JBIG2 Supported by OCR

Radim Hatlapatka

Masaryk University, Faculty of Informatics, Brno, Czech Republic
<208155@mail.muni.cz>

Bremen, 9th July 2012

EuDML
The EUROPEAN DIGITAL MATHEMATICS LIBRARY

Motivation

- DLs (even DMLs) contain a vast amount of PDFs with a scanned text
- Not only large storage space is required, but also high bandwidth is needed in order to provide the documents swiftly to the end-users
- Possible improvement using a good compression methods
- JBIG2 provides great compression ratio for this kind of documents
- JBIG2 principle partially corresponds to process of OCR text recognition

Example

$$a(\mathbf{e}, \mathbf{v}) = \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) N_j^0(\mathbf{v}^0) \left|\det \mathscr{F}\right| \mathrm{d}X =$$

$$= \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) b_{jm}^{(1)} \left|\det \mathscr{F}\right| \frac{\partial v_m^0}{\partial X_1} \mathrm{d}X +$$

$$+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) b_{jm}^{(2)} \left|\det \mathscr{F}\right| \frac{\partial v_m^0}{\partial X_2} \mathrm{d}X +$$

$$+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) n_{jm} v_m^0 \left|\det \mathscr{F}\right| \mathrm{d}X = a_1 + a_2 + a_3 .$$

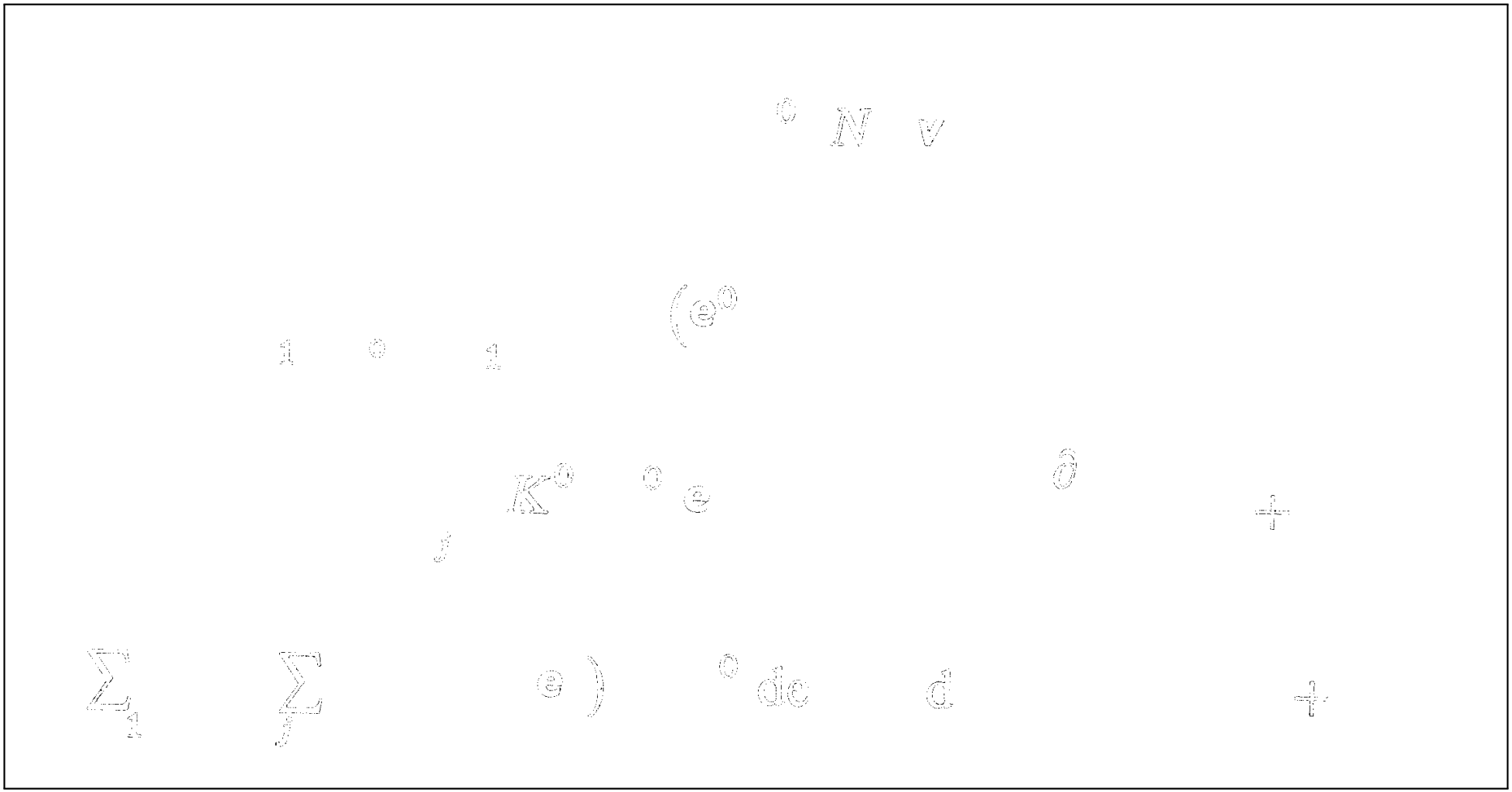
$$a(\mathbf{e}, \mathbf{v}) = \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) N_j^0(\mathbf{v}^0) \left|\det \mathscr{F}\right| \mathrm{d}X =$$

$$= \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) b_{jm}^{(1)} \left|\det \mathscr{F}\right| \frac{\partial v_m^0}{\partial X_1} \mathrm{d}X +$$

