A Framework for the Automatic Analysis and Interactive Exploration of Document Aesthetics

Technical Report, April 21, 2016

Guido Reina
University Stuttgart

Sebastian Grottel
TU Dresden

Carsten Dachsbacher
Karlsruhe Institute of Technology

Abstract— Modern word processing software and typesetting systems such as TeX enable the quick creation of documents of various kinds. Although the quality of the software packages varies, all can produce aesthetically pleasing documents in terms of layout and type setting. Problems typically originate from the large number of parameters which are exposed to the user. These range from simple settings like typeface, font size and column width to more elaborate ones, such as kerning and leading. Most often default values are modified without grasping the consequences for readability and aesthetic appeal of the resulting document.In this paper, we present a system for interactive visualization and exploration of quantifiable aspects of document aesthetics such as alignment, spacing, gray values, but also of image color harmony. This system also allows for comparative analysis of multiple documents and document versions side-by-side. The documents are rated using an extensible and parameterizable plug-in system allowing the user to define a task-specific processing pipeline interactively. The rating is hierarchically organized such that the user can drill down into the different aspects that influence the final score. Our system takes standard document formats such as Adobe PDF or Microsoft XPS as input. Our system serves as a platform for further research on document aesthetics as well as a utility to sensibilize authors for these often underestimated aspects of scientific publishing.

Index Terms—Document Aesthetics, Aesthetics in Visualization, Text and Document Data.

1 Introduction

The creation of visually pleasing and pleasingly readable documents is the goal of designers of magazines, newspapers, scientific articles, and books. The broad availability of word processing software and typesetting systems allows almost everybody to be a designer of a document of any kind. The guidelines for an aesthetically pleasing layout and typographic design can be found in any respective textbook, such as [10]. However, technical writers and researchers cannot be expected to be well-versed in all aesthetic implications of document layout. This is the reason why professional designers are involved in publications in book form, where a high value is set on an aesthetic appearance. In the publication process of typical conference proceedings the final layout quality is mostly dictated by the given document templates and aesthetics may still suffer from shortfalls of the software, space limitations (ignoring appropriate spacing and layouts), or simply from not paying enough attention.

In this paper we present a system providing an extensible framework for the automatic, quantitative evaluation of document aesthetics. Our system takes standard electronic document formats as input, performs the analysis, and presents a summary of the analysis and ratings to the user. The user can also explore multiple documents or a document history side-by-side, and examine and compare all analysis results interactively; e.g. to evaluate the evolution of a document over time or compare different documents. The system offers a plug-in interface allowing for integration of further aesthetics metrics. Such extensions can be easily prototyped and tested in our environment since all intermediate analysis data can be retained and accessed from all modules.

The remainder of this paper is structured as follows. In the next section we discuss related work, and describe the aesthetical aspects considered in our analysis and the rating scheme in Section 3. The implementation details are presented in Section 4. Thereafter we present the results in Section 5, and point to future research in Section 6.

2 Related Work

The pursuit of creating aesthetically pleasing documents – preferably even generated automatically – is not new [14], but becomes more and more important, e.g. in the context of on-demand printing of electronic publications. Depending on the targeted media and the circle of authors there might be templates defining many aspects of the layout, e.g. TEX templates for conference submissions. Sometimes publishers simply take text files of any format and designers take over the typography and layout. There have been many efforts to alleviate this process by creating document layouts automatically. Iwai et al. [13] developed a document layout system based on the extraction of logical and reference structures of documents. It has been implemented in a Japanese word processor and automatically formats text and creates layout of text, figures, and tables. However, we are not aware of a commercial word processor exposing this feature to the user. Oliveira [5] presents two algorithms for placing rectangular items on pages for sales brochures and similar documents, and for positioning free-form elements in documents with a multi-column layout. Purvis et al. [19] and Goldenberg [9] use genetic algorithms to generate page layouts, e.g. by minimizing white space and maximizing alignment of element edges.

Creating a "good" layout requires the measuring of the aesthetic quality which is a non-trivial task as aesthetic appeal is highly subjective. However, from experience it is known which aspects entail bad layout, and based on these observations this topic has been investigated [20, 12]. An automatic measurement of the aesthetics of document layout has been described by Harrington et al. [11]. They combine heuristic measures of attributes degrading the aesthetic quality, among others misalignment or irregular spacing of paragraphs, to obtain a final score for a document. Faria and Oliviera [7] measure the quality of an adaptive document (an instance of a template) with respect to the template itself by computing a score penalizing displacement of elements and modification of attributes, such as font sizes. If the score drops below a certain threshold the document is sent to further (possibly human) review.

