A Smooth Transition to Modern mathoid-based Math Rendering in Wikipedia with Automatic Visual Regression Testing

Moritz Schubotz[1] and Alan P. Sexton[2]

[1] Database Systems and Information Management Group,
Technische Universität Berlin, Einsteinufer 17, 10587 Berlin, Germany
schubotz@tu-berlin.de
[2] School of Computer Science
University of Birmingham
Edgbaston, Birmingham, U.K.
a.p.sexton@cs.bham.ac.uk
http://png.formulasearchengine.com

Abstract. Pixelated images of mathematical formulae, which are inaccessible to screen readers and computer algebra systems, disappeared recently from Wikipedia. In this paper, we describe our efforts in maturing mathoid, the new services that provides better math rendering to Wikipedia, from a research prototype to a production service and a novel visual similarity image comparison tool designed for regression testing mathematical formulae rendering engines.
Currently, updates to Math rendering engines that are used in production are infrequent. Due to their high complexity and large variety of special cases, developers are intimidated by the dangers involved in introducing new features and resolving non critical problems. Today's hardware is capable of rendering large collections of mathematical contents in a reasonable amount of time. Thus, developers can run their new algorithms before using them in production. However, until now they could not identify the most significant changes in rendering due to the large data volume and necessity for human inspection of the results.
The novel image comparison tool we are proposing, will help to identify critical changes in the images and thus lower the bar for improving production level mathematical rendering engines.

Keywords: Wikipedia, Math rendering, Mathematical formulae, Image comparison

1 Introduction

In [11], we analysed different ways to improve math rendering in Wikipedia and presented our solution, which we coined mathoid. However, two years later, only a small group of registered users benefit from the improvements in rendering. In the past two years, many bugs were reported and fixed. Those bugs can be categorized into two main concerns; performance and layout. While improving performance is relatively straightforward to measure, improving the layout is still an open problem [1, 9, 10, 17]. In particular, any work in this area is hindered by the issue of ensuring that improving the layout of some aspect of math rendering does not negatively impact other aspects. When such regression testing requires humans to visually inspect and compare tens of thousands of images, progress on layout can be slow and error prone.

In this paper, we present a method to automatically compare images of mathematical formulae generated by different rendering engines and thus automate visual regression testing in this domain.

Our paper is structured as follows: We begin by presenting an overview of the improvements over the old rendering, then we describe the request flow of the new rendering process in detail and analyse its performance.

Thereafter, we describe mathpipe, the tool we built to compare different rendering mechanisms at scale and finally present the current state of our image comparison tool. Since this is work in progress, we encourage the reader to visit `http://png.formulasearchengine.com` for the latest updates.

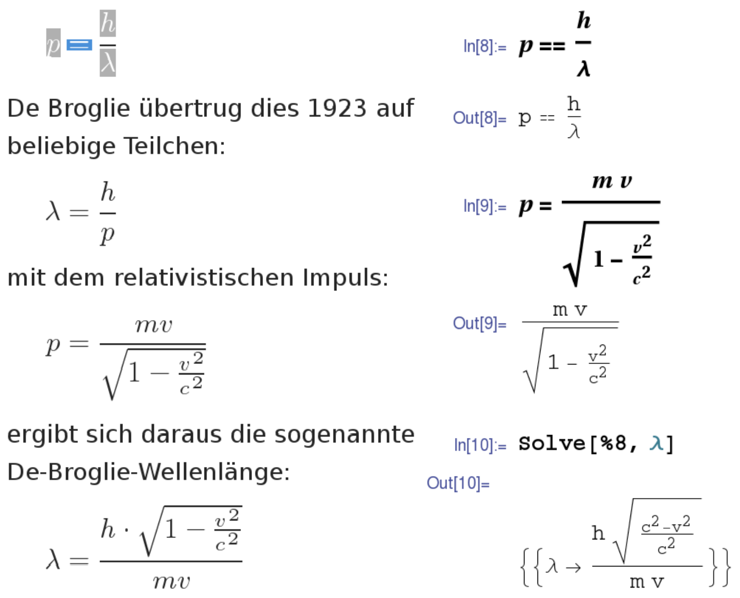
2 mathoid's Improvements Over texvc Rendering

As described in [11], mathoid provides "Robust, Scalable, Fast and Accessible Math Rendering for Wikipedia." In 2014 mathoid was one of the first Wikimedia nodeJS services, supporting the MediaWiki extension Math, which takes care of the handling of mathematical expressions used in Wikipedia and other websites running MediaWiki. Today mathoid is constructed from a huge number of services such as citoid and graphoid, which support the MediaWiki extensions Cite and Graph respectively. All of these service-supported extensions add complex functionality to MediaWiki, which would be hard to reimplement with native and efficient PHP code. Moreover, they all share the goal to be "Robust, Scalable, Fast and Accessible". The so called service-template provides a common ground for robustness and scalability and reduces the maintenance effort.