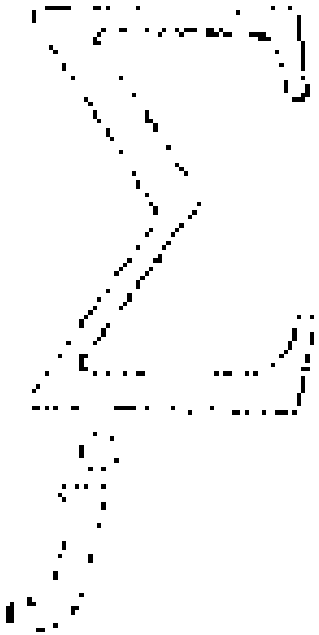
$$+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) b_{jm}^{(2)} \left|\det \mathscr{F}\right| \frac{\partial v_m^0}{\partial X_2} \mathrm{d}X +$$

$$+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) n_{jm} v_m^0 \left|\det \mathscr{F}\right| \mathrm{d}X = a_1 + a_2 + a_3 .$$

Example Showing Part of Redundant Data in Image



Example Showing Part of Redundant Data in Image



JBIG2 And Its Specific Characteristics

- Standard for compressing bitonal images
- Created mainly for compressing text in images
- Supports both lossless and lossy mode
- Supports multi-page compression
- Supported in PDF since version 1.4
- Image is segmented to different regions based on data type and specialized compression is used for each region type
- Text region is segmented to connected components where representants are identified and occurrences just points to them

JBIG2 vs OCR

- Both segment image to components (text blocks, words, symbols)
- OCR requires knowledge of font to achieve good recognition accuracy (uses existing collection of symbols)
- JBIG2 creates new font as image is being processed (creates new collection of symbols)
- OCR needs to choose letter representant for each symbol even though, it is uncertain
- JBIG2 can create a new symbol, if it is not certain about having already such symbol

JBIG2 vs OCR

- Both segment image to components (text blocks, words, symbols)
- OCR requires knowledge of font to achieve good recognition accuracy (uses existing collection of symbols)
- JBIG2 creates new font as image is being processed (creates new collection of symbols)
- OCR needs to choose letter representant for each symbol even though, it is uncertain
- JBIG2 can create a new symbol, if it is not certain about having already such symbol

JBIG2 vs OCR

- Both segment image to components (text blocks, words, symbols)
- OCR requires knowledge of font to achieve good recognition accuracy (uses existing collection of symbols)
- JBIG2 creates new font as image is being processed (creates new collection of symbols)
- OCR needs to choose letter representant for each symbol even though, it is uncertain
- JBIG2 can create a new symbol, if it is not certain about having already such symbol

Jbig2enc

- An open-source JBIG2 encoder written in C/C++ by Adam Langley
- Uses an open-source Leptonica library for manipulating with images and image segmentation
- Supports both lossless and lossy mode
- Allows creating output suitable for inserting into a PDF document

PdfJbIm

- Open-source tool written in Java for (re)compression of bitonal images inside PDF
- Uses benefits of standard JBIG2 which is supported in PDF since version 1.4 (Acrobat 5)
- Uses improved jbig2enc with symbol coding used for text area
- Supports multi-page compression

Tesseract OCR

- An open-source OCR engine written in C/C++ being developed by Google
- One of the best open-source OCR in character recognition accuracy
- Uses Leptonica library for manipulating with images and holding image structures
- Supports more than forty languages

