IMPROVING TYPOGRAPHY AND MINIMISING COMPUTATION FOR DOCUMENTS WITH SCALABLE LAYOUTS

Alexander J. Pinkney, B.Sc. (Hons)

THESIS SUBMITTED TO THE UNIVERSITY OF NOTTINGHAM
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
JULY 2015

ABSTRACT

Since the 1980s, two paradigms have dominated the representation of formatted electronic documents: flowable and fixed. Flowable formats, such as HTML, EPUB, or those used by word processors, allow documents to scale to any arbitrary page size, but typographical compromises must be made since the layout is computed in real time, and is re-computed each time the document is displayed. Conversely, fixed formats such as SVG or PDF are afforded the potential for arbitrarily complex typography, but are constrained to the fixed layout that is set at the time of creation. With the recent surge in popularity of low-powered portable reading devices — from tablets to e-readers to mobile phones — there is an expectation that documents should scale to any size, maintain their high-quality typography, and not provide unnecessary strain on an already overloaded battery.

This thesis defines a novel paradigm for electronic document representation — the *Malleable Document* — whereby documents are partially typeset at the time of creation, leaving enough flexibility that their content can be flowed to arbitrary page sizes with minimal computation. One tradeoff encountered is that of increased file size, and this is addressed with a bespoke, computationally-light compression scheme.

A sample implementation is presented that transforms documents from a source format into Malleable Document format, alongside a lightweight display engine that enables the documents to be viewed and resized on a wide range of devices, mobile and otherwise. Reviews of the technical aspects and a user study to evaluate the quality of the system's rendering and layout show that the Malleable Document paradigm is a promising alternative to both fixed and flowable formats, and builds upon the best of both approaches.

In a badly designed book, the letters mill and stand like starving horses in a field. In a book designed by rote, they sit like stale bread and mutton on the page. In a well-made book, where designer, compositor and printer have all done their jobs, no matter how many thousands of lines and pages they must occupy, the letters are alive. They dance in their seats. Sometimes they rise and dance in the margins and aisles.

Robert Bringhurst, The Elements of Typographic Style

ACKNOWLEDGMENTS

People who have got me through this

academically, socially, pastorally, alcoholically. . .

you know who you are![1]

Thanks

[1] If you're not sure, it probably isn't you

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

0 FOREWORD AND OVERVIEW

This thesis is structured in a slightly unconventional manner: its literature review is spread amongst the chapters, and sources are discussed within relevant areas of the text. Below is an outline of the thesis structure:

Chapter 1 provides an overview of the present state of affairs of the electronic document world, with particular emphasis upon the technologies used in contemporary ebook readers, and their benefits and drawbacks.

Chapter 2 takes a brief look at the history of movable type, and some of the techniques tradionally used in newspaper typesetting. Using some of these techniques as a basis, it then describes a novel paradigm for document representation that allows documents to fit a wide variety of screen sizes whilst maintaining high typographic quality. It then outlines a prototype implementation of a system to generate and display simple documents. Work in this chapter was published and presented at DocEng'11 in Mountain View, CA, USA [PBB11].

Chapter 3 introduces a complete reimplementation of the system described in Chapter 2 to enable its use on mobile devices, and extends the devised model to include support for floating items such as figures and tables. Work in this chapter was published and presented at DocEng'13 in Florence, Italy [PBB13].

Chapter 4 addresses the issue of enlarged file sizes, and details various methods to keep file sizes to a minimum, where possible avoiding unnecessary increases in computational complexity that would counter the work described in Chapters 2 and 3.

Chapter 5 provides technical and aesthetic analysis, focusing on the quantitative and qualitative aspects of the system as it runs at view-time, and includes the results of a user study of the system developed in the previous chapter.

Finally, Chapter 6 evaluates the success of the project, and details areas of potential new research that have been encountered throughout the process.

Part I

MOTIVATION

1

THE RISE OF EBOOKS

In this chapter we look in depth at the status quo of the electronic document world: we examine the hardware used to display documents, we examine the paradigms used to represent documents, we examine what is considered to be good typesetting, and then in the context of this information, we examine the limitations currently enforced upon us.

In the past five years, the surge in popularity of tablets and dedicated ebook readers has vastly increased the sale and distribution of ebooks. In August 2012, it was widely reported in the media[hex12] that Amazon's Kindle Store sales were outstripping print book sales by 114 to 100. This figure did not include free ebooks "sold" through the Kindle Store, which would skew the figures significantly further.

Project Gutenberg, a digital online library that (as of January 2015) hosts over 46,000 freely downloadable out-of-copyright ebooks, regularly exceeds 150,000 downloads *per day*.[1]

Ebook readers have now become a commodity item, and although their displays are becoming increasingly print-like, the typography and formatting that they offer does not meet the high standards of traditionally typeset documents.

Whilst at first glance this may not seem like a difficult problem to solve, the expectation that ebooks should scale to fit many devices and adapt to the reading preferences of many different users makes it far from trivial, as we shall see further on in this chapter.

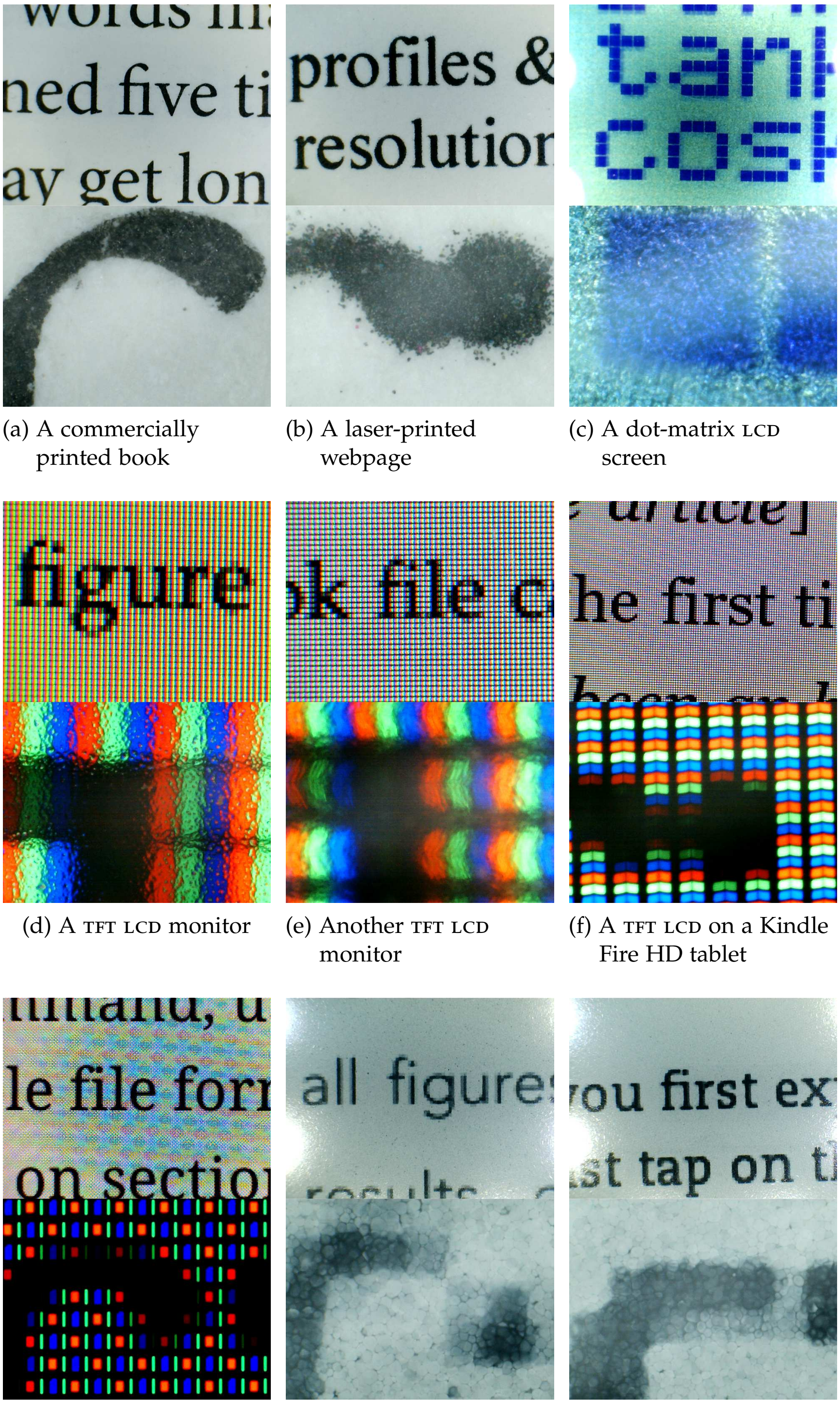
1.1 DEVICES

In addition to dedicated ebook readers, such as the Amazon Kindle and the Kobo eReader, many other devices (such as tablets, mobile phones, laptops, and desktop PCs) can be equipped to read ebook files. Indeed, virtually every modern device with a screen can be equipped to read ebooks.

Amazon distributes software that allows Kindle format books to be read on Android and iOS tablets and smartphones, and on Windows and OS X, in addition to its own range of Kindle hardware

The screen technologies used in these devices have vastly improved over the past decade or so, from the introduction of electronic paper displays that provide a reading experience very similar to that of real paper, to the enormous advances in LCD and OLED screens, which now often have resolutions high enough that it is difficult to resolve individual pixels with the naked eye. Figure 1 shows some examples

[1] See http://www.gutenberg.org/browse/scores/pretty-pictures for up-to-date statistics.



(a) A commercially printed book

(b) A laser-printed webpage

(c) A dot-matrix LCD screen

(d) A TFT LCD monitor

(e) Another TFT LCD monitor

(f) A TFT LCD on a Kindle Fire HD tablet

(g) A Super AMOLED display on a Galaxy Nexus smartphone

(h) An electronic paper display on a Kobo eReader

(i) An electronic paper display on a Kindle Touch

Figure 1: Examples of some document display technologies, magnified about 25 times (top halves of images) and 350 times (bottom halves). These images were all captured with a £25 USB microscope purchased from Amazon. Note that the resolutions of the screens in (f) and (g) are close to that of the microscope used to capture the image, hence the aliasing effect.

of document display technologies — note the visual similarity of the "real" print (Figure 1a and 1b) to the electronic paper displays (Figure 1h and 1i).

As a consequence of this, ebook file formats must be flexible enough that their content can be displayed and read on a vast range of devices with differing screen sizes and types.

1.2 "GOOD" TYPESETTING

Generally speaking, the better the quality of a document's typography, the less it should be noticed by the reader. Good typography should be transparent, in order that the reader may concentrate upon the content of the document, rather than be distracted by its presentation.