Related to our system is the analysis of documents such as scanned books or newspapers which exhibits many challenges: Pages have to be decomposed and text blocks and images have to be identified and separated [3, 6, 16]. In contrast, we take electronic documents as input and thus can access the internal data structures largely leaping the tedious task of typography extraction from scanned images [15] (see Section 4 for the subtleties arising from electronic documents).

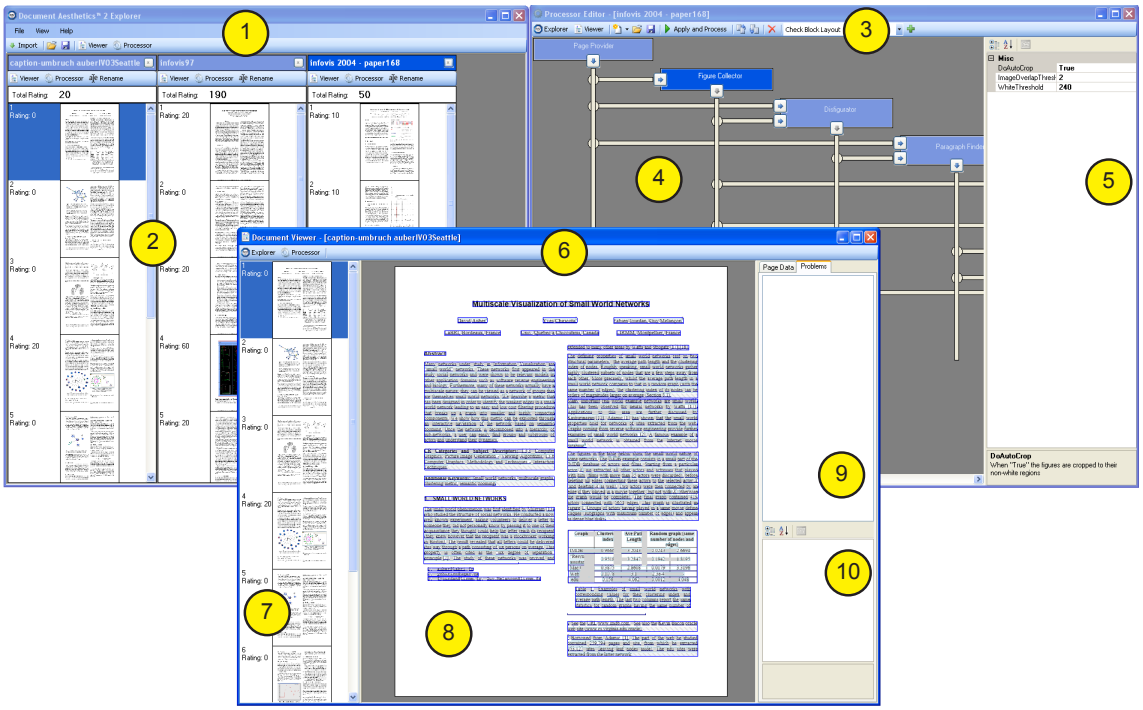

Fig. 1. The user interface of our system. In general, there are 3 types of windows: the Document Explorer (1), the Processor Editor (3), and the Document Viewer (6). The Document Explorer contains multiple previews (2) of opened documents, allowing for rough comparison of documents and the respective scores. The main part of the Processor Editor is the editing area (4) depicting the active plugins, the data interconnects and the produced data. On the right (5), parameters of the selected plugin can be adjusted. Since multiple Document Views can be opened, each one contains its own preview (7). The selected page is shown in detail with overlaid information (8). On the right, the problem tree summarized the rating of the document hierarchically (9). The weighting of the class of the selected aesthetic problem can be manipulated interactively (10).

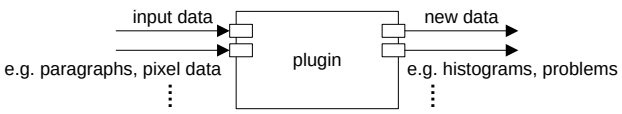

Fig. 2. General structure of a plugin of our system: an arbitrary number of typed input and output 'slots' can be defined. Produced data can be utilized by other plugins, drawn into the document view, or evaluated and rated as an aesthetics problem.

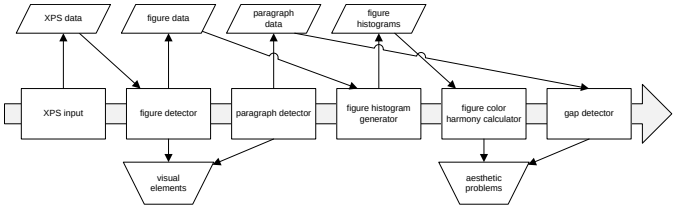

Fig. 3. Exemplary pipeline configuration for the detection of color harmony and text gap problems. The order in which the plugins are processed is implicitly determined by the required input data.