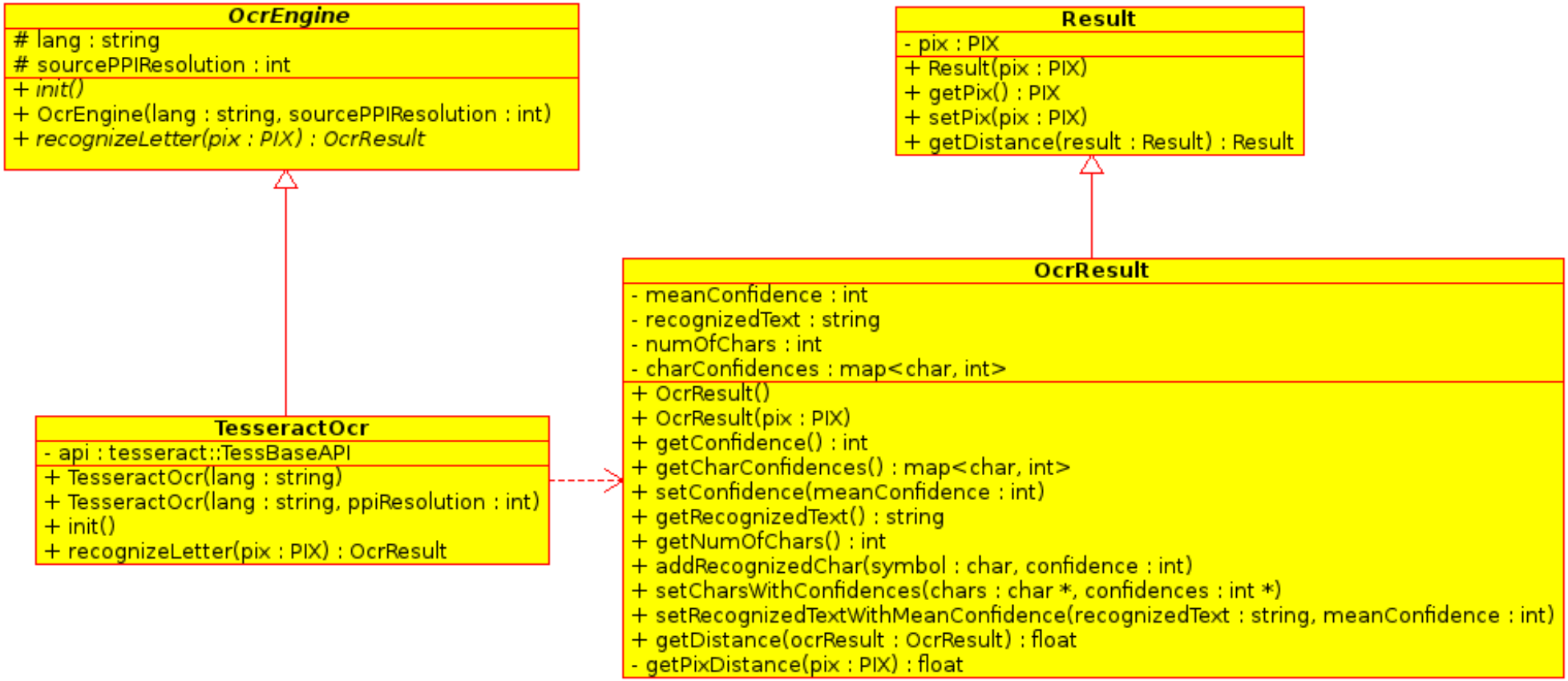
Improvement of Jbig2enc – Motivation

- Number of different symbols recognized for a page is several times greater than of born digital documents
- Our improvement without using OCR created in bachelor thesis reduces the number of recognized different symbols, but with OCR it can be improved even further

Improvement of Jbig2enc without OCR Usage

- Comparison of representative symbols
  - Two symbols are considered equivalent if there is not found a big enough difference to form a line or a point
- Unification of two equivalent symbols to one

Jbig2enc: API for Using OCR



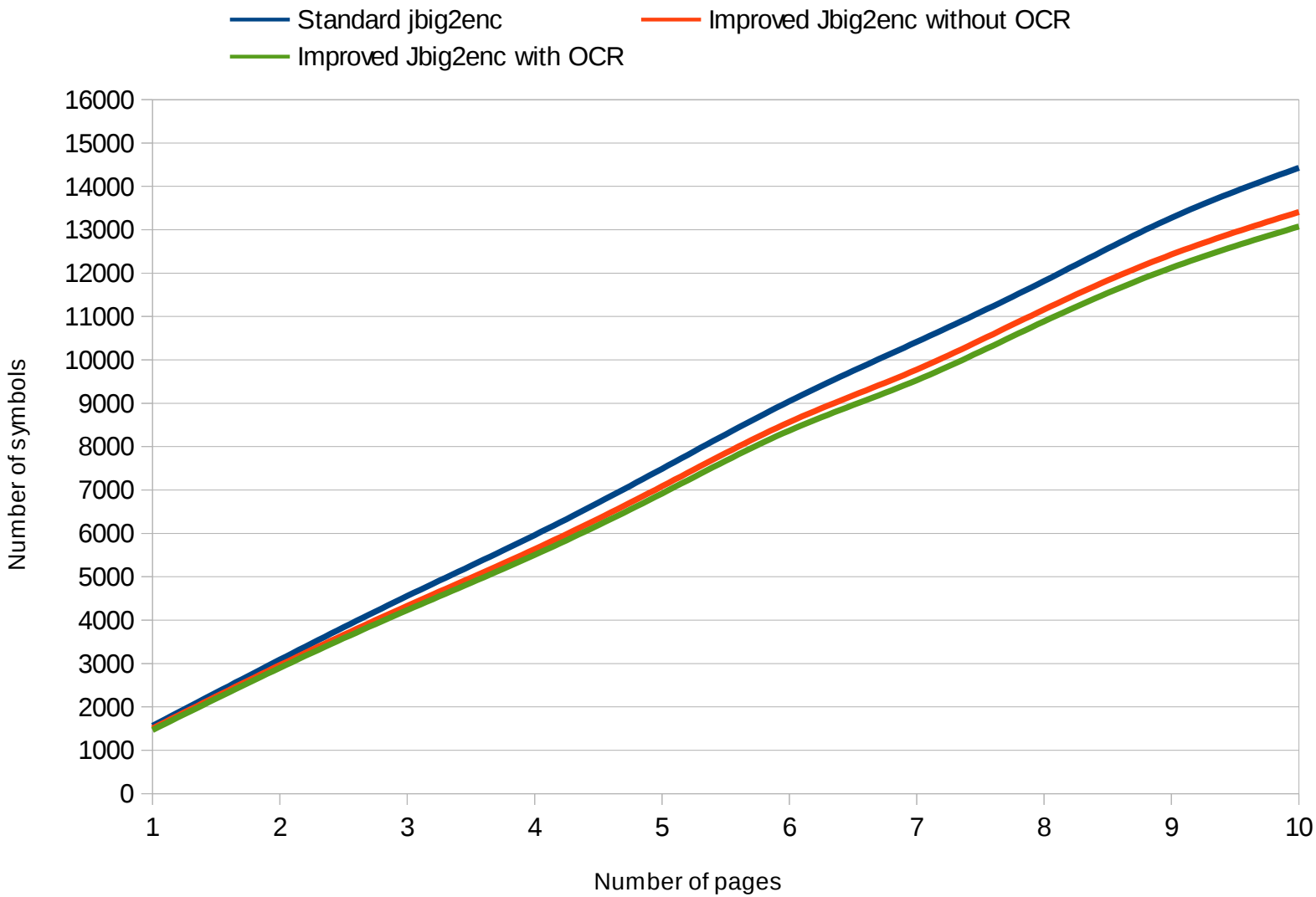
Jbig2enc: Comparison of representants

- Comparison is based on similarity distance function
- All symbols which are closer than preset value are considered equivalent
- For counting distance are used confidences, size of symbols and amount of different pixels