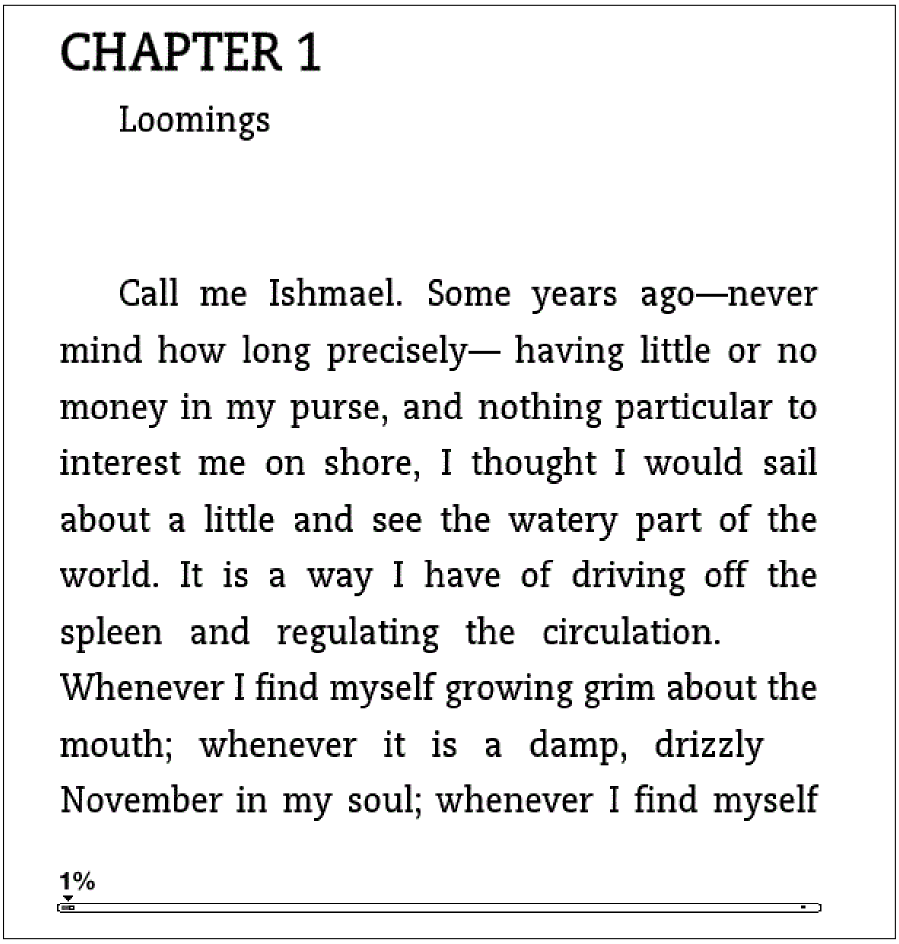
Many studies[MR91, Hil99, Bri08, Voo11, LB11] conclude that the readability of text is inextricably linked to the quality of its typography. In particular, it is stated in *The Magic of Reading*[Hil99] that both regularity of whitespace between words and the evenness of line lengths are of importance — and the only way to achieve this is to use a line-breaking algorithm that attempts to make line lengths as even as possible. Unfortunately, producing well-typeset text can be an extremely complex process [HLM09].

1.2.1 *Hyphenation and Line-Breaking*

Ebook readers typically use a "greedy" algorithm to lay out their text — that is, they place as many words as will fit onto the current line without exceeding it, then start a new line and continue. Although this algorithm is optimal in that it will always fit text onto the fewest possible lines, it often causes consecutive lines to have wildly varying lengths, accentuating either the "ragged right" effect of the text, or, in the case of fully-justified text, the inter-word spacing. In general, ebook readers will only hyphenate in extreme cases — indeed the Kindle seems not to do so at all, to the detriment of its typography (see Figure 2).

Donald E. Knuth and Michael F. Plass[KP81] developed a more advanced line-breaking algorithm (now used by TEX) that attempts to minimise large discrepancies between consecutive lines by considering each paragraph as a whole. TEX also uses the hyphenation algorithm designed by Franklin Liang[Lia83] (another of Knuth's grad students) which has been ported to many other applications. Figure 3 compares the Knuth-Plass algorithm against the default layout of a web browser (a first fit, greedy approach).

Knuth and Plass's line breaking algorithm, in conjunction with Liang's hyphenation algorithm, breaks paragraphs into lines of text to fit a page, resulting in what can be considered an aesthetically optimal configuration. TEX's default behaviour is then to alter the



CHAPTER 1

Loomings

Call me Ishmael. Some years ago—never
mind how long precisely— having little or no
money in my purse, and nothing particular to
interest me on shore, I thought I would sail
about a little and see the watery part of the
world. It is a way I have of driving off the
spleen and regulating the circulation.
Whenever I find myself growing grim about the
mouth; whenever it is a damp, drizzly
November in my soul; whenever I find myself

1%

Figure 2: The Kindle fully justifies its text, falling back to ragged-right when inter-word spacing would become too large. Its lack of hyphenation exacerbates this problem.

spacing between words in order to justify the line to fit the measure of the page. The Knuth-Plass algorithm nominally runs in $O(n^2)$ time (compared with $O(n)$ for a greedy first-fit approach), although with some pruning, the effective complexity can be reduced to nearer $O(n)$ [HL87, EG92, HLM09]. In practice though, large constant factors still make the algorithm slow. In any case, the Knuth-Plass algorithm is certainly not the last word in line-breaking algorithms: for example, it has no mechanism to avoid (nor indeed any knowledge of) vertical rivers of whitespace [MR91]. Inevitably, adding support to avoid rivers, and for any of the other nuances used by hand compositors, would add further complexity.

1.2.2 *Microtypographical Techniques*

Other techniques employed during hand-typesetting, and high-quality electronic typesetting, include the use of what is often termed *micro-typography*,[HLM09] such as the use of kerning and ligatures. Kerning

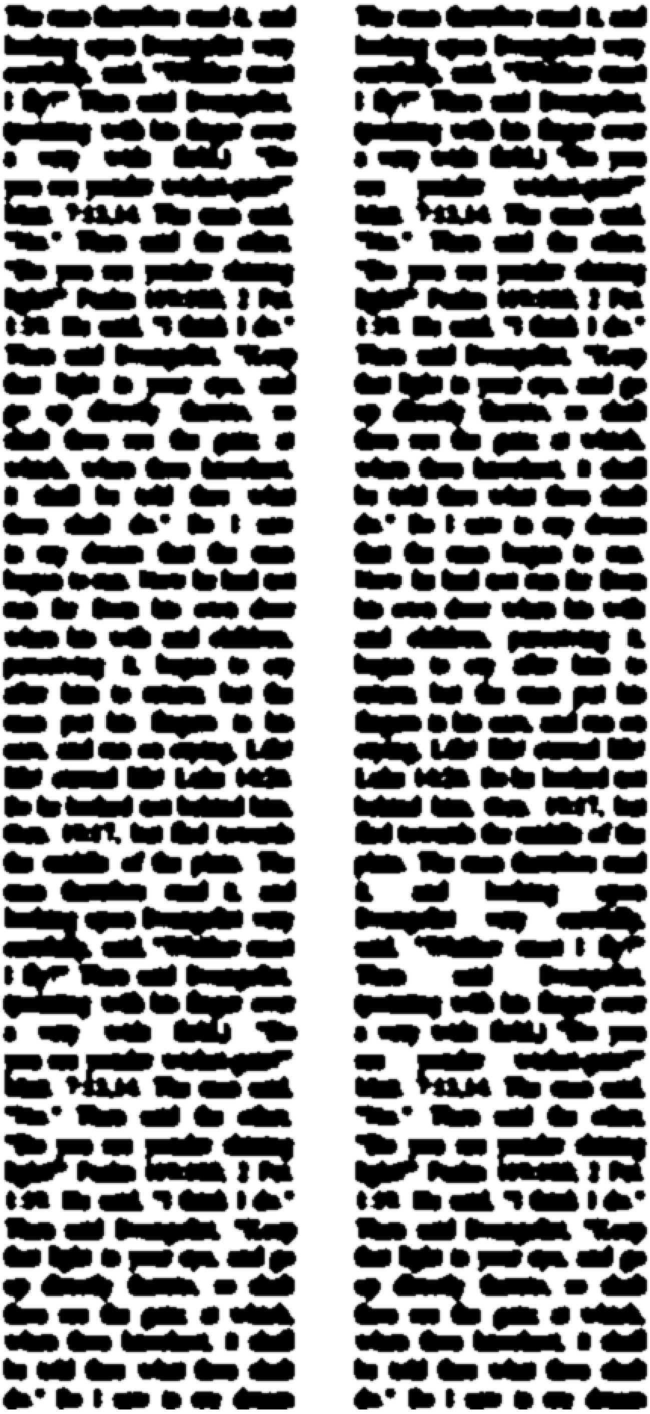


Figure 3: Knuth-Plass layout (left) versus a first-fit algorithm (right). The text has been greeked to draw attention to layout rather than content. Note that Knuth-Plass results in looser spacing of certain lines where it helps avoid extremely loosely-set lines ahead. Even without using hyphenation (this implementation of Knuth-Plass does not include a hyphenation algorithm) the differences are pronounced.

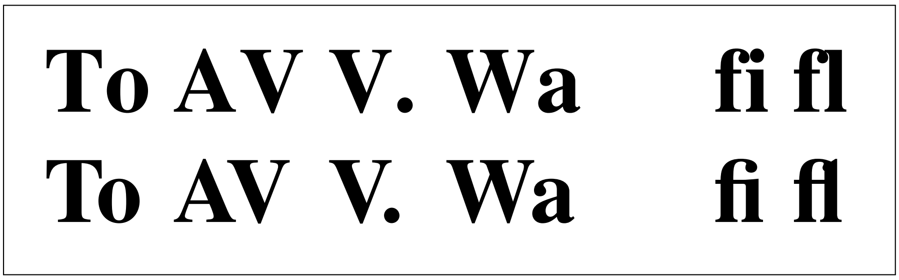


Figure 4: Examples of various letter-pairs and their kerned (left) or ligature (right) equivalents, as typeset by pdfLaTeX. Some further examples of fi ligatures (or not!) can be seen in Figure 1 on page 4.

```
KPX A y -92
KPX A w -92
KPX A v -74
KPX A u 0
KPX A quoteright -111
KPX A quotedblright 0
KPX A p 0
KPX A Y -105
KPX A W -90
KPX A V -135
KPX A U -55
KPX A T -111
KPX A Q -55
KPX A O -55
KPX A G -40
KPX A C -40
```

Listing 1: An excerpt from a kern table for Times Roman, showing kern pairs beginning with `A` only. This is taken from an AFM (Adobe Font Metrics) file, where the units are (according to the specification[Ado98]) *"equal to 1/1000 of the scale factor (point size) of the font being used"*.

involves altering the spacing between certain glyph pairs in order to give the appearance of more consistent letter spacing, and ligatures are single-glyph replacements for two or more single glyphs that may otherwise have had clashing components. Some examples of these are shown in Figure 4.

Kerning requires a table of kern-pairs, specific to each font; values from this table must then be looked up for every pair of adjacent glyphs in the document. An example of a kern table is shown in Listing 1.

Ligatures may or may not need to be inserted: if the component characters of the ligature lie over a potential hyphenation point, it cannot be decided whether to replace them with the ligature until it is known whether the hyphenation point needs to be used.

TEX handles kerning and insertion of ligatures automatically, but there are still further typographical tweaks that its default typesetting algorithm does not use.

More advanced methods than simply stretching or shrinking the word spacing do exist, however. Robert Bringhurst, in *The Elements of Typographic Style*,[Bri08] suggests that in addition to altering word spacing, subtle changes to inter-character spacing (also known as tracking) and to individual glyph widths (in the range of ± 3%) can produce more typographically and aesthetically pleasing results.

pdfLATEX, *used to typeset this document,* does *tweak tracking and glyph widths when justifying text*

1.3 PARADIGMS OF DOCUMENT REPRESENTATION

Computer representations of documents can be classified into two distinct paradigms:

- Documents stored in *fixed formats*, such as Portable Document Format (PDF), PostScript and SVG, are designed to be the direct analogues of printed pages.
- Documents stored in *flowable formats*, such as the Hypertext Markup Language (HTML) and EPUB, have no fixed presentation associated with them, thus their layouts must be computed each time they are displayed.