For the Math extension, that means no binaries need to be installed on the MediaWiki servers and no files in the file-system are created. Thus *robustness* is improved with respect to the old approach that relies on the texvc binary and creates local files [11], which uses MathJax [2] rather than LaTeX for the rendering. Note, that the set of supported "LaTeX like macros" is exactly the same for the old and the new system.

With the service template scaling the mathoid service is simple. As an improvement over the original version of mathoid presented in [11], now all formulae of a page are processed in parallel (cf. Section 3) instead of sequential. That way the page loading time is determined by the formula that takes the longest time to render and no longer by the sum of all the rendering times. See Section 4 for the time measurements of individual formulae.

However, most significant to the user are the accessibility component and the change in the layout itself. After the new rendering was tested on beta clusters



$$p = \frac{h}{\lambda}$$

De Broglie übertrug dies 1923 auf beliebige Teilchen:

$$\lambda = \frac{h}{p}$$

mit dem relativistischen Impuls:

$$p = \frac{mv}{\sqrt{1-\frac{v^2}{c^2}}}$$

ergibt sich daraus die sogenannte De-Broglie-Wellenlänge:

$$\lambda = \frac{h \cdot \sqrt{1-\frac{v^2}{c^2}}}{mv}$$

In[8]:= $p == \frac{h}{\lambda}$

Out[8]= $p == \frac{h}{\lambda}$

In[9]:= $p = \frac{m\,v}{\sqrt{1-\frac{v^2}{c^2}}}$

Out[9]= $\frac{m\,v}{\sqrt{1-\frac{v^2}{c^2}}}$

In[10]:= `Solve[%8, λ]`

Out[10]= $\left\{\left\{\lambda \to \frac{h\sqrt{\frac{c^2-v^2}{c^2}}}{m\,v}\right\}\right\}$

Fig. 1. Copying MathML expressions from the German version of Wikibooks to Wolfram Mathematica: On the left, there is a screen-shot, displaying a section of the book on quantum mechanics taken on May'16 2016 (the day MathML rendering became the default rendering mode for mathematical Formulae on Wikibooks) as non registered visitor using Firefox version 45, with the Native MathML plugin enabled. The photon momentum was selected, to demonstrate the bounding boxes of the symbols, and to copy and paste it to the computer Algebra System Mathematica (screen-shot on the right) as `In[8]`. The same step was performed for the relativistic momentum `In[9]`. Thereafter, an additional = sign has been manually inserted to `In[8]` and `Solve[%8,λ]` was typed to compute the De-Broglie wavelength.

it has been enabled on Wikidata and the German version of Wikibooks on May 16th, 2016, and was enabled globally on May 31th, 2016.

As visualized in Figure 1, the MathML code is available to the browser. This allows screen readers to verbalize the formulae from the additional information that is neither available from the new SVG image nor from the old PNG image. The documentation page of the Math extension contains links to examples of this feature and [3, 4, 8, 12, 14] provide further information on that topic.

However, since MathML requires certain fonts that are not available on all operating systems, MathML will only be provided to people that have installed a browser extension that indicates that their browser actually provides good MathML rendering. This could be either Mozilla Firefox with the Native MathML plugin [16] or Internet Explorer with the MathPlayer plugin [13]. The majority will see SVG images, which is already an improvement, especially for high resolution displays, or in situations were the formulae are printed.