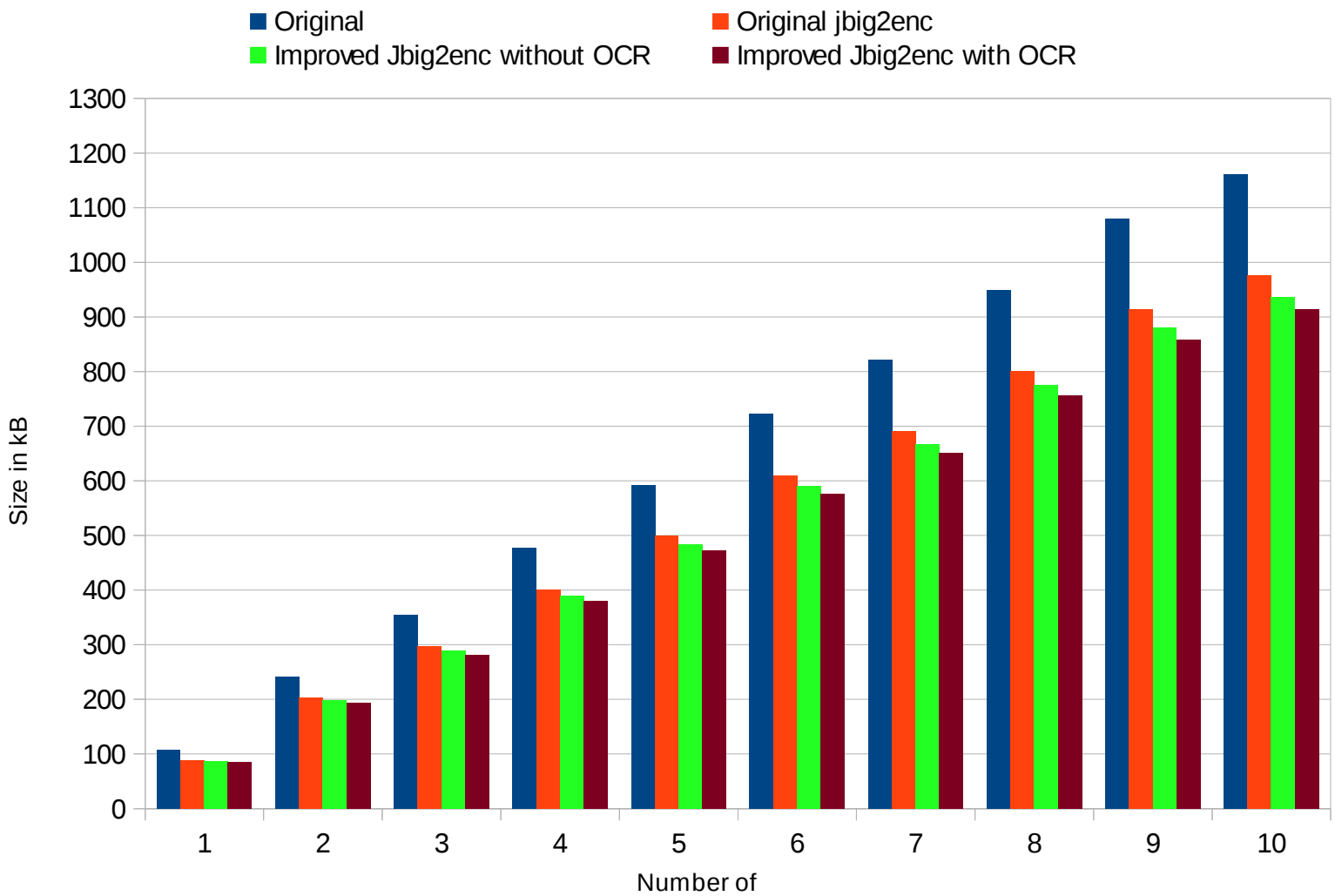
Evaluation: Testing Data Description

- Evaluated mainly on data from Czech Digital Mathematical Library (DML-CZ)
- Testsuite of more than 800 PDFs with more than 4000 pages
- PDF documents compressed using pdfJbIm tool and appropriate version of jbig2enc encoder
- For compression used default jbig2enc encoder thresholding level for minimizing loss (`-t 0.9`)

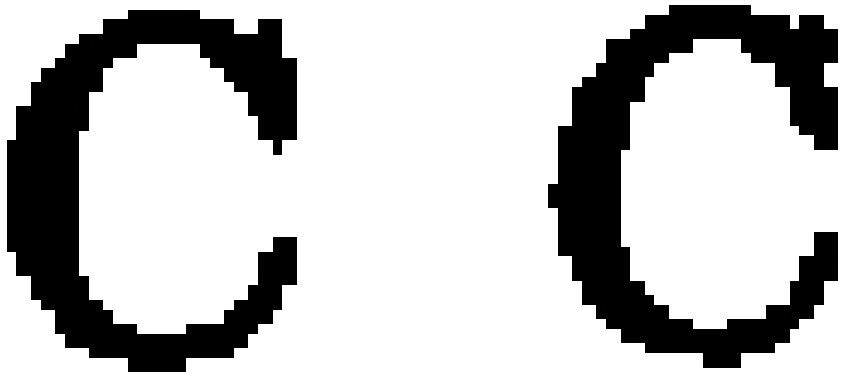
Evaluation: Amount of Different Symbols Recognized