3 Document Aesthetics

In this section we will describe the aspects considered in our document analysis. Prior to the analysis itself we need to access the document data, i.e. glyphs, images, and figures, as input to the analysis algorithms. The input data format is the first design choice when implementing a system for document analysis. The options range from directly integrating the analysis in a word processor, to just taking rasterized pages as input. We found that using a page description language, such as Postscript, Adobe PDF, or Microsoft XPS is a very good compromise as it provides fine grained access to the document data, as well as flexible document import. All of them describe the page contents and locations of glyphs, images, and vector graphics. We decided to use the Microsoft XPS format for two reasons: First, we can easily create XPS documents from any word processing software, and convert from Postscript or PDF to XPS by using the officially provided virtual printer device (under Windows). The main advantage, however, is that Microsoft provides an easy-to-use library for reading and writing XPS files, and for rasterizing them into bitmaps.

Our system consists of a user-definable chain of plugins that gradually process the data extracted from an XPS file. Each plugin defines typed input and output slots for required and produced data (see Fig. 2). The output can be used by multiple subsequently employed modules, and as such reducing the processing overhead for frequently required data. Output data can be overlaid on the document view, indicating the extracted information, e.g. paragraph bounds, spacing etc. Additionally, plugins can also produce output on aesthetic problems with a normalized score. These problems are grouped by the categories which can later be weighted individually depending on their overall visual impact. These weights can be adjusted interactively in the document viewer without re-processing the data. Figure 3 shows an exemplary pipeline along with the produced data for color harmony and text gap rating.

3.1 Typography

The analysis of the textual parts of the document is based on data obtained from *glyph runs* which are sequences of glyphs and their position, which we directly obtain from the XPS document. From this information we can easily compute the base lines and deduce informa-

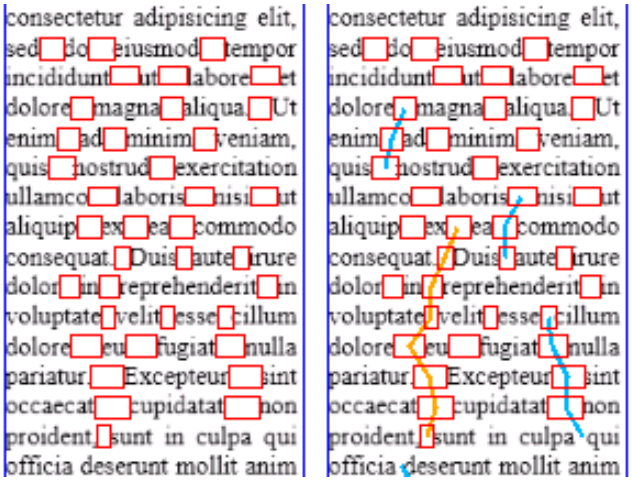

Fig. 4. Unsatisfactory justification may cause text gaps that are unpleasant for the reader. This example exhibits many such flaws due to a narrow columns width. Rivers caused by unfortunate alignment of white space are highlighted.

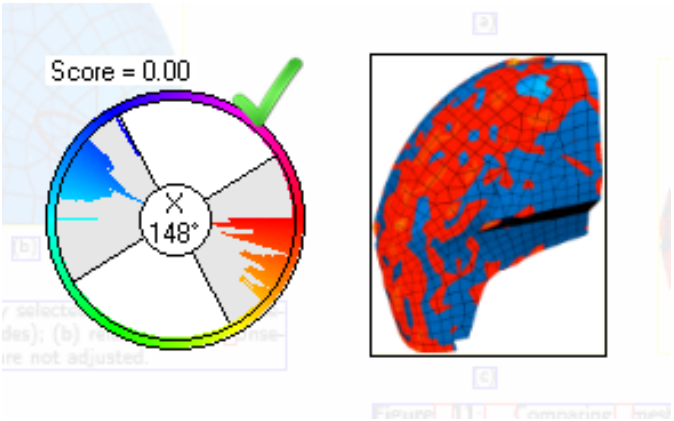

Fig. 5. For selected figures we show their saturation-weighted hue histogram and the best-fitting harmonic template along with a score and the derived rating icon: ok (checkmark), warning (warning sign), bad (red cross) – see also figure 8 or the summary tree in figure 12.

tion about the paragraph layout, i.e. analyze the glyph extents to estimate the text alignment. However, sometimes the XPS writer partially decomposes text into individual spline paths. In this case our system uses heuristics to compensate for this missing information on glyphs, their kerning, and baselines. After re-assembling text lines from glyph runs and paths, we detect (large) gaps between words which might arise from justification with small column-width and bad automatic hyphenation (Fig. 4).