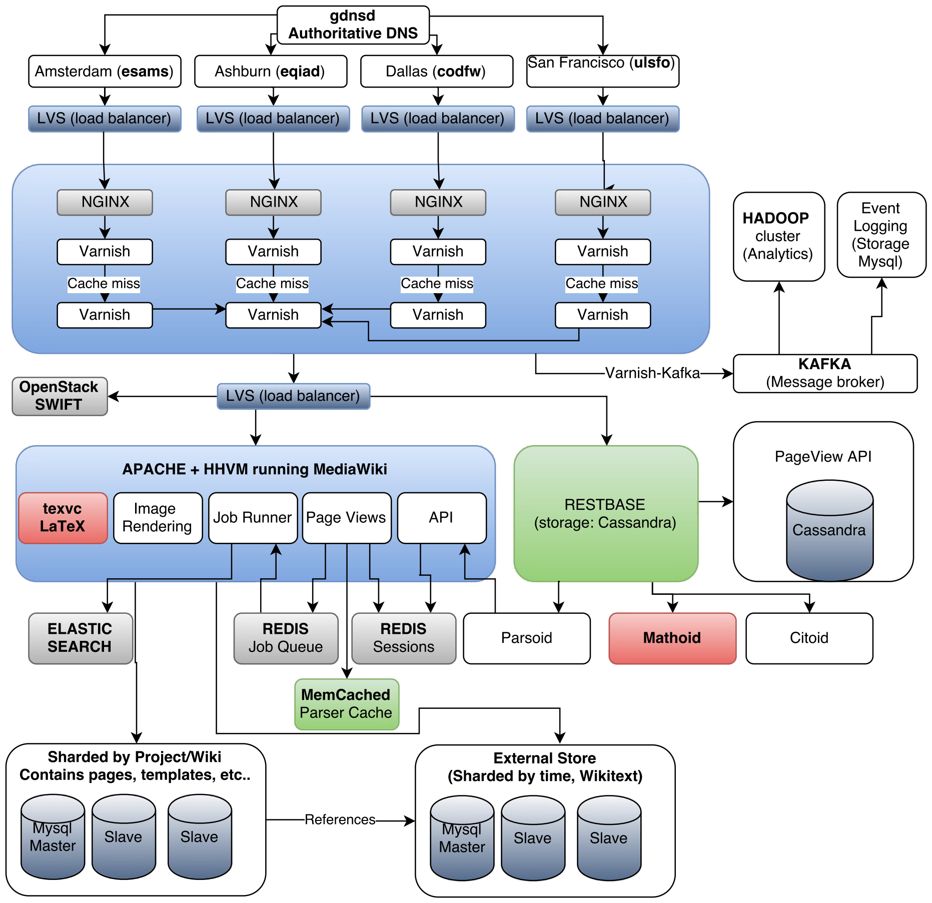


Fig. 2. Overview of the Wikimedia infrastructure (based on an image from Luca Toscano, which was released under the Creative Commons (CC3) license.)

3 Internals of the New mathoid Rendering

While the Wikipedia request flow as visualized in Figure 2 is quite complicated, we summarize a typical visit of a page that contains mathematical formulae below.

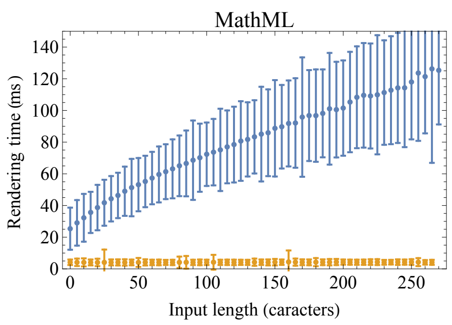
1. A user requests a page
2. If that page is not cached (neither physically close to the user or abroad), MediaWiki renders the page from the internal representation (wikitext) to HTML. Note that this is only true for a regular visit. If the user wants to edit the page using the visual editor the page would be rendered by parsoid (cf. 3).
3. All math elements on a page are collected

4. A bundle of requests (the size of this bundle is a performance tuning parameter) is sent to restbase (check request)
5. restbase checks the requests; either the result is cached in the internal cassandra data store
6. or it contacts mathoid which calls texvcinfo which calls texvcjs (this will be explained later in detail).
7. restbase returns the results of the check and other information about the formulae i.e. the sanitized tex, a hash and the SVG.
8. After a bundle of check requests is returned to the MediaWiki, the extension evaluates each check response, and either replaces the Math tag with an error message in case the check request was not successful or collects the hash of the MathML and SVG.
9. Now, the math extension has collected the hashes of all valid input formulae and a second request bundle is sent to restbase requesting the MathML rendering.
10. restbase responds with the MathML rendering from storage. In the header of the request response, the SVG dimensions are stored.
11. Thereafter, the math extension replaces the math tags with MathML and a link to the SVG fallback image. The style-sheet is configured in a way that the fallback image is visible by default and the MathML element is invisible.
12. Finally, MediaWiki returns the HTML of the page, including the links to the SVG fallback images.
13. If the browser of the user does not overwrite the visibility information of the MathML and SVG, the browser sends a request for each individual formula.
14. While most of the image requests will be served from either the browser cache or varnish (a caching http reverse proxy [15]), the rest will go directly to restbase without contacting MediaWiki.