Currently there is no middle ground — a document may either be fully rendered to a fixed layout, or completely unrendered, to be laid out at the mercy of a display device's decisions.

1.3.1 *Fixed Formats*

The only fixed document format commonly used for ebooks (or indeed commonly used at all) is PDF,[Ado01] which was originally designed as a way of faithfully reproducing documents both on screen and in print [War91]. For this reason, it is almost entirely presentation-oriented and will not necessarily include any metadata pertaining to the semantic structure of a given document.

The archetypal PDF file consists solely of drawing operators that describe the document pages. There is no compulsion for these drawing operators to render the page in an order that might be considered sensible by a human reader. For example, if a PDF generator program decided to render every character on a page in alphabetical order, or radially outwards from the centre, the resulting file would still be semantically valid, and the result would be imperceptible to the reader.

This lack of imposed semantic structure makes it difficult to infer the best way to "unpick" PDF files to allow their content to be reflowed into a new layout [LB95, BB05]. For example, it is not easy to decide programatically whether a line break between adjacent lines of text is explicitly intended to be there (for example, the end of a paragraph) or if it is an artefact of the document layout. As an example, the Adobe Reader app for Android cheerfully offers to reflow PDF pages should you require it, but usually results in what can be seen in Figure 5.

It would be inaccurate to state that PDF files *cannot* represent the semantic structure of their content — indeed as early as 1999, PDF 1.3 introduced *logical structure* facilities,[Ado01] adding an optional *structure tree* to the PDF specification, and *tagged* PDF, introduced in PDF 1.4 in 2001, provides various extensions to this. PDF documents that actually make good use of these facilities are few and far between, even a decade after their introduction.

1.3.2 *Flowable Formats*

The two most common flowable ebook formats are EPUB and Mobipocket, both of which are largely based on HTML [IDP11]. HTML was chosen not only because of its inherent support for reflow, but also because it allows document content to be semantically marked up into paragraphs, various levels of headers, and so on. At the time, this was an enormous improvement over ebooks stored as plain text, which consequently had neither formatting nor semantic structure.

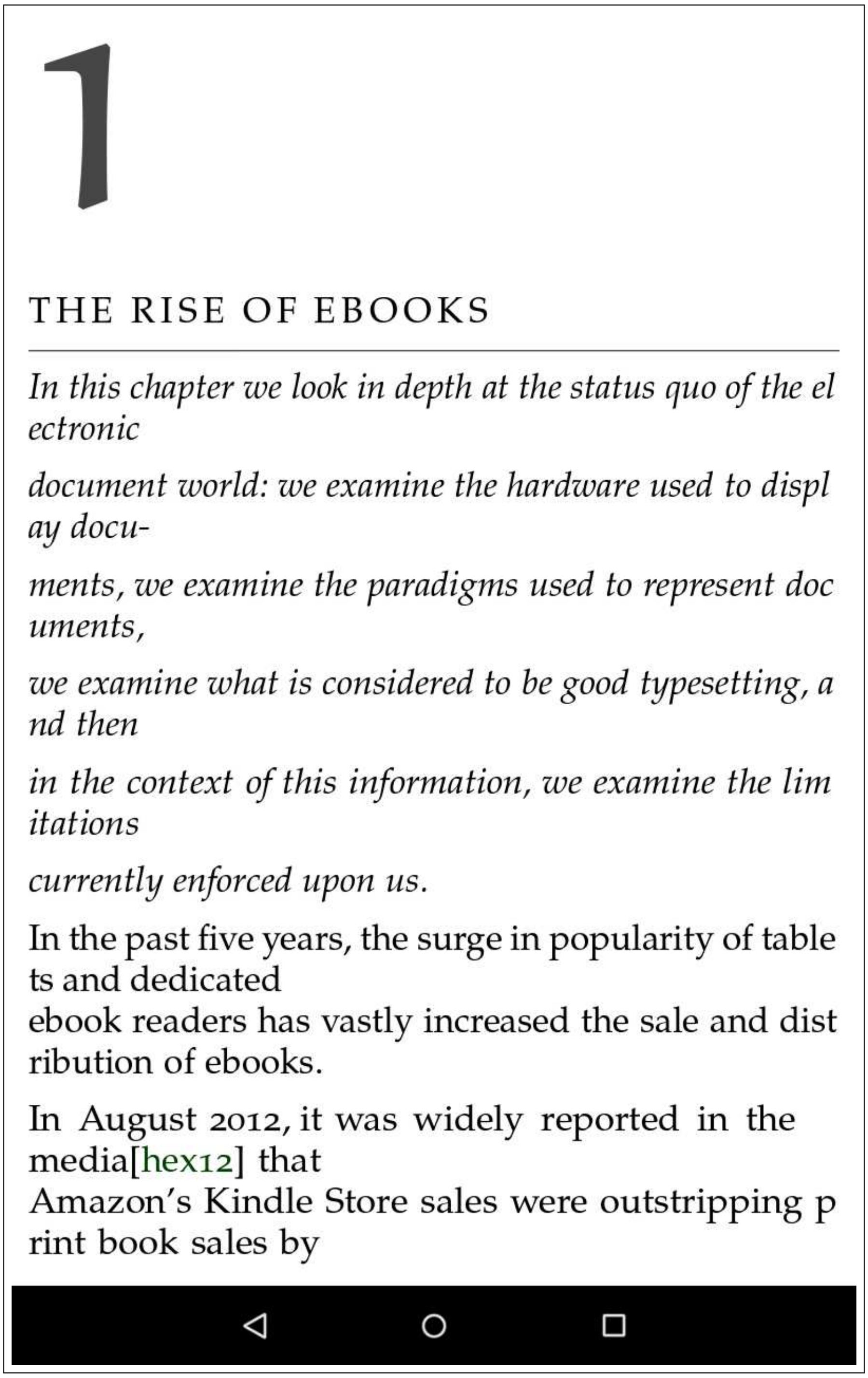
Amazon's proprietary Kindle format is derived from Mobipocket; PDF and EPUB are open standards

Whilst the use of these HTML-like formats allows the semantic structure of documents to be very well defined, in general their presentation can only be specified in a very loose manner. On an ebook reader (or in ebook reading software) the user is often presented with a choice of typefaces and point sizes, which gives the e-reader software some scope for rendering the document in arbitrary ways.

Since a document stored in a flowable format does not have any concrete presentation associated with it, each time the document is displayed, its layout must be recomputed. For an ebook reader to maximise its battery life, this computation must be as simple as possible. As a consequence, the algorithm used must not be too complex, since the more CPU cycles spent executing it, the less time the CPU can spend idle, and thus the greater the drain on the device's battery [PBB11].

1.4 LIMITATIONS OF CURRENT FORMATS

The design paradigm of PDF, conceived in the early 1990s,[War91] was to form a perfect analogue of the printed page, which would be exactly reproducible regardless of the system on which the file was rendered. For this reason, it is possible to embed fonts within PDF files, to ensure faithful reproduction on any system, regardless of which fonts are actually installed. In general, a well typeset PDF file looks good wherever it is displayed, but, stemming from the "digital sheet of paper" paradigm, page sizes in a PDF document are necessarily



1

THE RISE OF EBOOKS

In this chapter we look in depth at the status quo of the el
ectronic

document world: we examine the hardware used to displ
ay docu-

ments, we examine the paradigms used to represent doc
uments,

we examine what is considered to be good typesetting, a
nd then

in the context of this information, we examine the lim
itations

currently enforced upon us.

In the past five years, the surge in popularity of table
ts and dedicated
ebook readers has vastly increased the sale and dist
ribution of ebooks.

In August 2012, it was widely reported in the
media[hex12] that
Amazon's Kindle Store sales were outstripping p
rint book sales by

Figure 5: Adobe's Reader app for Android includes an option that allows PDF files to be reflowed, but the lack of semantic structure within most PDF files limits the process to using only what can be inferred from the original layout. This figure shows the start of this chapter reflowed by the app — see page 3.

fixed at creation-time. An overwhelming majority of PDF documents are rendered for US letter or A4 size paper. This is fine if the document is to be printed and read. On a reasonably large screen, the document remains perfectly readable, and on a 10″ netbook or tablet screen it may provide an acceptable reading experience. Anything much smaller (notably mobile phones, and ebook readers) requires a combination of zooming and panning in order to read the document.

Documents may, of course, be rendered to a smaller page size, but the problem still remains — it is unlikely that any one page size will be suited to *all* reading platforms. Most ebook readers, using their native (i.e. non-PDF) formats, allow text to be resized according to user preference. Indeed, it seems unnecessarily restrictive to force one size of type upon the user. While physical books suffer from this affliction, ebooks need not. Selling ebooks separately in standard and large-print versions seems perverse when, for virtually no difference in cost to the publisher/distributor, both can be included in one file.

Systems to reflow the content of PDF documents have been devised, [LB95, Mar13] but as noted in Section 1.3.1, this is often extremely difficult to accomplish satisfactorily. Even when the logical order of page components can be identified correctly, the benefit of any precomputed high-quality typesetting is lost if the text itself must be re-typeset.

EPUB and Mobipocket, both based on HTML, provide a higher level of abstraction for documents, whereby the logical structure of their content is still present. They allow many rendering decisions to be made at view-time, such as choice of typeface and font size. Line breaks and page breaks are then calculated and inserted as necessary, in order to wrap the text to fit the screen, and to paginate the content.

The rendering engines of ebook readers use simplistic reflow algorithms — but necessarily so. One of the major bottlenecks in today’s portable electronic devices is their battery life: battery capacity has not improved at anywhere near the same rate as other facets of mobile computing. Manufacturers of ebook readers may claim their products have batteries that can last for weeks, but this is principally due to the many typographical corners that they cut when laying out flowable content. As noted in Section 1.3.2, were these devices to use more complex layout algorithms, which can produce far higher quality typeset output, any savings made by using a low-power electronic paper screen would quickly be lost. Furthermore, the more time that is spent formatting the output, the longer the delay between page turns on the device, given that each subsequent page is only rendered when a page turn is requested. This time delay could be shortened by precomputing the layout of subsequent pages between page turns, but this would not solve the battery-drain problem.

EPUB allows fonts to be embedded, but Mobipocket does not. Mobipocket files (and by extension, Kindle files) are therefore restricted to be rendered in a typeface local to (and often chosen by) the reader software. The Kindle, as an example, provides the user with a choice of "regular", "condensed", or "sans-serif" for the main body text of its documents. There are bold and italic variants of these, which are applied according to formatting instructions within the documents themselves. Additionally, there is a typewriter-style font which document authors may choose to use in the same manner.

Amazon has begun to address this issue by developing a new format, KF8*, still Mobipocket-based, that allows more complex styling, in a manner comparable to* EPUB

The Mobipocket specification supports a very limited subset of HTML and CSS, which makes it virtually impossible to achieve complex layouts such as those involving arbitrary indentation or font size changes. Figures 6 and 7 overleaf demonstrate the well known "Mouse's Tale" from Lewis Carroll's *Alice in Wonderland*, and the limitations of various formats.