We also rasterize the document to extract further information about the text blocks (implementation details are outlined in Section 4):

- We determine row-wise and column-wise histograms for distinguishing textual parts belonging to different paragraphs (Fig. 7) and for determining paragraph gray values. The gray value is a typical criterion for evaluating document aesthetics and it should not vary strongly across the document (see Fig. 9).

- We also detect the geometry of whole paragraphs including bounding boxes (shown blue in Fig. 1), first line indents, and the used length of the last line (which do both not count towards the gray value).

- We compare the width of individual lines to the paragraphs' bounding boxes to determine the alignment and to detect overhangs of text (Fig. 7) and figures (Fig. 11).

3.2 Color Harmony

For many technical documentations and articles it is desirable that the images and figures in the document exhibit an aesthetically pleasing coloring or choice of colors. Likewise, when multiple illustrations are used we would like their colors to go well together. Cohen-Or et al. [4] analyzed the hue histograms of images and harmonize colors by modifying them to fit into a set of experimentally determined templates. We adapt this idea and compute a color harmony score by comparing the actual hue histogram to the best-fitting template yielding a score for every image (Fig. 5) based on those hues that do not fit the template. A large score means inharmonic colors. To determine the color harmony across figures, the user can select multiple images and the application displays a pairwise color harmony, i.e. how harmonic the result would be if the images were merged prior to computing the histogram. Also, for every page of the document all combinations of figures are analyzed to derive the overall color harmony. Our implementation also creates a *harmony matrix* (presented in a separate window, see Fig. 8) which shows the pairwise harmony of all images in the document. The harmony matrix supports synchronized selection (linked views), i.e. when the user clicks on images in the main window, the respective entries of the matrix are highlighted, and clicking into the matrix opens the respective page and selects the corresponding figure.

3.3 Rating Scheme

Our application presents a quick overview of the estimated aesthetic quality to the user by providing a structured summary of all possible problematic document locations that have been detected. The rating is categorized into *warning* and *error*, which contribute accordingly to the overall rating. These ratings are linearly accumulated for a document to give a concise overall impression.

The (relative) thresholds we refer to can be customized by the user, however, the default values in combination with automatically detected text parameters generally work well. The individual aspects which contribute to the overall score, or generate warnings or errors, are the following:

- **Gray value deviation:** The gray values of all text blocks in the document are collected and the global median gray value is determined, as well as the global median deviation thereof. If any block has a higher deviation from the global median than the median deviation, it will generate a warning.

- **Text gaps:** If two glyphs or two glyph runs on a single line are separated by too large an empty space, a gap error is generated. We count the detected gaps for each document page and generate a warning or an error if the number exceeds the thresholds.

- **Boundary violations:** These are items jutting out of the determined paragraph bounds. Such violations usually happen due to strings which are not hyphenated (such as URLs), equations, and considerable tweaking of figure placement. Typically the user is responsible for such flaws and thus we create an error.

- **Color harmony:** The hue histogram of each figure is considered individually and might generate a warning or an error if it is inharmonic, i.e. it creates a high score. All pair-wise combinations of figures can generate an additional warning or error.

The result is displayed in a tree view (see Fig. 6) such that the user can expand the interesting parts and interactively navigate the document preview by selecting the different reported problems. By clicking an item the application automatically navigates to and highlights the respective document portion.

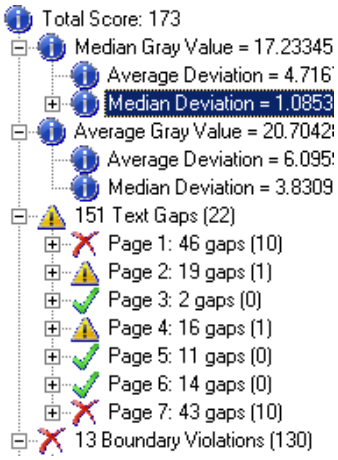

Fig. 6. An excerpt from an exemplary document rating. The problems are listed hierarchically, with the respective sub-scores in parentheses.