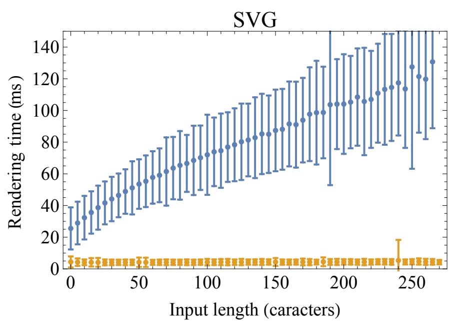
4 Measuring mathoid's performance

To evaluate the absolute performance of the new rendering, we performed measurements with the following set-up: We use two identical work stations (Intel(R) Core(tm) i7-3770 CPU @ 3.40GHz 16 GB RAM, Gigabit Ethernet, 2x1 TB HDD, Ubuntu 14 LTS). One system (hereafter referred to as the server) has the Cassandra storage engine, the restbase Client and mathoid. The other system (client) has MediaWiki with the extension Math and MathSearch, it uses as many parallel workers as CPUs are present (8) to call to get the data as would be done in production. Our performance tests script can be downloaded from github script.

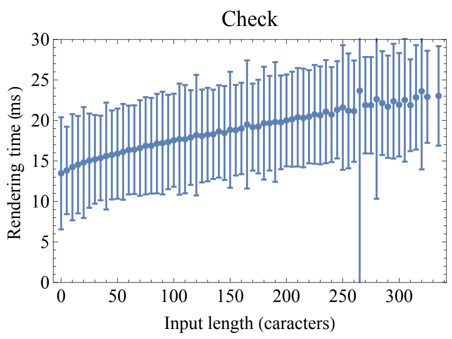
For each formula, the requests are sent in as described in 3 and the time is measured. Note that, for this test, we did not use the complete endpoint but the MathML or SVG endpoint respectively. For each formula we randomly chose if it was rendered first in SVG or MathML mode. The measurement results are visualized in Figure 3. The figures indicate an almost linear relationship between the formula length and the rendering time. It should be noted that formulae whose request could be answered by restbase from the cassandra storage without



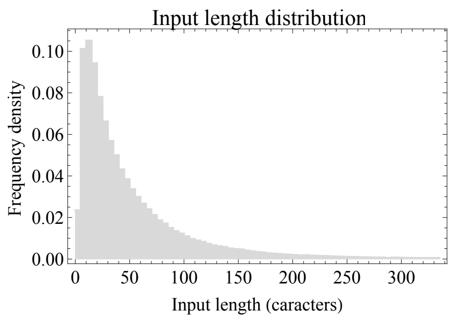
(a) MathML rendering



(b) SVG rendering



(c) Input checking



(d) Distribution of input length

Fig. 3. Rendering and verification time versus input length: Measurements for the Test Collection, with different rendering modes. (a-c) shows the rendering, times and standard deviations respectively. (a-b) Include two series, where the yellow (lower) series is the cached result.

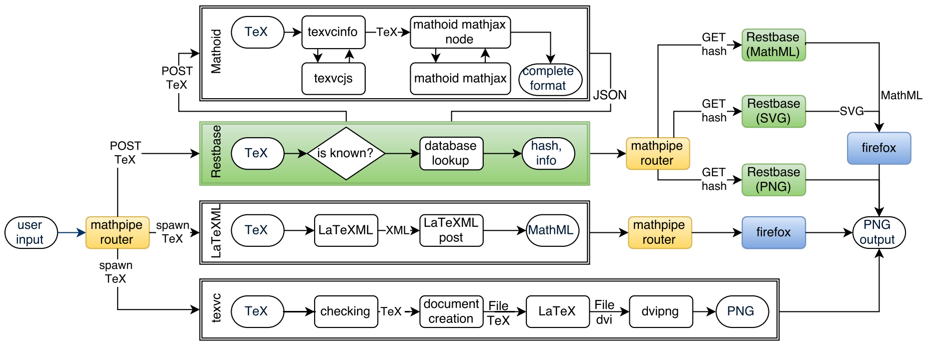


Fig. 4. mathpipe processing chain

contacting mathoid were fast, independent of the length of the input tex string. The average was around 4ms. These measurements indicate that, in the average case, where already rendered formulae get requested again, most of the time is spend at the checking phase (>20ms). Therefore, the most recent version of restbase, which has not been tested yet, also caches the check responses from mathoid. This is supposed to reduce the checking significantly.