EPUB is a little more flexible, since it supports a more comprehensive range of XHTML and CSS, has support for SVG, and allows for arbitrarily complex styling. EPUB files are still entirely reliant on the rendering engine of the display device correctly displaying their content, as they have no concrete layout associated with them.

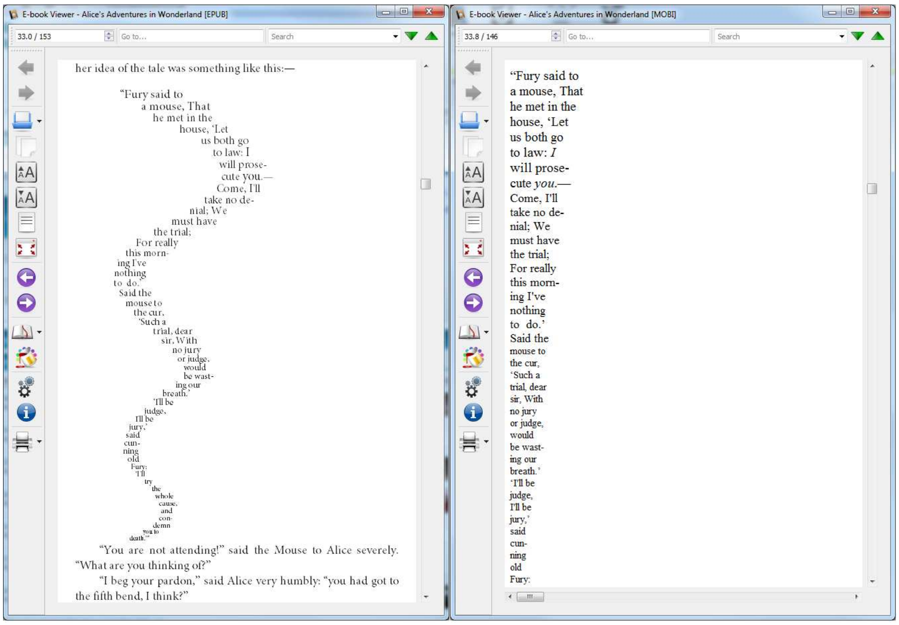
1.5 SUMMARY

We have so far seen that electronic representations of documents have layouts that are either fully fixed, or fully flowable. Currently there is no middle ground. Documents with fixed layouts may be of arbitrarily high typographic quality, since their layout is fully computed when they are created. Documents with flowable layouts are not provided with any guarantee that their content will be laid out with any semblance of typographic quality. In any case, to compute a high-quality layout in real-time is difficult, especially on a low-powered portable device such as an ebook reader.

We have seen that screen technologies for ebook readers have been evolving to become better and better, allowing documents to be displayed in a quality that rivals physical, printed pages. Document representation paradigms have not caught up. They are based on technologies that have been repurposed to be used in ways they were never designed. Fixed layout representations were designed for display on paper. Flowable layout representations were designed for display on such low-resolution screens and underpowered devices that quality typography would have been nothing but a pipe dream.

1.6 CONTRIBUTIONS OF THIS THESIS

It is clearly time for a new document representation paradigm to be devised, in order to catch up with contemporary document display technologies—it is time for the introduction of a more *malleable* document format. There has been little research into this area in the past.


E-book Viewer - Alice's Adventures in Wonderland [EPUB]
33.0 / 153
Go to...
Search
E-book Viewer - Alice's Adventures in Wonderland [MOBI]
33.8 / 146
Go to...
Search

her idea of the tale was something like this:—

“Fury said to
a mouse, That
he met in the
house, ‘Let
us both go
to law: I
will prose-
cute you.—
Come, I'll
take no de-
nial; We
must have
the trial;
For really
this morn-
ing I've
nothing
to do.’
Said the
mouse to
the cur,
‘Such a
trial, dear
sir, With
no jury
or judge,
would
be wast-
ing our
breath.’
‘I'll be
judge,
I'll be
jury,’
said
cun-
ning
old
Fury:
‘I'll
try
the
whole
cause,
and
con-
demn
you to
death.’”

“You are not attending!” said the Mouse to Alice severely. “What are you thinking of?”

“I beg your pardon,” said Alice very humbly: “you had got to the fifth bend, I think?”

“Fury said to
a mouse, That
he met in the
house, ‘Let
us both go
to law: *I*
will prose-
cute *you.*—
Come, I'll
take no de-
nial; We
must have
the trial;
For really
this morn-
ing I've
nothing
to do.’
Said the
mouse to
the cur,
‘Such a
trial, dear
sir, With
no jury
or judge,
would
be wast-
ing our
breath.’
‘I'll be
judge,
I'll be
jury,’
said
cun-
ning
old
Fury:

Figure 6: On the left is an EPUB version of Alice in Wonderland, displayed in Calibre (an open source desktop ebook viewer). On the right is the same file, converted to Mobipocket, also displayed in Calibre. Note that in addition to the indentation being lost, the (embedded) font from the EPUB is no longer present in the Mobipocket file.

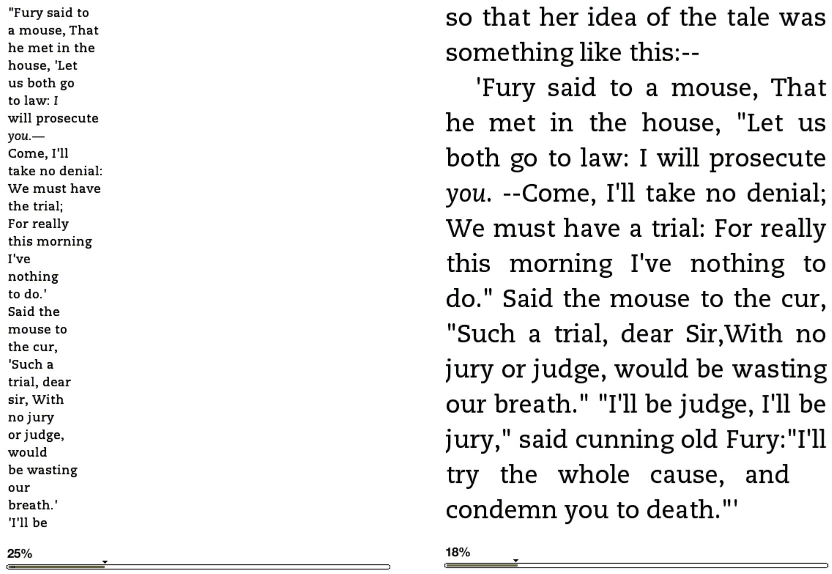
Much effort has been put into high-quality typesetting for documents with fixed layouts. Much effort has been put into providing styling for documents with flowable layouts—notably via css—but virtually no consideration has been given to producing well-typeset output on the fly, particularly with the minimum required computation.

1.6.1 *Scope*

In this thesis, such a novel document representation paradigm, geared towards use on portable devices, is proposed and implemented. The desired malleability is afforded to documents by precomputing key parts of the typesetting process, which allows their content to be gently coaxed to fit any page size, without needing to make any compromises in the quality of their typography.

It is shown how this system can be extended from supporting straightforward sequential content to allowing “floating” blocks of varying sizes whose absolute placement is unimportant. With this addition, the system becomes capable of rendering its documents into layouts reminiscent of those of newspapers, magazines, and academic journals.

Despite focusing mainly on reducing computational complexity at view-time, attention also must be paid to the filesize of the resultant documents, to ensure that storage space is not being wasted unne-



"Fury said to
a mouse, That
he met in the
house, 'Let
us both go
to law: *I*
will prosecute
you.—
Come, I'll
take no denial:
We must have
the trial;
For really
this morning
I've
nothing
to do.'
Said the
mouse to
the cur,
'Such a
trial, dear
sir, With
no jury
or judge,
would
be wasting
our
breath.'
'I'll be

25%

so that her idea of the tale was something like this:--

'Fury said to a mouse, That he met in the house, "Let us both go to law: I will prosecute *you.* --Come, I'll take no denial; We must have a trial: For really this morning I've nothing to do." Said the mouse to the cur, "Such a trial, dear Sir,With no jury or judge, would be wasting our breath." "I'll be judge, I'll be jury," said cunning old Fury:"I'll try the whole cause, and condemn you to death."'

18%

Figure 7: On the left is the Mobipocket version of Alice in Wonderland (from Figure 6) displayed on the Kindle. Note that the sizing instructions appear to have been ignored. On the right is the free version of Alice in Wonderland from the Kindle store, displayed on the Kindle. Note that no attempt has been made to render the poem in a "tail" shape.

cessarily. To address this issue, a computationally-light compression scheme is developed.

1.6.2 *Limits*

To ensure maximum portability for a single codebase, the final implementation is designed to run within any modern JavaScript enabled web browser, and can therefore be used on virtually any modern web-enabled device, including desktops, mobiles, and tablets. However, for many devices, JavaScript running within a web browser will not be most efficient way to display text and images. Some other, more lightweight, native implementation tailored to the specific device will almost certainly provide a more computationally- and battery-efficient solution. Within this thesis, no attempt will be made to implement such a program, though all algorithms described herein are applicable to other languages and concrete implementations.

Other aspects of the proposed system that will only partially be explored are those described in Sections 2.5 and 2.6. These areas cover the more qualitative aspects: respectively, how to choose the range of galley renderings at document compile-time, and how to choose the "best" single galley rendering to display at document view-time.

These two areas contain so many variables that fine-tuning them is almost a research project in itself. On this basis, both the initial choice of range of renderings, and the algorithm to choose which rendering to show at view-time, were developed by trial and error by the author, in conjunction with a user study that provided feedback from a wider range of participants.

Part II

IMPLEMENTATION

2

THE MALLEABLE DOCUMENT

In Chapter 1 we looked in considerable detail at precisely which elements of a typeset document must be considered computation-heavy (and thus should be avoided if possible at view-time).

In this chapter we delve briefly into typesetting's mechanical history, and examine a technique used in the past for newspaper layouts that we adapt to our purposes today. We then discuss which portions of the typesetting process can be pre-computed and which cannot, and analyse where shortcuts can be taken, and their effects on the final document.

We then look at the development of a first prototype implementation, made up of a plugin for Adobe Acrobat, in conjunction with a program that produces PDF *files with extra embedded metadata. These special* PDF *files can then be viewed and reflowed within Adobe Acrobat when the plugin is active. This actual implementation is not used beyond this chapter, as its reliance on Adobe Acrobat stymies its portability, though the underlying ideas are transferred to a new, more portable implementation, as we will see in subsequent chapters.*

The research in this chapter was previously published in [PBB11]

The ultimate goal for a "malleable" document format is a system that allows the majority of typographical decisions (and therefore hard computation) to be carried out at document compile-time, but leaves enough flexibility that at view-time the content can be rendered to fit any screen size. The requirement for "malleability" in pre-typeset text is not a new one, and we discuss this next.

2.1 HISTORICAL INTERLUDE

The invention of movable type in China in the 11th Century, and independently in Europe in the 15th Century, led to an enormous increase in the availability of printed material. In its various incarnations, movable type formed an extremely important part of the newspaper industry, from its advent in the 17th century, until digitisation in the mid-1980s.