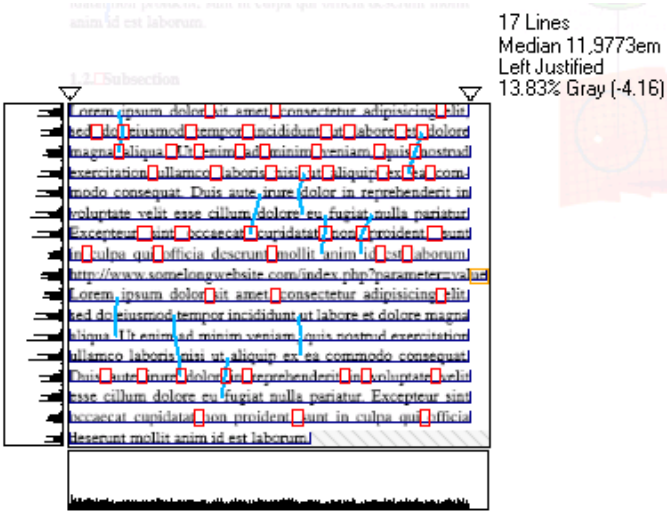

Fig. 7. For selected paragraphs we show extracted information, like the baseline of the detected lines, a local histogram and the estimated horizontal extents (shown as arrows on top) that can be used to derive the paragraph alignment. A border violation caused by an URL is highlighted in this example.

4 Implementation

If the input file is in PDF format it will be converted into XPS using Microsoft's virtual printer device. All document processing in our application uses the XPS format as it makes the document structure very comfortable to access via a dedicated API. The first step is to extract (and re-assemble if required) bitmap images in the document in their native resolution. We also crop figures as there might be empty or white borders overlapping the surrounding text.

Subsequent to the image extraction we gather information about the textual parts. For this, we generate a fixed-resolution preview of every page and compute horizontal and vertical histograms. Then we use the xy-cut method [17] to detect text blocks on the (rasterized) document pages. Please note that images are ignored during this detection. This implies that all blocks that we detect now are initially considered as textual parts. To detect other document structures, such as tables, we rely on heuristics (disussed below). Typically the xy-cuts create too many text blocks, e.g. headings where the numbering is distant from the label create two blocks, or justified text with large gaps is not detected as belonging together. We complement this step by rejoining those text blocks which are connected by a glyph run.

As previously mentioned, we use a heuristic to detect non-textual parts: e.g. we assume that a table has been found if we detect several thin horizontal and vertical paths nearby. Please note that no such information is stored explicitly in PDF or XPS documents. Tables are currently removed from further analysis and are not rated.

All glyph runs and paths inside the textual parts (from the previous xy-cuts step) are used to detect the individual text lines therein. Again, no such information is stored explicitly in the electronic document. A glyph run always belongs to one line only. Thus we detect the lines by considering all glyph runs one after another: A glyph run belongs to an already detected text line if it has a pronounced vertical overlap with it or its baseline is roughly equal to that of the line. If neither is the case, then the glyph is part of a new line. As mentioned before, characters might be represented by paths (splines or line segments) instead of glyphs. For paths there is no baseline information and we do not even have information how they form characters. In this case we determine an approximate baseline by averaging their respective lowest boundary. This simple approach proved to work well in our experiments (see Fig. 7), however, more advanced (and costly) algorithms can be used, such as the line-finder described by Breuel [1]. It is also imaginable to rely on clustering techniques taking the text block histogram into account.

Next, our application extracts the median font size for each text block directly if it contains at least one glyph run, otherwise we use the average line height detected in the previous step.

Using the information that has been gathered so far, we can decide whether a text block represents a single paragraph, or if it has to be further split. The text blocks are separated if the distance between two consecutive baselines is larger than a certain threshold (implies a regular leading), if an indent is detected on a line, or if a line is shorter than the previous one (for left flushed text). Alignment is determined by analyzing the boundary variance of text lines: High variance at both sides indicates centered text, high variance at only one side is interpreted as ragged text (i.e. flush to the opposite side), and no (very low) variance indicates justified paragraphs.

To compute the gray values in a later step, we generate a tight bounding box for every paragraph, excluding leading and trailing spaces. Both glyph runs and single paths in every line are analyzed to detect large gaps which point at problems with the layout. Using the spacing information (both small and large gaps), each paragraph is searched for rivers, i.e. whitespace vertically spanning several lines. Text columns are detected using the median bounds of the text blocks. These bounds are then used to detect figures, equations, and text violating the column widths.

For the analysis of the color harmony we create a saturation-weighted hue histogram for every image. Next we find the best-fitting harmonic template using Brent's algorithm [18]. The score for a figure is determined by the sum of the hue distance of every contained color to the best template, as described in [4] (Fig. 5) and normalized by the number of pixels in the image:

$$\frac{1}{\#pixels}\sum_{h} H(h)\cdot D(T,\alpha,h)\cdot S(h), \qquad (1)$$

where $H(h)$ is the histogram for hue h in degrees, $S(h)$ its total saturation, and $D(T,\alpha,h)$ the distance to the nearest edge in template T at the (optimal) orientation α. Perfect matches get a score of 0 and our application creates warnings or errors for each figure if the score exceeds the respective thresholds. In addition to the per-figure color harmony, the inter-figure color harmony is computed for all pair-wise combined figures in the whole document. This yields a figure matrix as shown in Fig. 8. The harmony between two images is defined by merging them (identical to combining the histograms and saturations) and then find matching templates.