Compared to the old LaTeX based rendering, which created a LaTeX document for each formulae, rendered that document and thereafter converted it to a PNG image, this is a significant performance improvement.

5 Comparing Different Rendering Engines with mathpipe

mathpipe is a program specifically designed for testing the different options for rendering mathematics on the web. It starts with user input (currently limited to texvc dialects) and goes via different routes to PNG output. All current routes are visualized in Figure 4. From there on, we perform the analysis of different generated png images and calculate the similarity scores as described below. mathpipe has two modes of operation. A command-line mode that can be used for batch jobs and a web interface. The command-line will be used to identify the formulae that have the biggest divergence. The web interface will provide a quick comparison for humans to manually investigate the difference outputs and check the derived results. We encourage the reader to visit `http://png.formulasearchengine.com` and test an instance of our service.

6 Image comparison

The lack of practical automatic regression testing on mathpipe-generated renderings of the huge collection of mathematical formulae in Wikipedia has been a serious hinderance to speedy improvement of rendering quality and performance. Currently, any change made to the rendering pipeline can only be checked visually by humans. Even then, it is very easy for a human inspector to overlook some small but important error that has arisen in the rendering. The motivation for automating this procedure with a suitably high precision machine-based approach is clear.

The problem to be solved involves comparing two PNG images rendered using different methods from the same input source. These methods vary in their choice of fonts (and therefore also the thickness of the character strokes), the resolutions of rendered images, the approaches to anti-aliasing and background transparency rendering, spacing between characters and relative sizes of characters. Some of the methods also generate extra empty padding around the images and most generate some form of greyscale or colour images, the details of which can vary, rather than monochrome.

In comparing these images, our ideal is to make judgements about the relative validity of these renderings as an extremely careful human inspector would. Differences insignificant to the readability and correct interpretation of the underlying mathematical formulas should be overlooked. For example:

- changes of font where corresponding characters from the two images are visually very similar.
- differences in character spacing in the two images which are within aesthetically reasonable bounds.
- characters that are spatially discrete but close together in one rendering may touch in another. This is often due to the rendering resolution chosen together with the antialiasing approach used. If the resulting touching characters are perfectly readable, then this should not cause significant concern or trigger reporting of errors unless the user is explicitly looking for such problems.
- with symptoms very similar to the previous case, a single character in one image may be broken into two or more characters in the other. This arises not from characters touching, but by the renderer building a character, often an extendable fence or integral character, out of separate character components. The renderer is supposed to make such characters touch, but may fail to do so completely. Such a case rarely interferes with the readability of the image, but should be overlooked or highlighted as the user requires.

Conversely, significant issues should be identified and reported:

- characters in one image but missing from the other.
- characters that have significantly different shapes, for example if the character is missing or mis-indexed in the font used in one of the images.
- significant differences in spacing that may confuse the reader, e.g. a super-superscript that is rendered on the same baseline as a superscript.
- touching characters that are unreadably overlapped.
- broken characters that are so spatially separated that it confuses the reader.

The tolerances used to determine many of the fuzzy quality issues (i.e. *"visually similar"*, *"aesthetically reasonable"*, etc.) described above should be choosable by the user to correspond to the user's purpose at the time, although defaults should be set to practical values for general purpose regression testing.

In general, the system should be biased to not overlook serious issues, even if that means that more insignificant differences are reported as significant (false positives).

Finally, a practical and robust way of reporting issues found is necessary to assist developers in quickly identifying problems and their sources.

6.1 Approach

The constraints of a solution as described above mean that a purely image-focused approach is unlikely to be successful. Instead we have taken an approach based on a structural analysis of the image into a form where we can deal with the image components and their relationships more abstractly. This form is based on *connected components*. A connected component in a monochrome image is a maximal set of foreground pixels that are horizontally, vertically or diagonally connected. Thus an equals symbol, "=" and the letter "i" both have two connected components, while an equivalence symbol "≡" and a capital greek letter xi, "Ξ", both have three.