The inherently volatile nature of newspaper layout (caused, for example, by important stories breaking shortly before going to press) coupled with the expense and time-consuming nature of physical typesetting, led to the development of the familiar columnar appearance of the newspaper that is prevalent worldwide.

2.1.1 *The use of Galleys in Typesetting*

In a traditional newspaper layout, each page is divided into columns that are of equal width, or measure. All text that is to appear in the newspaper is typeset to fit this measure (or integer multiples thereof, for example in the case of headlines) allowing articles to be slotted anywhere into the final layout of the newspaper, simply by breaking the article text between lines where necessary to span across columns and/or pages. This means that wherever an article is placed, it never requires retypesetting as long as its content remains unchanged — an advantage only available when all text is rendered to fit into columns of the same width.

The metal trays that are used to contain typeset lines of physical type are known as galleys. Newspapers use reasonably narrow galleys; paperback books tend to use wider galleys, and hardback books wider still. Narrower galleys offer the advantage that less space is wasted if the final line in a paragraph does not span the full width: this is more important in newspaper layout than in most other typesetting situations, since space is at a premium. Wider galleys aid readability, up to a certain point, after which it becomes difficult for the eyes to keep track between lines [Bri08, BMM+09, Voo11].

A straw poll of various items of print within arms' reach suggests that newspapers use approximately 2″ galleys, paperback books around 4″, and hardback books around 5″. This thesis uses a galley width of $4\frac{2}{3}$″

2.2 GALLEYS AS A REFLOW TOOL

Since each individual line never changes, typesetting text into physical galleys is directly analogous to precomputing many of the "hard" parts of typesetting. In particular, all hyphenation, line breaking, justification, kerning, glyph substitution, and in fact all horizontal layout, is "compiled out" as the galley is created.

When the galleys are later fitted to a physical page, there is no requirement to re-typeset any of the horizontal layout: only vertical layout problems remain, such as attempting to avoid widowed and orphaned lines (where the last/first lines of a paragraph appear first/ last in a column) and choosing optimal placements of figures and other floating bodies. What remains is essentially the problem described by Michael F. Plass in his Ph.D. thesis *Optimal Pagination Techniques for Automatic Typesetting Systems*,[Pla81] and by Donald E. Knuth in Chapter 15 of The TeXbook, *How TeX Makes Lines into Pages* [Knu84].

As mentioned in the previous section, setting the text of a document into a galley provides it with some limited flowability. Specifically, the document can be paginated to fit pages of any size at least as wide as the galley, and of arbitrary height. Wider pages may be able to accommodate multiple columns, though if care is not taken when choosing the page size, the likelihood of noticeable extra horizontal whitespace is increased.

Figure 8 shows how the extra horizontal whitespace on a page varies with page width. The peaks occur just before the point where an extra column can be added, and the amount of extra whitespace that is required drops to a minimum. The blue line shows the extra whitespace divided by the number of columns that fit on the page, which gives a more useful metric to work with: if we physically divide the extra whitespace and insert it between the columns to increase their spacing (as opposed to leaving it on the right- or left-hand margins) then the wider the page, the less detriment is caused by the extra fraction of galley width.

This process of setting text into galleys and reflowing can easily be simulated programatically, and as such is a promising way to precompute many of the more complex parts of the typesetting process, without sacrificing flowability.

2.3 MULTIPLE GALLEY RENDERINGS

The problem of extra whitespace can be overcome in several ways. Firstly, and most simply, the precomputed galley could be scaled up or down, effectively simulating a change in the point size of the font. This is an obvious side-effect and is probably undesirable, unless a change in point size has explicitly been requested, and especially if the size change is particularly noticeable.

A second way in which columns can be better fitted to the page width is to typeset the source document into a range of galleys, each of different measure. When the document is to be rendered at view-time, the most appropriate measure (according to some metric) can be selected for display. One very simple (and therefore fast) metric is to select whichever galley rendering would minimise the additional whitespace.

By overlaying the Figure 8-like graphs for each galley, we are able to obtain a graph like Figure 9, which features all available galleys. If we use our simple metric of minimum whitespace, we can simply select whichever galley requires the smallest amount of extra whitespace for a given page width. Further consideration is given to this process in Section 2.6.

2.4 A SIMPLE IMPLEMENTATION

The algorithm described above was prototyped using existing tools from the University of Nottingham Document Engineering Laboratory: specifically, the Component Object Graphic (COG) model[BBH03] for creating and managing modularised PDF documents. This was chosen specifically to avoid the need to write a typesetter or layout engine from scratch; typesetting is performed by the *troff* suite, and the display of the final layout by Adobe Acrobat.

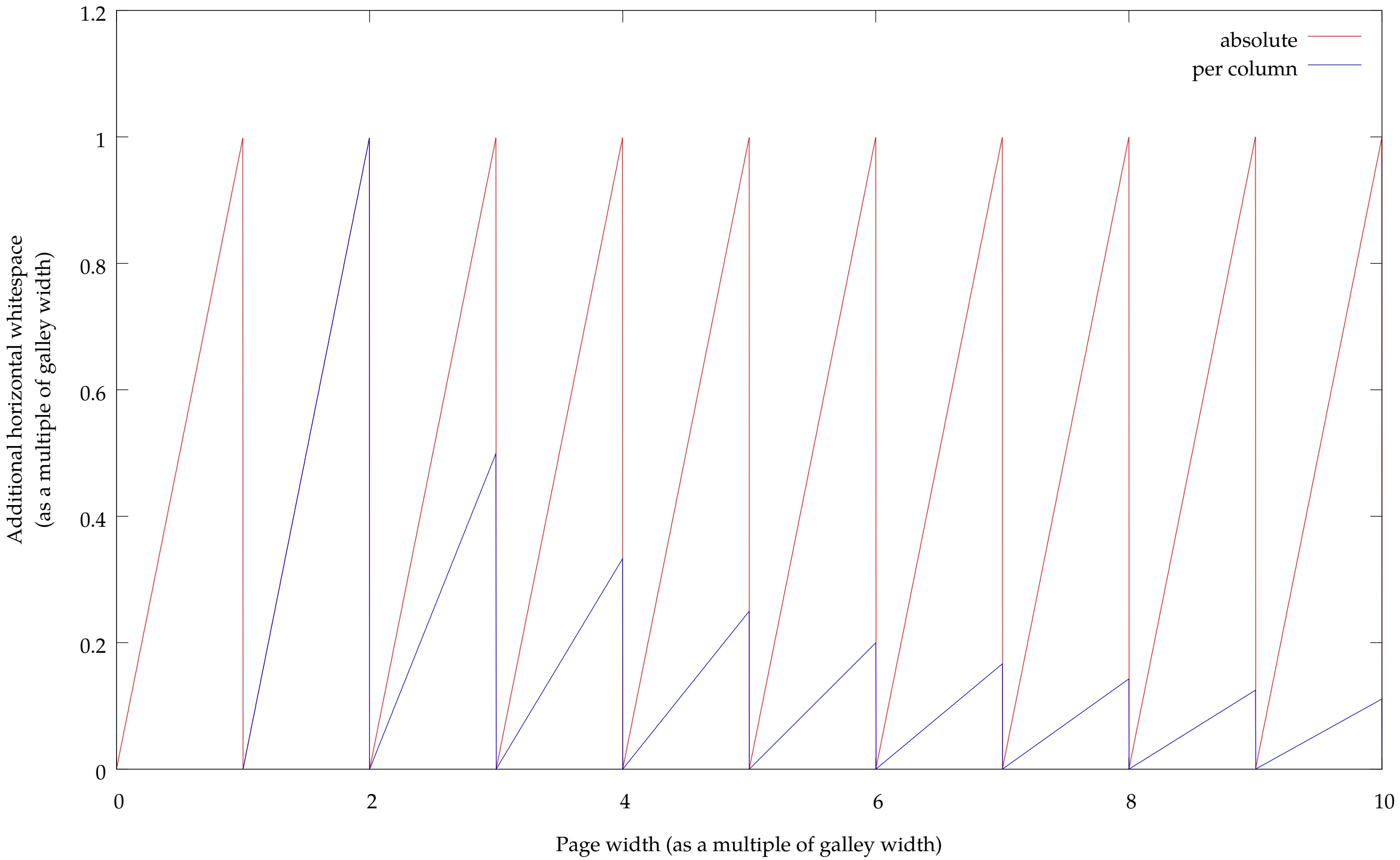


Figure 8: As more columns fit on a page, the extra whitespace required per column (shown in blue on the graph) decreases. The peaks occur just before the point where an extra column can be added, and the amount of extra whitespace that is required drops to a minimum.

2.4.1 *The COG Model*

The COG model was developed to enable the reuse of semantic components within PDF documents, by breaking the traditional graphically-monolithic PDF page into a series of distinct, encapsulated graphical blocks, termed COGs. Initial work on what later became the COG model was conducted in the mid-1990s,[SB95] and further developed throughout the 2000s [BBH03, Bag04, BB05, MBB05, Bag06, BBO07]. The COG model, as described in the previous citations, does not account for any relationship between individual COGs — it was simply designed as a method with which document components could be easily reused, reordered, or extracted. The COGs generated are largely at the granularity of a paragraph, but there is still no directive to image them onto the page in any particular order (for example in reading order).

In order to implement a galley-based design, it was necessary to change the granularity at which the COGs were produced, such that each line of text is represented by a separate COG. However, it was also important to maintain the semantic structure of documents. This is crucial for the process of layout at view-time: not only must the logical order of the document content be preserved, but also the relationship between each component of the content, such as which line belongs to which paragraph, whether a certain item is floatable, and so on.

The COG model takes advantage of the fact that the PDF specification [Ado01] allows page content to be described by an array of streams of imaging operators, rather than the more commonly encountered single, monolithic stream. Unfortunately, this array is one-dimensional, meaning that whilst it can state an explicit ordering of components, it cannot be used, say, to group lines into paragraphs, or paragraphs into sections.

Since the PDF specification allows essentially arbitrary insertion of data structures into a document, this flexibility was used to embed the document's structure as a tree, in parallel to the PDF's content array. The term *Galley Structure Tree* will henceforth be used to refer to this data structure. (At this point, it is important to make the distinction between the term *Galley Structure Tree* and the unrelated *structure tree* that is defined in the PDF specification and mentioned in Section 1.3.1.)

According to the specification, PDF readers that encounter unknown data within a PDF file that they do not recognise should simply ignore it

An example of a simple Galley Structure Tree is shown in Figure 10. At the level of its leaves, this tree contains pointers to the COGs which make up the content of the document. In the simplest case, where the document contains only one rendering (and thus the paragraph-level items have only one child) the COGs pointed at by the leaves can simply be rendered in order, adding vertical space as appropriate.