The last analysis steps include the computation of the text block gray value and the rating of the single blocks based on their deviation from the median gray value across the whole document.

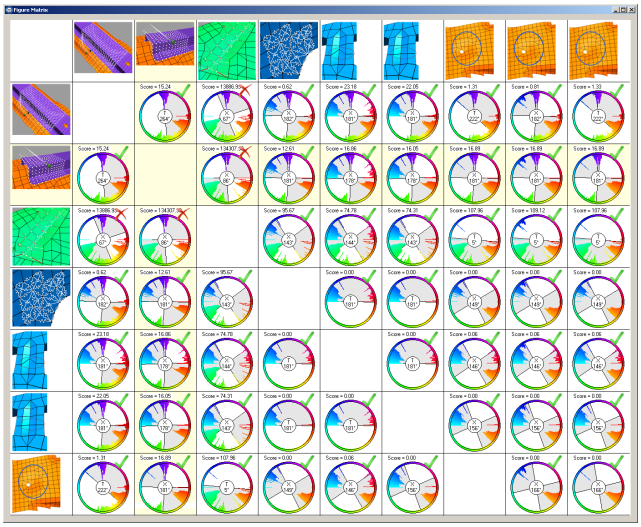

Fig. 8. A separate window shows the color harmony matrix for all possible combinations of figures, which are also used to derive the total color harmony score for the entire document.

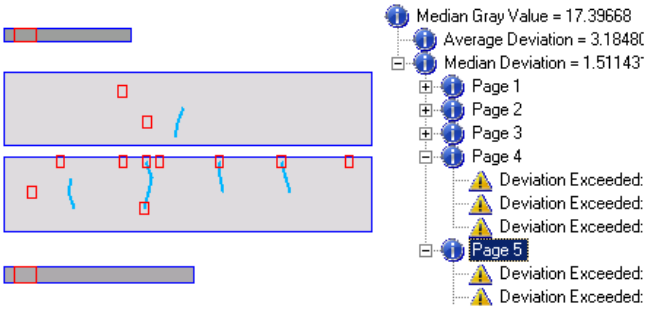

Fig. 9. Left: the document visualized in 'gray mode': text box contents are replaced by boxes with its average gray value. The option to double the deviation from the median gray value of the document (for easier comparison) is active in this example. Right: a part of the associated summary tree.

5 Results

The document analysis with our application requires several seconds (for an 8-page paper) for the full processing pipeline. Please note that the analysis results can be saved for later use without processing the source document again. The navigation and exploration of the document analysis results is interactive at all times.

Fig. 12 shows a sample output from our prototype. Even though a LaTeX template has been used, two long rivers can be seen (both at least 6 lines high, which is the threshold for error-grade rivers), as well as several gaps. The accompanying video illustrates the interactive usage of the software and the presentation of the detected problems. The detection-based navigation as well as the interactive manipulation of thresholds and ratings can also be observed in real-time.

6 Future Work

A logical next step is to analyze the use of type families, as it is known, for example, that long consecutive use of italic fonts has negative impact on the readability and frequent changes interrupt the text flow. Further, combinations of typefaces can be regarded – often too many of them are combined, or inappropriate typefaces with different characteristic styles are selected [10].

The presented approach can also be combined with OCR and more

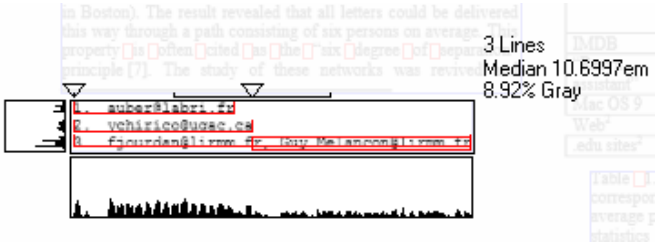

Fig. 10. Extents with strong deviation hint at lines which exceed the overall bounds.