Evaluation: PDF Size Before and After Compression



Evaluation: Example of Equivalent Symbols



Evaluation: Problematic Symbols

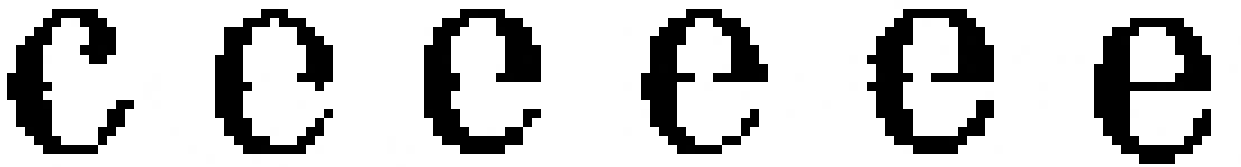


Image Before and After Compression With OCR and Without OCR

$$a(\mathbf{e}, \boldsymbol{v}) = \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) N_j^0(\boldsymbol{v}^0) \left|\det \mathscr{F}\right| \mathrm{d}X =$$

$$= \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) b_{jm}^{(1)} \left|\det \mathscr{F}\right| \frac{\partial v_m^0}{\partial X_1} \mathrm{d}X +$$

$$+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) b_{jm}^{(2)} \left|\det \mathscr{F}\right| \frac{\partial v_m^0}{\partial X_2} \mathrm{d}X +$$

$$+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) n_{jm} v_m^0 \left|\det \mathscr{F}\right| \mathrm{d}X = a_1 + a_2 + a_3 .$$

$$a(\mathbf{e}, \boldsymbol{v}) = \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) N_j^0(\boldsymbol{v}^0) \left|\det \mathscr{F}\right| \mathrm{d}X =$$

$$= \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) b_{jm}^{(1)} \left|\det \mathscr{F}\right| \frac{\partial v_m^0}{\partial X_1} \mathrm{d}X +$$

$$+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) b_{jm}^{(2)} \left|\det \mathscr{F}\right| \frac{\partial v_m^0}{\partial X_2} \mathrm{d}X +$$

$$+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) n_{jm} v_m^0 \left|\det \mathscr{F}\right| \mathrm{d}X = a_1 + a_2 + a_3 .$$

$$a(\mathbf{e}, \boldsymbol{v}) = \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) N_j^0(\boldsymbol{v}^0) \left|\det \mathscr{F}\right| \mathrm{d}X =$$

$$= \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) b_{jm}^{(1)} \left|\det \mathscr{F}\right| \frac{\partial v_m^0}{\partial X_1} \mathrm{d}X +$$

$$+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) b_{jm}^{(2)} \left|\det \mathscr{F}\right| \frac{\partial v_m^0}{\partial X_2} \mathrm{d}X +$$

$$+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K_{ij}^0 N_i^0(\mathbf{e}^0) n_{jm} v_m^0 \left|\det \mathscr{F}\right| \mathrm{d}X = a_1 + a_2 + a_3 .$$