In brief, the general approach we have chosen involves binarising the images, decomposing them into sets of connected components, scaling the meta-data (bounding-box information) about the connected components to make them comparable between the images, identifying viable pairings of corresponding connected components in the different images based on relative positions, aspect ratios, and size, verifying the pairings using features of the underlying connected component shape and pixel distributions, and finally treating connected components not successfully paired as possible candidates for touching/broken character analysis.

It should be noted that, though we use techniques borrowed from the areas of document analysis and optical character recognition [5], we do not attempt to classify characters or interpret the mathematical formulae in any way, and hence we avoid the problems of classifier training, mis-classifications and incompleteness of a model for the structure of mathematical formulae. More precisely, we borrow only methods of binarisation, connected component analysis and some simple shape feature extraction methods that are commonly used in document analysis for character classification purposes but here are used only to provide a basis for metric shape similarity measurements between corresponding components of the two images.

We shall now discuss each part of the approach in detail.

Binarisation and Connected Component Analysis Connected component analysis requires binary choices between foreground and background images. Since these images are generated rather than scanned, a very simple binarisation method selecting the pixel class based only on the RGBA (red-green-blue-alpha) value of the pixel itself is sufficient. The only issue of (minor) concern is the different approaches to background colours and anti-aliasing. Background pixels and anti-aliasing in an effectively monochrome image is best implemented by using the same (foreground) colour for all pixels but varying the transparency of the anti-aliased pixels down to fully transparent for background pixels. However, some images do change the grey-scale/colour as well as the transparency while others do not use transparency at all but merely blend the foreground into a different background colour. Binarisation must be aware of these variations and manage them all.

Connected component analysis is accomplished using the algorithms described in [6, 7]

Cropping and Scaling Removing extraneous background padding is a simple matter of cropping to the size of the rectangle union of the bounding boxes of the identified connected components.

Scaling is slightly more subtle; Since these images can be of relatively low resolution, the size of a pixel relative to the whole connected component can be significant. Hence scaling the image to a common size can introduce discretisation artifacts that impede the analysis. Hence the images themselves are not scaled but a scaled version of the connected component bounding box information is added. The scale factors are chosen so that one of the images' information is scaled to correspond to an image of width 1.0 (using floating point rather than integer numbers of pixels) and a height that maintains the original aspect ratio. The other image is scaled so that the image is exactly the same size, even if that distorts its aspect ratio. This results in relative positions of connected components in the scale spaces being directly comparable. The underlying image pixels are not changed so shape analysis is not affected by the scaling.

Simple Component Pairing At this point we attempt to pair off connected components in one image with the corresponding ones in another. A simple matching of corresponding positions does not work for a number of reasons:

- While scaling ensures that the left-most and rightmost characters are in very close to the same horizontal position, variations in spacing may mean that characters in the centre of the image are significantly out of horizontal alignment with the corresponding character of the other image, and, indeed, may be in exactly the same relative position as a non-corresponding character. Ditto for vertical alignments.
- Multiple characters in one image may be touching and therefore occur as a single connected component while they are separated in the other. Therefore the correct pairing should be of at least one connected component with a set of connected components.
- Depending on the actual parameters used, a component of one image may viably, but incorrectly, be paired with any one of a number of different components from the other. We call such a case a "*multi-match*"
- Because of variations of character fonts, positioning and sizing, any choice of discrimination parameters that correctly accept/reject a pairing in one part of the image tends to be wrong for another part of the image. An adaptive

approach that varies the parameters over different parts of the image might be possible but it is not clear what criterion can be used to accomplish it without more in-depth classification or recognition.

For these reasons we chose an iterative approach, where we start with very tight constraints on acceptable parameters to pair components based on their position, aspect ratio and size (i.e. area of the bounding boxes), with a verification element based on a metric shape similarity measure, to guarantee reliability of the pairing. This ensures that any pairings found are robust and can be removed from consideration. Repeating the process with slightly relaxed parameters on the thinned out set of remaining components allows robust pairings to be made where, with the full set of components, an unambiguous pairing would not have been possible. This cycle continues until one of the following holds:

– no unpaired components remain (in which case the two images can be considered to have passed the comparison check) or
– any further pairings require relaxing the parameters beyond their upper limits or
– there is a component which, at the current parameter setting, could viably be paired with more than one component (a multi-match).

In the latter two cases further analysis is necessary.