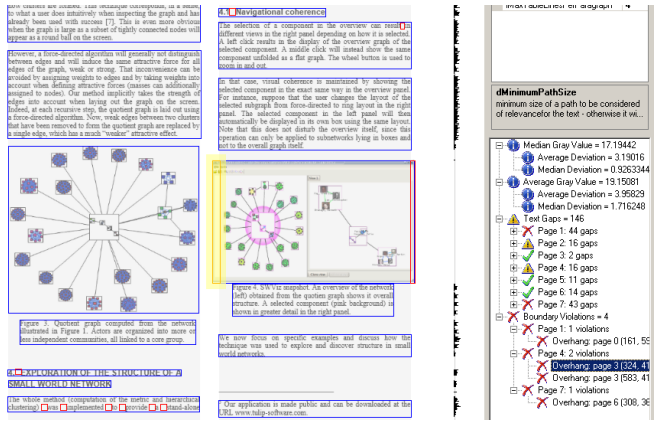

Fig. 11. Figures violating the detected column bounds are shown in this example, highlighted in yellow and red.

involved layout analysis algorithms to extend the range of supported document formats to nearly arbitrary input. Along the same lines, the table and vector figure detection heuristic (a weak point of the current implementation) is subject to future work to make it more robust.

In addition to this pure "mechanical" analysis we could imagine to extend our system to analyze if the selected typefaces are appropriate for the document content (although this is not relevant for all kinds of documents, e.g. scientific papers). We believe that a full semantic analysis and understanding of the text is most often not required for this. Instead we would like to search the document for characteristic terms or words, and find matching semantic profiles which allow to assess the chosen type faces.

7 Conclusions

In this paper we presented a novel, extensible system for the automatic analysis of document aesthetics. It allows the user to interactively browse and explore the document and the analysis results. Our implementation is very easy to use and even unexperienced users quickly find layout and typographic shortfalls in their documents and can develop a better understanding for the underlying problems. The parameters required for the document analysis are kept to a minimum and their values can be intuitively altered, if required. Lastly, our implementation is flexible and extensible and thus a starting point for a more comprehensive document analysis.

References

[1] T. M. Breuel. A review of branch-and-bound algorithms for geometric and statistical layout analysis. In *Colloque International Francophone sur l'ecrit et le Document CIFED 2004*, 2004.

[2] L. Caponetti, C. Castiello, and P. Górecki. Document page segmentation using neuro-fuzzy approach. *Applied Soft Computing*, 8(1):118–126, 2008.