Image Before and After Compression Without OCR Usage: Differences

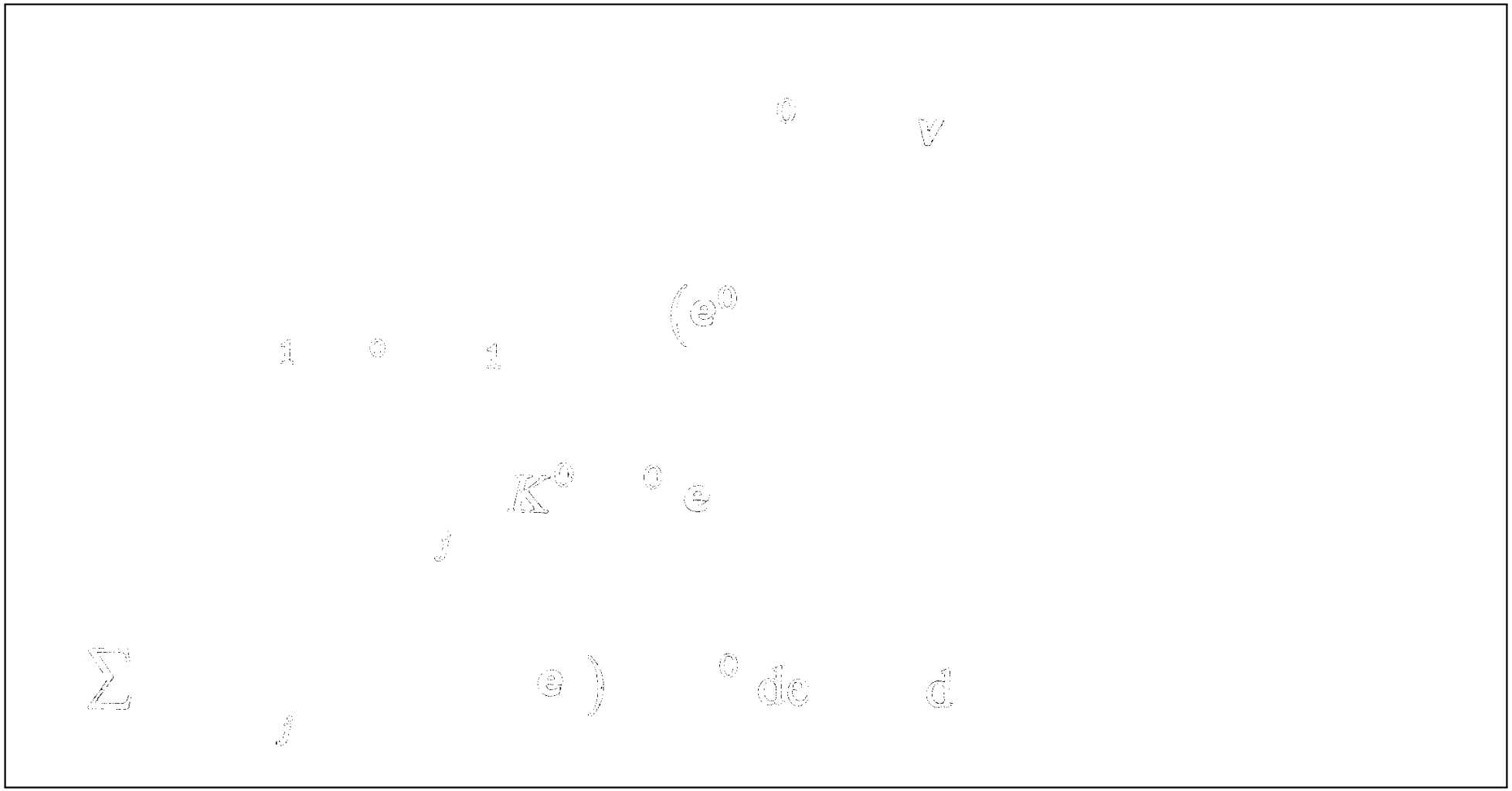


Image Before and After Compression Without OCR Usage: Differences

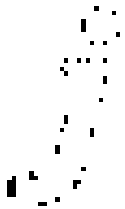


Image Before and After Compression with OCR Usage: Differences

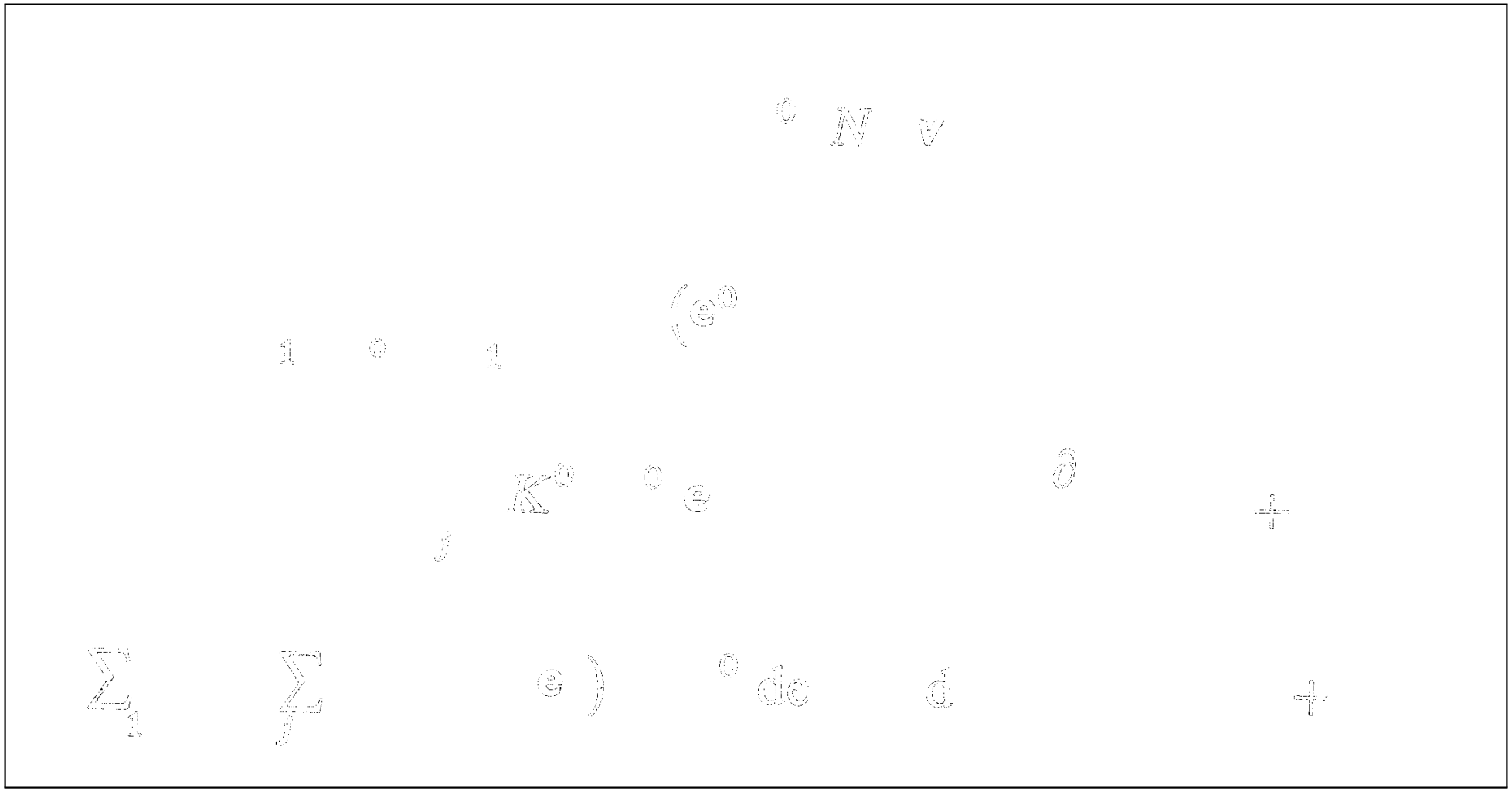


Image Before and After Compression with OCR Usage: Differences

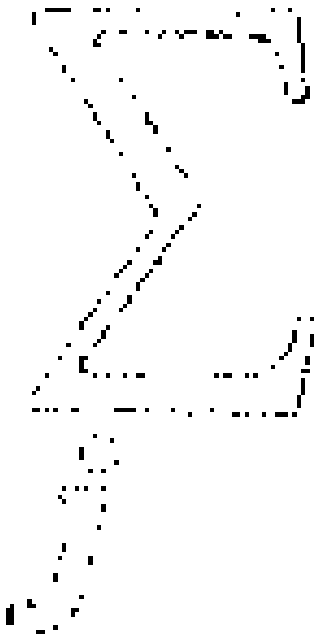


Image Before and After Compression with OCR Usage: Differences