Touching Component Analysis At this point, assuming there are multi-matches or unpaired components remaining, there is a set of components from each image that could not be paired with components from the others. The only allowable remaining situation that would not justify reporting this as an error is if components are touching in one image but not in the other, and there may be multiple separate cases involved. Simply trying all possible combinations of ways that components could touch is computationally infeasible for our purposes.

To find such cases, note that a touching component in one image that corresponds to a group of components from the other is necessarily larger than the individual sub-components. Therefore we work iteratively starting at the largest remaining unpaired component of **both** images, the *target* component, and select the set of unpaired components (the *candidate* components) from the other image that overlap with an expanded version of the bounding box of the target component. This excludes from consideration components that should not realistically be considered as candidates, but the expansion allows for some distortion of the spacing between the different images.

Even if the target does correspond to some of the candidate components, it may not correspond to all. For example, consider the following expressions where the left is from one image and the right from another:

$$\sqrt{x^K} \qquad \sqrt{x^K}$$

Here the target would be the touching $\sqrt{}^{K}$ component from the right and the candidate set would include all three components from the left, because all three are within the appropriate space. However, the x should not be included in the pairing or it will cause the shape matching to fail.

For these reasons, all combinations of the candidate components are considered by calculating the comparison attributes of the union of each combination, where the attributes involved are; centre of the bounding box, aspect ratio, area of bounding box and, only if the checking of those attributes passes, the more expensive shape similarity test. The final shape similarity test ensures that the resulting merged shape is still readable. The limitation of the set of candidate characters to those that overlap the expanded target component bounding box ensures computational feasibility.

If a pairing is found, the components are removed and the process repeats on any remaining unpaired components until exhaustion of unpaired components or failure to find a pairing.

Reporting of Results The results of the above analysis is four-fold:

1. *Simple matches:* A set of pairs of single compatible components that are within appropriate matching parameters.
2. *Touching matches:* A set of pairs of sets of components corresponding to target/candidate set matching pairs for touching component cases.
3. *First image unpaired:* A set of components from the first image that could not be paired with corresponding components from the second.
4. *Second image unpaired:* A set of components from the second image that could not be paired with corresponding components from the first.

The test passes if all except *simple matches* is empty. It can be considered to pass if *touching matches* is also non-empty and the user chooses that option. A short narrative report is generated of the results to a log file or to the standard output stream. However, a textual description of the problems when errors occur is frustratingly difficult to interpret. Hence we also generate two error images, These are cropped binarised images but with unpaired components drawn in one colour, target components of touching matches in another and the corresponding matching candidate components in a third. This is usually sufficient for immediate identification of the problem to the user. However, sometimes it is necessary to investigate more directly the relative positioning or sizing issues that triggered the comparison failure so we provide a python plugin for the GIMP image processing tool that allows the two error images to be loaded as separate layers, scaled and positioned to exactly the same size and position so that, by varying the transparency of the layers with the GIMP's layer tool, one can precisely see the issues involved.

An example of the error results when touching matches are found are shown in Figure 5

7 Further work

As a work-in-progress, there is still much work to do in refining and improving the image comparison tool, testing and evaluating it on the various mathpipe

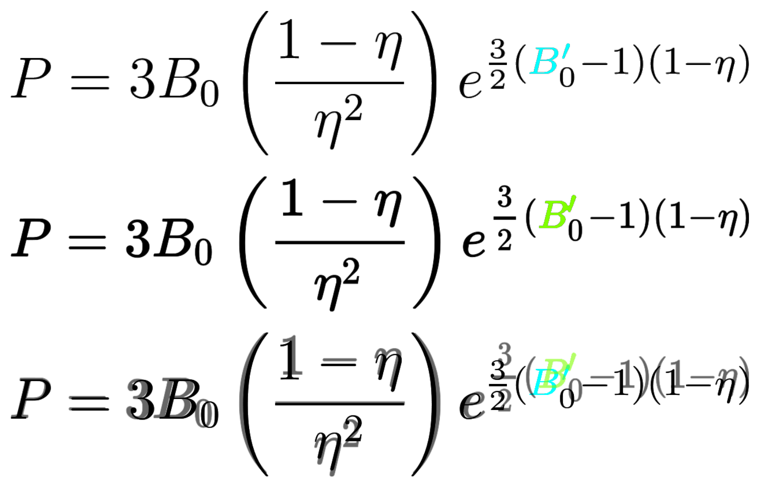


Fig. 5. Error images from two different renderings of the same formula. The orginal of the second image has been artificially edited to force the B' characters to touch, as indicated by the green colour. In the first image the two characters that form the matching candidates are in blue. The third image shows the result when the two images are scaled and overlaid using the GIMP plugin to demonstrate differences in the spacing and character shapes between the two images.

rendering pipelines and, eventually, building it into the mathpipe construction toolchain.