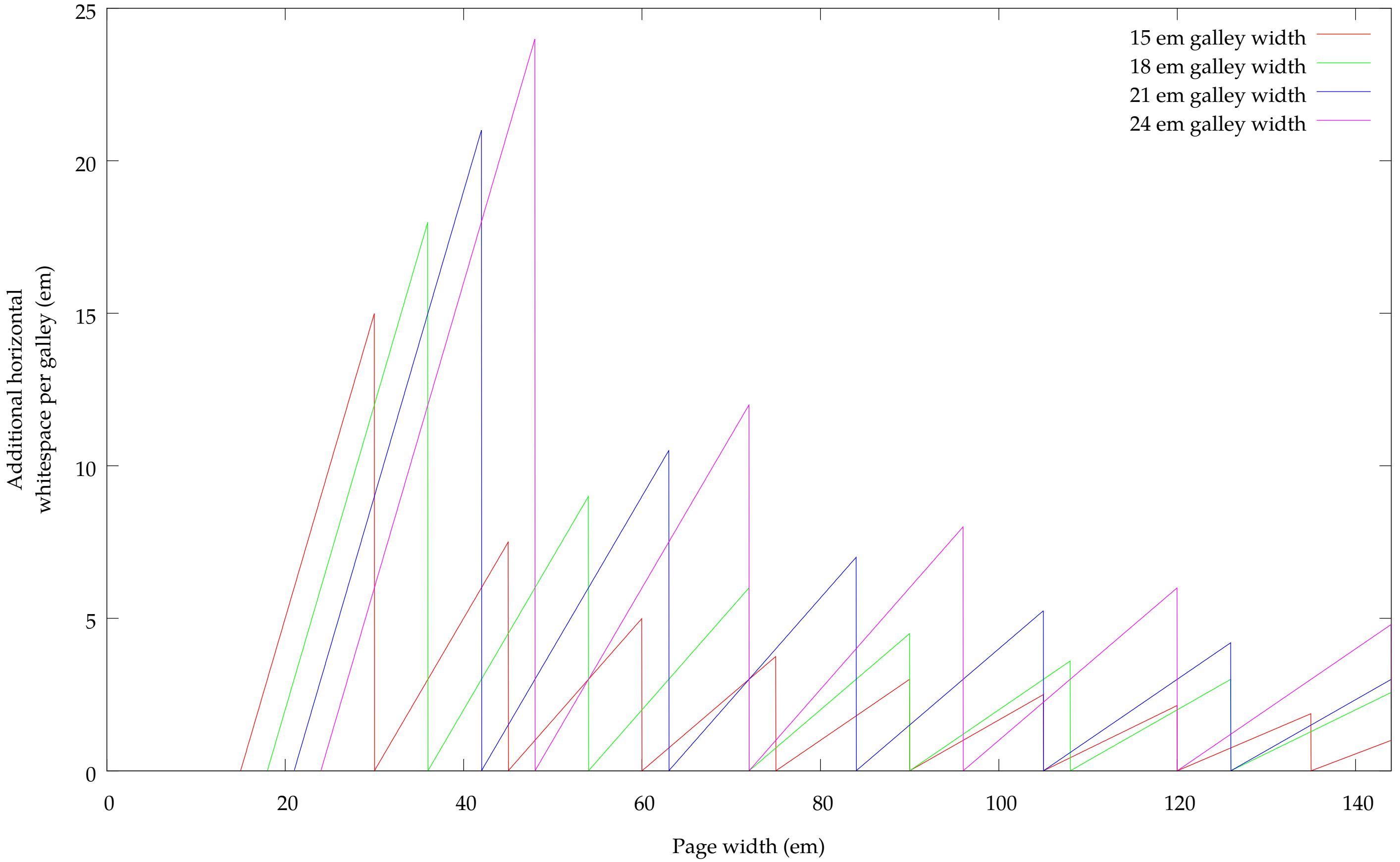


Figure 9: Overlaying the sawtooth graphs for several galleys of differing widths allows us to easily identify the galley that minimises extra added whitespace: at any point on the x-axis, the line with the smallest value on the y-axis corresponds to the galley that will most tightly fit the space. The choice of galley widths used here is essentially arbitrary, to demonstrate the effect of overlaying these graphs. Section 2.5 goes into detail about how to choose the widths of the included renderings.

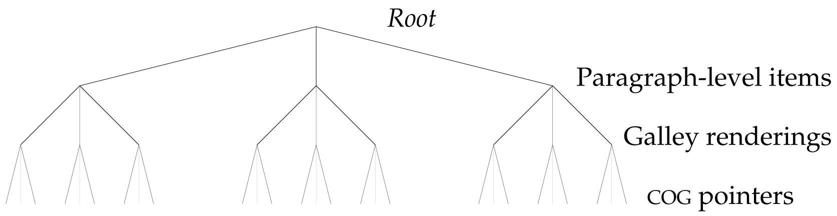


Figure 10: A simple Galley Structure Tree. The first level below the root represents all paragraph-level items: headings, paragraphs, figures etc. These items have one child for each galley rendering of the document. These in turn have one child for each COG comprising their content — in the case of a paragraph or heading its lines; in the case of a figure, the figure itself and any associated caption.

2.4.2 *The Source Document*

Since the majority of available tools for producing COGged PDFs rely on the typesetting package *ditroff*,[Ker82] it was decided to use this as the basis for the source document. *Ditroff* is particularly amenable to many of the features required here — it is quite happy to have its page size set to large values — one sample document used a page length of 2000 inches (approximately 50 metres) with no complaints from *ditroff*. (This is important because it avoids *ditroff* performing any pagination, which would otherwise cause COG resources to be spread across multiple pages in the resultant PDF file, which in turn would cause difficulties accessing these resources from other pages.) An example of a source document is shown in Listings 2 and 3 overleaf.

2.4.3 *pdfdit*

The output from *ditroff*, Ditroff Intermediate Code, is very expressive, and, unlike TEX's equivalent DVI, contains enough information that post-processors are easily able to locate the start and end of lines and paragraphs within the document. This meant that only minimal changes were needed to the *pdfdit* package,[BBH03] which was developed to produce COG-PDF from *troff* documents.

The first modification necessary was to alter the granularity of the output COGs, to produce them at line level, rather than at paragraph level. Secondly, some method of generating the requisite tree representing the document structure was required. Fortunately, since *pdfdit* could already detect paragraphs, this code was adapted to create a new node in the Galley Structure Tree. Each subsequent line-level COG produced can then be added as a child of this node.

```
.nr HM 0.5i
.nr FM 0.5i
.ds CH
.\" Overwrites the Centre Header (suppresses page number)
.pl 2000i
.\" Make the page quite long to avoid troff doing any pagination
.nr PO 0
.\" set Page Offset (ie left margin) to zero
.ps 11
.vs 13
.nr PS 11
.nr VS 13
.\" Set point size to 11 and vertical spacing (leading) to 13
.\" Below, alter value of LL (line length) and include document
.\" content with the .so macro
.nr LL 1i
\X’cWidth:72’
.\" Line Length (ie galley width)
.so contents.inc
.nr LL 1.5i
\X’cWidth:108’
.so contents.inc
.nr LL 2i
\X’cWidth:144’
.so contents.inc
.nr LL 2.5i
\X’cWidth:180’
.so contents.inc
.nr LL 3i
\X’cWidth:216’
.so contents.inc
.nr LL 3.5i
\X’cWidth:252’
.so contents.inc
.nr LL 4i
\X’cWidth:288’
.so contents.inc
```

Listing 2: A sample troff source document. The actual document text is in a file named `contents.inc` , and is imported multiple times with the `.so` macro. After each import, the current line length is changed (using, for example, `.nr LL 1.5i` to set the Line Length register to 1.5 inches). The `\X` commands are used to pass arbitrary data through the typesetter, and into the resultant ditroff intermediate code for later use. In this case, it is used to pass the column width (hence `cWidth`) in points, so that this data can later be embedded within the final PDF file.

```
.PP
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras vel
enim vitae mauris vestibulum egestas. Suspendisse potenti.
Pellentesque leo nunc, lobortis vitae gravida vel, congue at
nulla. Praesent a placerat mauris. Praesent sed erat ac dui
tincidunt consectetur vel nec leo. In velit odio, congue non
eleifend at, accumsan eu diam. Suspendisse dignissim, quam quis
euismod laoreet, est leo euismod lectus, sed consequat leo nunc
in ante. Duis risus tellus, suscipit ut fermentum et, ornare non
lorem. Morbi nibh elit, dignissim ullamcorper posuere at, lacinia
condimentum odio. Fusce vitae metus mi. Pellentesque scelerisque
fermentum magna a dictum. Mauris ut ante mauris, ac viverra
felis. Praesent ut elit ut purus malesuada suscipit. Fusce mollis
eros ac lectus suscipit gravida. Pellentesque vel nisl nec eros
convallis luctus nec eu quam. Aliquam tincidunt ultrices blandit.
.PP
Vestibulum lorem felis, consectetur ornare vehicula ac, cursus
tincidunt nisl. Aliquam in enim nisi, quis hendrerit est. Nullam
pretium congue sapien ac tincidunt. Suspendisse suscipit felis
eget nibh luctus sit amet imperdiet ligula venenatis. Vestibulum
eu dui nulla. Vivamus interdum ullamcorper sapien eget dapibus.
Proin sed dictum arcu. Curabitur velit justo, fringilla non
sodales laoreet, dignissim nec nulla. Praesent convallis ipsum
quis dolor ultricies non sodales dui viverra. Phasellus nec nisi
at nisi bibendum aliquam vitae et arcu. Donec feugiat dolor ut
felis dapibus eget auctor enim mattis.
.PP
Curabitur eget eros neque, in pulvinar massa. Suspendisse ac
massa quis justo fringilla consectetur. Vivamus lacinia tincidunt
purus, sit amet ullamcorper neque imperdiet quis. Nulla at lectus
turpis, in semper augue. Donec eu rhoncus turpis. Maecenas lectus
lacus, porta et dictum eget, eleifend a nibh. Vestibulum pulvinar
pellentesque lectus, et tincidunt eros consequat sed. Duis risus
lorem, placerat et molestie ut, porta a mi. Fusce eu elit enim,
id consequat nibh. Cras elementum, odio a tristique rutrum, nibh
neque sodales lorem, eu feugiat ipsum leo a nunc. Quisque enim
felis, luctus dapibus iaculis ac, tempor vitae lacus. Nunc eu mi
quis lacus scelerisque tincidunt. Sed sed nulla dui. Suspendisse
porta imperdiet tortor vel ultricies. Donec sit amet ligula
velit. Nulla tempor, risus sit amet congue aliquam, est nulla
tincidunt lectus, scelerisque cursus lacus diam vel metus. Donec
eu elit dolor. Nullam id libero ac metus ornare iaculis id ut
lorem. Quisque iaculis justo nec nibh interdum vuPPutate.
```

Listing 3: A three-paragraph excerpt from a sample `contents.inc` file, as described in Listing 2. Each paragraph is preceded by a call of the `.PP` macro, which signifies to *troff* (via the `ms` macros) the start of a new paragraph.

```
%PDF-1.4
1 0 obj
<<
 /Type /Catalog
 /Pages 2 0 R
 /CogData 7472 0 R
 /ParagraphData 7473 0 R
>>
endobj
2 0 obj
<<
 /Type /Pages
 /Kids [7471 0 R  ]
 /Count 1
>>
endobj
```

Listing 4: An excerpt from the start of a malleable document pdf file. A reference to the `/ParagraphData` object (shown in Listing 5) has been added to the pdf's `/Catalog` object. The `/Pages` object is also shown — note that there is only ever one page in a malleable pdf.

The general algorithm for the processes alluded to in Sections 2.4.2 and 2.4.3 is detailed below:

1. The *troff* document's line length is set to a small value (2 inches or less) in order to produce a narrow column of text, and its page length to a very large value, to prevent pagination.

2. Following this, the actual document content is inserted several times, and the line length increased after each iteration, producing one document that effectively contains multiple galley-width renderings of the same content.