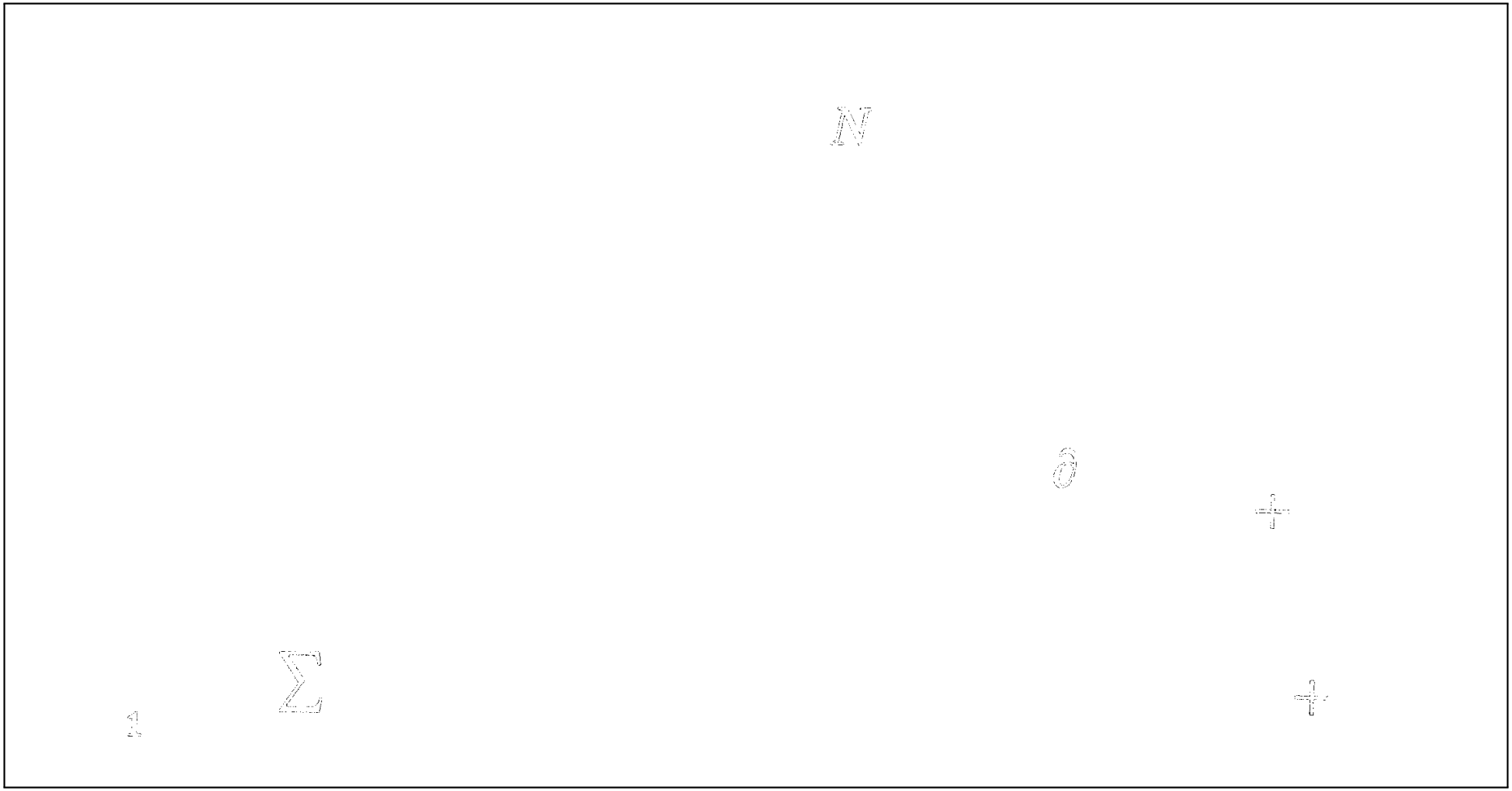


Image Before and After Compression with OCR Usage: Differences

Summary

- Using OCR engine we are achieving further size reduction
- Choice of the new representant for equivalent symbols is based on OCR recognition result (confidence) $\Rightarrow$ improves image quality
- Integrated into two digital mathematical libraries: DML-CZ and EuDML (or rather prepared to be used after more testing)