8 Conclusion

We have presented Wikipedia's new approach to higher performance, higher quality, scalable, accessible mathematical formula rendering and delivery and the work we carried out in performance analysis of its results that demonstrates its huge performance improvement over the previous approach.

We have also presented our work on addressing a critical need for speedier and more robust development of further improvement in mathematical formula rendering; namely an image comparison program suited for use in automatic regression testing of mathematical formula rendering software. While this program is still under heavy development, it is already showing promise in providing support for more aggressive development of new mathoid-based rendering methods.

Acknowledgments. The authors would like to thank the Wikimedia Foundation employees, especially Marko Obrovac and Gabriel Wicke for their code reviews, guidance, contributions, and discussions in the context of mathoid.

Bibliography

[1] Börjesson, E. and Feldt, R. (2012). Automated system testing using visual GUI testing tools: A comparative study in industry. In Antoniol, G., Bertolino, A., and Labiche, Y., editors, *5th IEEE Int. Conf. on Software Testing, Verification and Validation, ICST 2012*, pages 350–359. IEEE Computer Society.

[2] Cervone, D. (2012). Mathjax: A Platform for Mathematics on the Web. *Notices of the American Mathematical Society*, 59(2):312–316.

[3] Chisholm, W., Vanderheiden, G., and Jacobs, I. (2001). Web content accessibility guidelines 1.0. *Interactions*, 8(4):35–54.

[4] Cooper, M., Lowe, T., and Taylor, M. (2008). Access to mathematics in web resources for people with a visual impairment. In Miesenberger, K., Klaus, J., Zagler, W., and Karshmer, A., editors, *Computers Helping People with Special Needs*, volume 5105 of *LNCS*, pages 926–933. Springer.

[5] Doermann, D. and Tombre, K., editors (2014). *Handbook of Document Image Processing and Recognition*. Springer.

[6] He, L., Chao, Y., and Suzuki, K. (2008). A run-based two-scan labeling algorithm. *IEEE Transactions on Image Processing*, 17(5):749–756.

[7] He, L., Chao, Y., Suzuki, K., and Wu, K. (2009). Fast connected-component labeling. *Pattern Recognition*, 42(9):1977–1987.

[8] Maddox, S. (2007). Mathematical equations in Braille. *Maths, Stats and Operations Research (MSOR) Connections*, 7(2):45–48.

[9] Memon, A., Nagarajan, A., and Xie, Q. (2005). Automating regression testing for evolving GUI software. *J. of Software Maintenance*, 17(1):27–64.

[10] Memon, A. M. (2008). Automatically repairing event sequence-based GUI test suites for regression testing. *ACM Trans. Softw. Eng. Methodol.*, 18(2).

[11] Schubotz, M. and Wicke, G. (2014). Mathoid: robust, scalable, fast and accessible math rendering for wikipedia. In Watt, S. M., Davenport, J. H., Sexton, A. P., Sojka, P., and Urban, J., editors, *Proc. Int. Conf. on Intelligent Computer Mathematics (CICM 2014)*, pages 224–235. Springer.

[12] Soiffer, N. (2005a). MathPlayer. In *Proc. 7th Int. ACM Conf. on Computers and Accessibility – ASSETS 2005*, page 204, New York. ACM Press.

[13] Soiffer, N. (2005b). MathPlayer. In *Proc. 7th Int. ACM Conf. on Computers and Accessibility – ASSETS 2005*, page 204, New York, New York, USA. ACM Press.

[14] Sorge, V., Chen, V., Raman, T., and Tseng, D. (2014). Towards making mathematics a first class citizen in general screen readers. In *11th Web for All Conference*, Seoul, Korea, 6–9 April 2014. ACM.

[15] Varnish (2016). Varnish HTTP cache. `https://varnish-cache.org/`. Accessed: 25 June 2016.

[16] Wang, F. (2016). Native MathML. `https://addons.mozilla.org/en-US/firefox/addon/native-mathml`. seen May, 2016.

[17] Yoo, S. and Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. *Softw. Test., Verif. Reliab.*, 22(2):67–120.