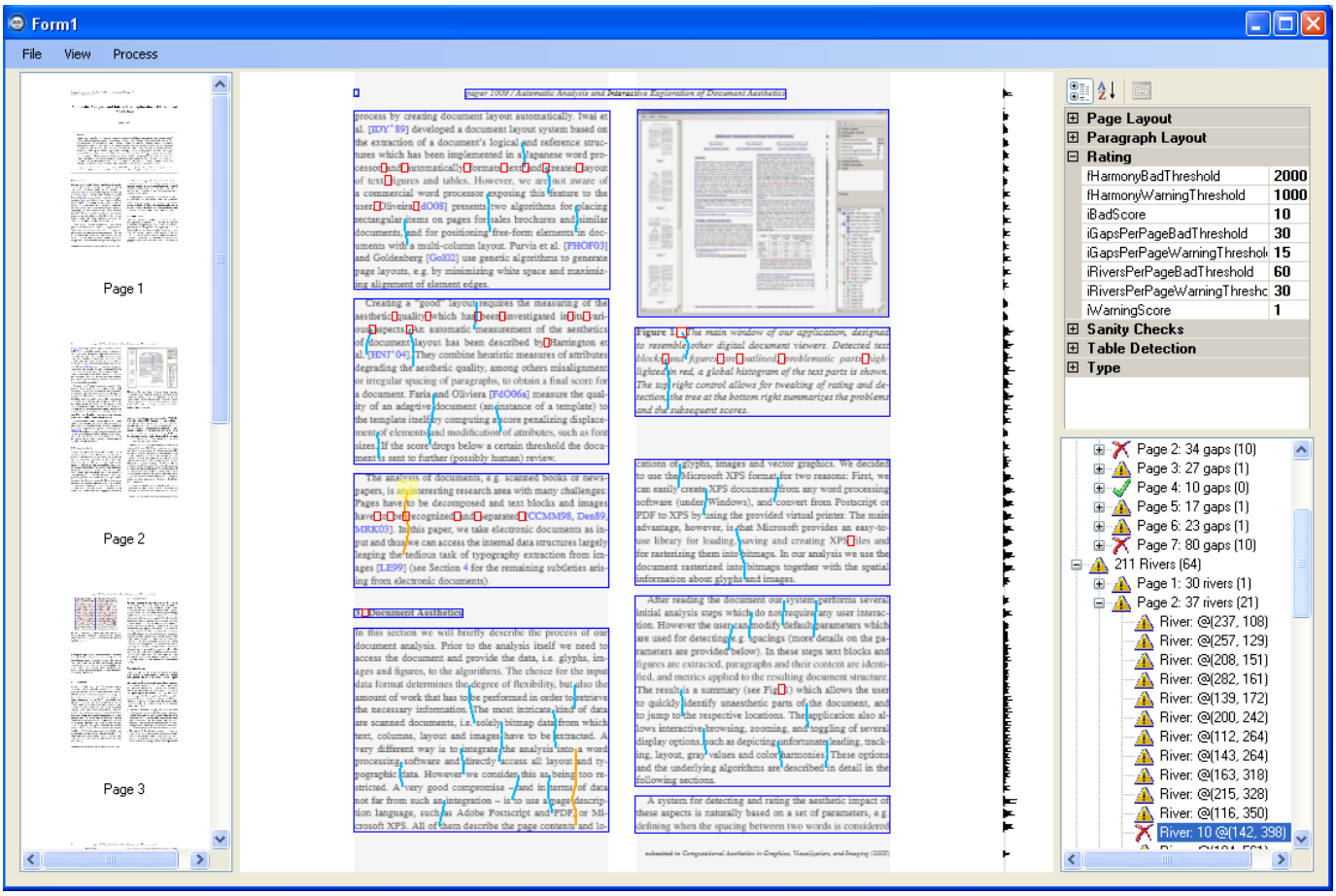

Fig. 12. Our application has analyzed a preliminary version of this paper. Although it has been produced with LaTeX(typically a very consistent type setting system), a couple of gaps and rivers are still noticeable.

[3] R. Cattoni, T. Coianiz, S. Messelodi, and C. M. Modena. Geometric layout analysis techniques for document image understanding: A review. *Technical Report TR9703-09, ITC-irst*, 1998.

[4] D. Cohen-Or, O. Sorkine, R. Gal, T. Leyvand, and Y.-Q. Xu. Color harmonization. *ACM Transactions on Graphics*, 25(3):624–630, 2006.

[5] J. B. S. de Oliveira. Two algorithms for automatic document page layout. In *DocEng '08: Proc. ACM Symposium on Document Engineering*, pages 141–149, 2008.

[6] A. Dengel. Automatic visual classification of printed documents. In *Proc. International Workshop on Industrial Applications of Machine Intelligence and Vision*, pages 276–281, 1989.

[7] A. C. Faria and J. B. S. de Oliveira. Measuring aesthetic distance between document templates and instances. In *DocEng '06: Proc. ACM Symposium on Document Engineering*, pages 13–21, 2006.

[8] A. C. Faria and J. B. S. de Oliveira. Measuring aesthetic distance between document templates and instances. In *DocEng '06: Proceedings of the 2006 ACM Symposium on Document Engineering*, pages 13–21, 2006.

[9] E. Goldenberg. Automatic layout of variable-content print data. *Technical Report HPL-2002-286, HP Laboratories Bristol*, 2002.

[10] L. Graham. *Basics of Design: Layout and Topography for Beginners*. Delmar, 2nd edition, 2005.

[11] S. J. Harrington, J. F. Naveda, R. P. Jones, P. Roetling, and N. Thakkar. Aesthetic measures for automated document layout. In *DocEng '04: Proc. of ACM Symposium on Document Engineering*, pages 109–111, 2004.

[12] M. Y. Ivory and M. A. Hearst. Improving web site design. *IEEE Internet Computing*, 6(2):56–63, 2002.

[13] I. Iwai, M. Doi, K. Yamaguchi, M. Fukui, and Y. Takebayashi. A document layout system using automatic document architecture extraction. In *CHI '89: Proc. SIGCHI Conference on Human Factors in Computing Systems*, pages 369–374, 1989.

[14] C. Jacobs, W. Li, E. Schrier, D. Bargeron, and D. Salesin. Adaptive grid-based document layout. *ACM Transactions on Graphics*, 22(3):838–847, 2003.

[15] F. LeBourgeois and H. Emptoz. Document analysis in gray level and typography extraction using character pattern redundancies. In *ICDAR '99: Proc. International Conference on Document Analysis and Recognition*, page 177. IEEE Computer Society, 1999.

[16] S. Mao, A. Rosenfeld, and T. Kanungo. Document structure analysis algorithms: A literature survey, 2003.

[17] G. Nagy and S. Seth. Hierarchical representation of optically scanned documents. In *Proc. International Conference on Pattern Recognition*, pages 347–349, 1984.

[18] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[19] L. Purvis, S. Harrington, B. O'Sullivan, and E. C. Freuder. Creating personalized documents: an optimization approach. In *DocEng '03: Proc. ACM Symposium on Document Engineering*, pages 68–77, 2003.

[20] K. Schriver. *Dynamics in document design: creating text for readers*. John Wiley and Sons, Inc., 1997.