Future work

- Wide testing of created similarity function and jbig2enc improvement (as much as possible automatized)
- Creating universal language dictionary specialized on individual symbols including Math and train Tesseract for it
- Create modules for additional OCR engines such as InftyReader
- Make other parts of jbig2enc encoder for running in parallel
- Test integration of jbig2enc encoder in EuDML and DML-CZ
- Create better image quality detection and image quality improvement methods

End of the talk

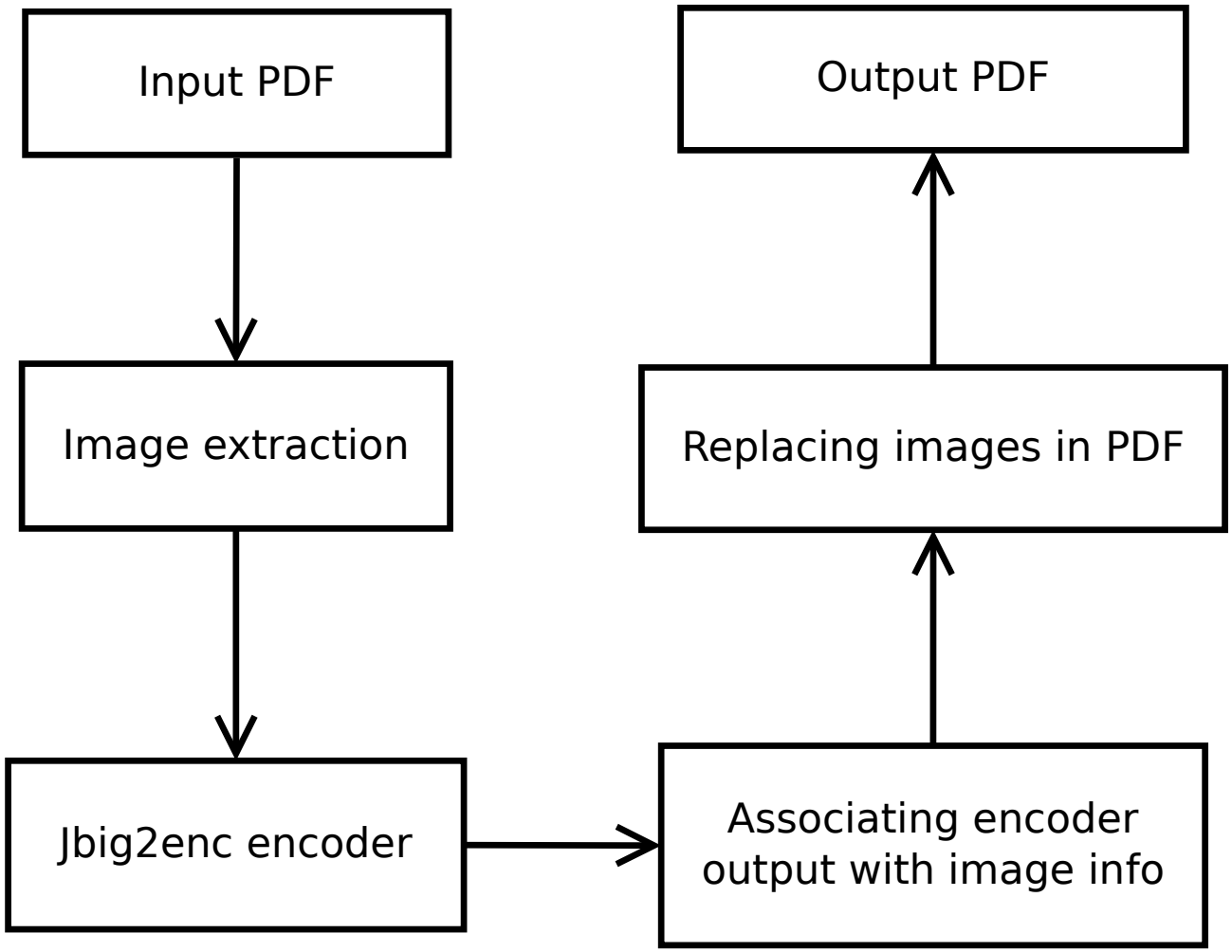
Questions? Comments?

PDF and JBIG2

```
5 0 obj
<< /Type /XObject /Subtype /Image
/Width 52 /Height 66
/ColorSpace /DeviceGray /BitsPerComponent 1
/Length 224
/Filter [ /ASCIIHexDecode /JBIG2Decode ]
/DecodeParms [ null << /JBIG2Globals 6 0 R >> ]
>>
stream .... endstream
endobj

6 0 obj
<< /Length 126 /Filter /ASCIIHexDecode >>
stream .... endstream
endobj
```

PdfJbIm – Workflow



Jbig2enc: Hash Function and Speed Improvement

- Two different hash functions
  - Without OCR
    - Uses size of symbol and number of holes
  - With OCR
    - Two layered hash – OCR result (recognized text) and hash counted from symbol size
- OCR recognition done in hash function
- Each symbol (representant) is recognized only once
- OCR engine initialized only once (expensive operation)
- OCR text recognition run in parallel

Evaluation: Speed Improvement