentlich mehr Informationen (Zuordnung Untersegmente zu Obersegment)
- diese werden vom Plugin nicht zurückgegeben

4.2.1 Top-Down Methoden (6)

Runlength Smearing

Idee:

- fülle weißen Zwischenraum innerhalb der Zeilen mittels Füllen kurzer weißer Lauflängen (Runlengths)
- Ergebnis: Bounding-Boxes der Zeilen als schwarze Blöcke

Beispiel:

- die Bildwerte einer Zeile seien

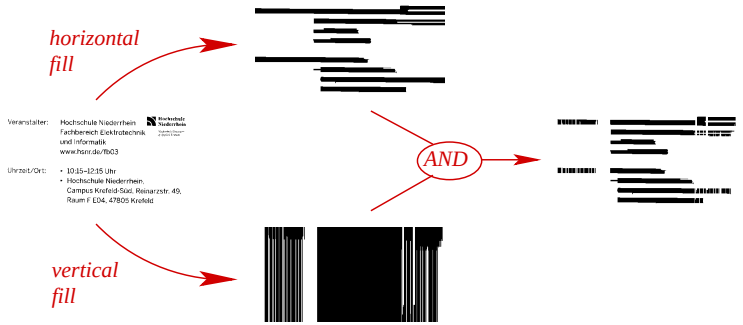
  `00001101001111010000011001110010`

- alle weißen Lauflängen < 4 gefüllt:

  `00001111111111110000011111111111`

4.2.1 Top-Down Methoden (7)

Füllen der Lauflängen vertikal und horizontal



4.2.1 Top-Down Methoden (8)

Algorithmus *Runlength Smearing*

- erzeuge neues Bild *rlshor*, in dem horizontale weiße Lauflängen kleiner *Cx* gefüllt sind
- erzeuge neues Bild *rlsver*, in dem vertikale weiße Lauflängen kleiner *Cy* gefüllt sind
- verknüpfe *rlshor* und *rlsver* mit logischem AND
- fülle im Ergebnisbild horizontale weiße Lauflängen kleiner *Csm* (Grund: siehe Abb. vorherige Seite)
- mache auf Ergebnisbild CC-Analyse

. Bounding-Boxen der CCs entsprechen Zeilen

9

4.2.1 Top-Down Methoden (9)

In Gamera leicht von Hand umsetzbar:

```
# filter_*_runs verändert Bild statt es
# zurückzugeben => Bild vorher kopieren!
rlshor = img.image_copy()
rlshor.filter_narrow_runs(Cx, "white")
rlsver = img.image_copy()
rlsver.filter_short_runs(Cy, "white")
ergebnis = rlsver.and_image(rlshor)
ergebnis.filter_narrow_runs(Csm, "white")
zeilen = ergebnis.cc_analysis()
```

□ alternativ fertiges Plugin *runlength_smearing*

- Rückgabewert wie bei *projection_cutting*

10