3. The source document is then processed with *ditroff* to generate the Ditroff Intermediate Code.

4. The Ditroff Intermediate Code is processed by *pdfdit* to produce a cog representation of the document, then it amalgamates the tree representations of each of the various galley widths into a Galley Structure Tree, in the form indicated in Figure 10.

5. Finally the pdf file is serialised, replete with cogs and Galley Structure Tree.

Various key excerpts from a malleable pdf file's internal structure are shown in Listings 4, 5, 6, and 7.

2.4.4 *Acrobat Plugin*

The decision to use Acrobat as an "ebook reader emulator" stemmed once again from the existing cog-based tools (see Section 2.4.2), as well as the extensive API and developer support available for Acrobat.

```
7473 0 obj
<<
 /Type /Multi-Width
 /Paragraphs
 [
    [72 108 144 180 216 252 288] % Width of each set of paragraphs, in
        points (ie 72*inch)
    [ % Paragraph 1
        [4 0 R 6 0 R 8 0 R 10 0 R 12 0 R 14 0 R 16 0 R 18 0 R 20 0 R 22 0
             R 24 0 R 26 0 R 28 0 R 30 0 R 32 0 R 34 0 R 36 0 R 38 0 R 4
            0 0 R 42 0 R 44 0 R 46 0 R 48 0 R 50 0 R 52 0 R 54 0 R 56 0
            R 58 0 R 60 0 R 62 0 R 64 0 R 66 0 R 68 0 R 70 0 R 72 0 R 74
             0 R 76 0 R 78 0 R 80 0 R 82 0 R 84 0 R 86 0 R 88 0 R 90 0 R
             92 0 R 94 0 R 96 0 R 98 0 R 100 0 R 102 0 R 104 0 R 106 0 R
             108 0 R 110 0 R 112 0 R 114 0 R 116 0 R 118 0 R 120 0 R 122
             0 R 124 0 R 126 0 R 128 0 R 130 0 R 132 0 R 134 0 R]
        [2303 0 R 2305 0 R 2307 0 R 2309 0 R 2311 0 R 2313 0 R 2315 0 R 2
            317 0 R 2319 0 R 2321 0 R 2323 0 R 2325 0 R 2327 0 R 2329 0
            R 2331 0 R 2333 0 R 2335 0 R 2337 0 R 2339 0 R 2341 0 R 2343
             0 R 2345 0 R 2347 0 R 2349 0 R 2351 0 R 2353 0 R 2355 0 R 2
            357 0 R 2359 0 R 2361 0 R 2363 0 R 2365 0 R 2367 0 R 2369 0
            R 2371 0 R 2373 0 R 2375 0 R 2377 0 R 2379 0 R 2381 0 R 2383
             0 R]
        [3755 0 R 3757 0 R 3759 0 R 3761 0 R 3763 0 R 3765 0 R 3767 0 R 3
            769 0 R 3771 0 R 3773 0 R 3775 0 R 3777 0 R 3779 0 R 3781 0
            R 3783 0 R 3785 0 R 3787 0 R 3789 0 R 3791 0 R 3793 0 R 3795
             0 R 3797 0 R 3799 0 R 3801 0 R 3803 0 R 3805 0 R 3807 0 R 3
            809 0 R 3811 0 R 3813 0 R]
        [4821 0 R 4823 0 R 4825 0 R 4827 0 R 4829 0 R 4831 0 R 4833 0 R 4
            835 0 R 4837 0 R 4839 0 R 4841 0 R 4843 0 R 4845 0 R 4847 0
            R 4849 0 R 4851 0 R 4853 0 R 4855 0 R 4857 0 R 4859 0 R 4861
             0 R 4863 0 R 4865 0 R 4867 0 R]
        [5661 0 R 5663 0 R 5665 0 R 5667 0 R 5669 0 R 5671 0 R 5673 0 R 5
            675 0 R 5677 0 R 5679 0 R 5681 0 R 5683 0 R 5685 0 R 5687 0
            R 5689 0 R 5691 0 R 5693 0 R 5695 0 R 5697 0 R 5699 0 R]
        [6355 0 R 6357 0 R 6359 0 R 6361 0 R 6363 0 R 6365 0 R 6367 0 R 6
            369 0 R 6371 0 R 6373 0 R 6375 0 R 6377 0 R 6379 0 R 6381 0
            R 6383 0 R 6385 0 R]
        [6951 0 R 6953 0 R 6955 0 R 6957 0 R 6959 0 R 6961 0 R 6963 0 R 6
            965 0 R 6967 0 R 6969 0 R 6971 0 R 6973 0 R 6975 0 R 6977 0
            R 6979 0 R]
    ] % Note: "<x> <y> R" is a pointer to the object defined with "<x> <y
        > obj" -- these all point to COG spacer objects
    [ % Paragraph 2
        [136 0 R 138 0 R 140 0 R 142 0 R 144 0 R 146 0 R 148 0 R 150 0 R
            152 0 R 154 0 R 156 0 R 158 0 R 160 0 R
%%% truncated %%%
```

Listing 5: An excerpt from a Galley Structure Tree in a malleable document PDF file. The tree structure described in Figure 10 can be observed in the /Paragraphs array, with the exception that the first element is an array containing the widths of each included galley rendering.

```
2303 0 obj
    <<
        /Type /XObject
        /Subtype /Form
        /Name /Cog8f692dee-5919-11e0-90bd-9eb109602c3e
        /FormType 1
        /BBox [0.000000 0.000000 82.933000 9.889000  ]
        /Baseline 2.387000
        /Indent 25.000000
        /Length 219
        /Resources <<
            /Font <<
                /R 3 0 R
            >>
            /ProcSet [/PDF /Text  ]
        >>
    >>
    stream
        q
        0.125000 0 0 0.125000 0 0 cm
        BT
        /R 1 Tf
        88.000000 0 0 88.000000 0.000000 19.096000 Tm (Lorem ) Tj
        88.000000 0 0 88.000000 448.000000 19.096000 Tm (i) Tj
        88.000000 0 0 88.000000 473.000000 19.096000 Tm (psum ) Tj
        ET
        Q
    endstream
endobj

2304 0 obj
    <<
        /Type /CogReference
        /Height 9.889000
        /Length 83
        /ptrTo /Cog8f692dee-5919-11e0-90bd-9eb109602c3e
        /X 25.000000
        /Width 82.933000
        /Y -14118.497000
    >>
    stream
        q 1 0 0 1 25.000000 -14118.497000 cm /Cog8f692dee-5919-11e0-90bd-
            9eb109602c3e Do Q
    endstream
endobj
```

Listing 6: A cog (object 2303) and its associated spacer (object 2304) from a malleable pdf. The cog object is never modified. To reposition the cog on the page, the spacer object is deleted and replaced with a new one that has the required /X and /Y values set.

```
7471 0 obj
<<
 /Type /Page
 /Subtype /Cogified
 /MediaBox [0 0 595 841]
 /Contents [5 0 R 7 0 R 9 0 R 11 0 R 13 0 R 15 0 R 17 0 R 19 0 R 21 0 R 2
     3 0 R 25 0 R 27 0 R 29 0 R 31 0 R 33 0 R 35 0 R 37 0 R 39 0 R 41 0
     R 43 0 R 45 0 R 47 0 R 49 0 R 51 0 R 53 0 R 55 0 R 57 0 R 59 0 R 61
      0 R 63 0 R 65 0 R 67 0 R 69 0 R 71 0 R 73 0 R 75 0 R 77 0 R 79 0 R
      81 0 R 83 0 R 85 0 R 87 0 R 89 0 R 91 0 R 93 0 R 95 0 R 97 0 R 99
     0 R 101 0 R 103 0 R 105 0 R 107 0 R 109 0 R 111 0 R 113 0 R 115 0 R
      117 0 R 119 0 R 121 0 R 123 0 R 125 0 R 127 0 R
%%% truncated %%%
```

Listing 7: An excerpt from a malleable PDF file showing that the /Contents of a /Page can be an array of content streams rather than one stream. In this case, each of the objects being pointed to are COG spacers, as shown in Listing 6. Consequently, when a spacer object is deleted and a new one created (to reposition a COG on the page) the /Contents array must be modified to reflect this.

By the point the document is to be displayed in Acrobat, most of the computationally expensive typesetting has already been carried out, which means that the algorithm used to lay out the lines of the selected galley can be very simple.

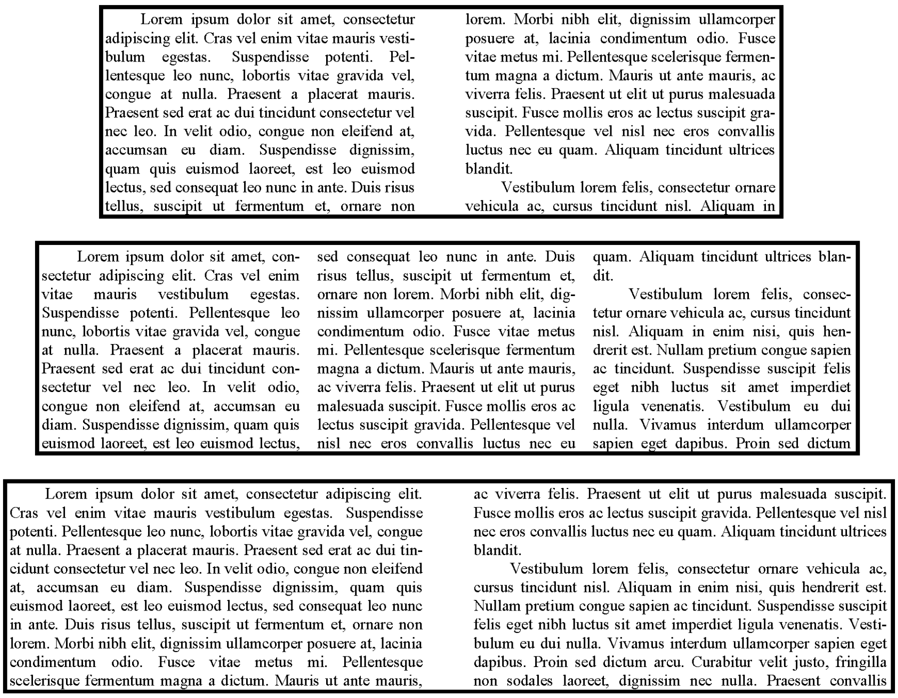
The plugin chooses the most appropriate galley width to lay out, based on the current page width, and according to some measure of aesthetics (see Sections 2.6 and 5.2) and then simply lays the document out line by line, with appropriate vertical spacing, until no more lines will fit in the current column. Any subsequent columns that will fit on the same page are then laid out in the same manner.

For convenience of testing, the plugin also automatically resizes the page to that of the window of Acrobat, and re-lays out the text on the fly, allowing various combinations of page sizes, zoom levels, and aspect ratios to be tried out. Some sample renderings from the Acrobat plugin are shown in Figure 11, and an excerpt of the code used for the layout is shown in Listing 8.

2.5 INCLUDED GALLEY RENDERINGS

In order for a document to achieve optimal fit on every device of every conceivable size, we must produce one galley rendering of every conceivable width. Clearly, this is infeasible, both in terms of space and time. For reasons of practicality, we are therefore forced to choose some finite subset of galley widths to include within a document.

In *The Elements of Typographic Style*,[Brio8] typographer Robert Bringhurst states that lines that range in length between 45 and 75 characters (including both letters and spaces) produce satisfactory layouts for a single-column page, and further that the 66-character line is widely



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras vel enim vitae mauris vestibulum egestas. Suspendisse potenti. Pellentesque leo nunc, lobortis vitae gravida vel, congue at nulla. Praesent a placerat mauris. Praesent sed erat ac dui tincidunt consectetur vel nec leo. In velit odio, congue non eleifend at, accumsan eu diam. Suspendisse dignissim, quam quis euismod laoreet, est leo euismod lectus, sed consequat leo nunc in ante. Duis risus tellus, suscipit ut fermentum et, ornare non lorem. Morbi nibh elit, dignissim ullamcorper posuere at, lacinia condimentum odio. Fusce vitae metus mi. Pellentesque scelerisque fermentum magna a dictum. Mauris ut ante mauris, ac viverra felis. Praesent ut elit ut purus malesuada suscipit. Fusce mollis eros ac lectus suscipit gravida. Pellentesque vel nisl nec eros convallis luctus nec eu quam. Aliquam tincidunt ultrices blandit.

Vestibulum lorem felis, consectetur ornare vehicula ac, cursus tincidunt nisl. Aliquam in

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras vel enim vitae mauris vestibulum egestas. Suspendisse potenti. Pellentesque leo nunc, lobortis vitae gravida vel, congue at nulla. Praesent a placerat mauris. Praesent sed erat ac dui tincidunt consectetur vel nec leo. In velit odio, congue non eleifend at, accumsan eu diam. Suspendisse dignissim, quam quis euismod laoreet, est leo euismod lectus, sed consequat leo nunc in ante. Duis risus tellus, suscipit ut fermentum et, ornare non lorem. Morbi nibh elit, dignissim ullamcorper posuere at, lacinia condimentum odio. Fusce vitae metus mi. Pellentesque scelerisque fermentum magna a dictum. Mauris ut ante mauris, ac viverra felis. Praesent ut elit ut purus malesuada suscipit. Fusce mollis eros ac lectus suscipit gravida. Pellentesque vel nisl nec eros convallis luctus nec eu quam. Aliquam tincidunt ultrices blandit.

Vestibulum lorem felis, consectetur ornare vehicula ac, cursus tincidunt nisl. Aliquam in enim nisi, quis hendrerit est. Nullam pretium congue sapien ac tincidunt. Suspendisse suscipit felis eget nibh luctus sit amet imperdiet ligula venenatis. Vestibulum eu dui nulla. Vivamus interdum ullamcorper sapien eget dapibus. Proin sed dictum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras vel enim vitae mauris vestibulum egestas. Suspendisse potenti. Pellentesque leo nunc, lobortis vitae gravida vel, congue at nulla. Praesent a placerat mauris. Praesent sed erat ac dui tincidunt consectetur vel nec leo. In velit odio, congue non eleifend at, accumsan eu diam. Suspendisse dignissim, quam quis euismod laoreet, est leo euismod lectus, sed consequat leo nunc in ante. Duis risus tellus, suscipit ut fermentum et, ornare non lorem. Morbi nibh elit, dignissim ullamcorper posuere at, lacinia condimentum odio. Fusce vitae metus mi. Pellentesque scelerisque fermentum magna a dictum. Mauris ut ante mauris, ac viverra felis. Praesent ut elit ut purus malesuada suscipit. Fusce mollis eros ac lectus suscipit gravida. Pellentesque vel nisl nec eros convallis luctus nec eu quam. Aliquam tincidunt ultrices blandit.

Vestibulum lorem felis, consectetur ornare vehicula ac, cursus tincidunt nisl. Aliquam in enim nisi, quis hendrerit est. Nullam pretium congue sapien ac tincidunt. Suspendisse suscipit felis eget nibh luctus sit amet imperdiet ligula venenatis. Vestibulum eu dui nulla. Vivamus interdum ullamcorper sapien eget dapibus. Proin sed dictum arcu. Curabitur velit justo, fringilla non sodales laoreet, dignissim nec nulla. Praesent convallis

Figure 11: Three sample renderings from the Acrobat plugin, with page widths of 42, 48, and 54 em. Note that the middle rendering is considered to be a "better fit" than either the top or the bottom renderings, since the space between columns has been reduced, but that in each case, the galley rendering chosen is the one that minimises the space between columns for that particular page width.

```
int numPars = CosArrayLength(parArray) - 1; // first element is not a paragraph
CosObj widthsArray = CosArrayGet(parArray, 0); // it is the widths array
int numWidths = CosArrayLength(widthsArray);
ASFixed *widths = (ASFixed*)calloc(numWidths, sizeof(ASFixed));
double *badness = (double*)calloc(numWidths, sizeof(double));

int min_badness_index = 0;
int numCols = 0;

// Choose the "best" rendering
for (int i = 0; i < numWidths; i++) {
    widths[i] = CosFixedValue(CosArrayGet(widthsArray,i));
    int nc = pageWidth / (widths[i] + mingutter);
    ASFixed ex_ws = pageWidth % (widths[i] + mingutter);

    badness[i] = (ex_ws / 65536.0 + 100) * sqrt((double)nc);

    if (nc == 0) badness[i] = 1000 * i;

    if (badness[i] <= badness[min_badness_index]) {
        min_badness_index = i;
        numCols = nc;
    }
}

ASFixed colsep = pageWidth / numCols;
ASFixed linesep = 13<<16; // 13pt. (Ideally, infer from data within PDF)
ASFixed parsep = 0;

int curr_x = mediabox.left + (colsep - widths[min_badness_index]) / 2;
int curr_y = mediabox.top - topmargin;

CosObj newContents = CosNewArray(cosDoc, false, 100);

// Create new spacer objects with correct COGs, and insert into newContents
for (int p = 0; p < numPars; p++) {
    CosObj para = CosArrayGet(CosArrayGet(parArray, p + 1), min_badness_index);
    int numLines = CosArrayLength(para);
    for (int l = 0; l < numLines; l++) {
        CosObj baseline = CosDictGet(CosStreamDict(CosArrayGet(para, l)),
             ASAtomFromString("Baseline"));
        CosObj indent = CosDictGet(CosStreamDict(CosArrayGet(para, l)),
             ASAtomFromString("Indent"));
        CosObj name = CosDictGet(CosStreamDict(CosArrayGet(para, l)),
             ASAtomFromString("Name"));

        int y_offset = CosFixedValue(baseline);
        int x_offset = CosFixedValue(indent);

        if (curr_y < (mediabox.bottom + botmargin)) { //start new column
            curr_y = mediabox.top - topmargin;
            curr_x += colsep;
        }

        CCosDoc cDoc(cosDoc);
        CSpacerCreator spacerCreator(cDoc);
        CCosStream newSpacer = spacerCreator.Create(ASAtomGetString(CosNameValue(
             name)), curr_x + x_offset, curr_y - y_offset);

        CosArrayInsert(newContents, CosArrayLength(newContents), newSpacer);

        curr_y -= linesep;
    }
    curr_y -= parsep;
}

PDPageAddCosContents(page, newContents);
```

Listing 8: An excerpt from the C++ Acrobat plugin, showing the implementation of the layout algorithm described in Section 2.4.4.

regarded as ideal. He also asserts that an average of 40–50 characters is reasonable for multiple-column work. Furthermore, he states that line lengths exceeding 75–80 characters make continuous reading difficult, and that at the other end of the spectrum, lines shorter than 38–40 characters are unlikely to be easy to fully justify, and should therefore in most cases be set ragged right. Below 30 characters, Bringhurst states, even ragged right text begins to look “anorexic”.

On this basis, the choice of galleys to display should range from a minimum of 40 characters to a maximum of 75. To account for smaller screens or larger font sizes, it may be necessary to include galley renderings narrower than 40 characters. These galleys would be reserved for use only when the screen size is so narrow as to mandate it.

In order to establish a suitable range of galleys to include within these limits, some informal experimentation was performed on peers and colleagues. This suggested that there should be an absolute minimum of three galley renderings, and that the beneficial effects become less noticeable when there are more than seven or ten renderings included. A far more comprehensive user study, detailed in Chapter 5, covers this in more detail.

2.6 THE BEST GALLEY?

As discussed in Sections 2.1.1, 2.2, and 2.3, galleys of text lend themselves to being used in a columnar format, and therefore a method of fitting columns appropriately to the available page width must be devised.

A sensible first approach is simply to calculate how many columns of each galley rendering will fit, by adding the galley width to a specified minimum inter-column spacing, and dividing the page width by the result. The remainder of this division will then specify the amount of additional horizontal whitespace required, which can then divided up and inserted between the columns.

A simple measure of aesthetic quality here is to apply a linear penalty for any extra whitespace required, as we seek to keep page margins and column gutters to a minimum.

Equations 1 and 2 below show the formulae used to calculate the number of columns that will fit ($\mathrm{N}_{\mathrm{cols}}$), and the requisite extra whitespace (S_{extra}). W_{page} is the total width of the page, W_{galley} is the width of the current galley, and W_{ICS} is the width of the minimum required inter-column spacing.

$$\mathrm{N}_{\mathrm{cols}} = \left\lfloor \frac{W_{\mathrm{page}}}{W_{\mathrm{galley}} + W_{\mathrm{ICS}}} \right\rfloor \tag{1}$$

$$S_{\text{extra}} = W_{\text{page}} \mod \left(W_{\text{galley}} + W_{\text{ICS}}\right) \tag{2}$$

As the page width increases, so must the widths of the inter-column gutters. In accordance with the extra-whitespace penalty, each galley rendering will produce penalties which vary in a sawtooth manner as the width of the page is increased. With a careful choice of galley widths, when these sawtooth penalties are overlaid, and the galley producing the minimum penalty chosen at each page width, a flatter and finer-toothed penalty graph emerges, as shown in Figure 12.

In addition to penalising extra whitespace, wider columns should, in general, be favoured over narrower ones, i.e. for a given page width, fewer, wider columns are generally considered preferable to a greater number of narrower columns. By multiplying the existing penalty by a smaller-than-linear function of the number of columns, the penalty may be subtly increased for greater numbers of columns.

The formula for the penalty used in Figure 12 is

$$P = (C + S_{\text{extra}}) \cdot \sqrt{N_{\text{cols}}} \tag{3}$$

where P is the penalty, S_{extra} is the extra whitespace required to be inserted (as computed in equation 2), N_{cols} is the number of columns that are required to fill the width of the page (as computed in equation 1), and C is a positive constant.

The purpose of the constant is to prevent the penalty from ever evaluating to zero, which would have the effect of disregarding the weighting of the number of columns. Figure 12 uses $C = 1$.

This is calculated for each galley rendering, and the galley with the minimum value of P is selected.

2.7 EFFICIENCY

It can easily be observed that the view-time complexity of the layout algorithm described in this chapter is linear.

At view-time, it must be decided which galley rendering to display: this is a trivial operation that requires one calculation per galley rendering. Assuming that there will never be more than ten galley renderings within one document (this seems reasonable given the discussion in Sections 2.5 and 5.3) the time taken to choose the best-fitting galley will always be dominated by the time taken to perform the layout itself.

Once the best-fitting galley rendering has been selected, all that is left is to traverse the Galley Structure Tree, laying out each line of text sequentially. When the bottom of the physical page is reached, text is then laid out in a new column adjacent to the previous one.

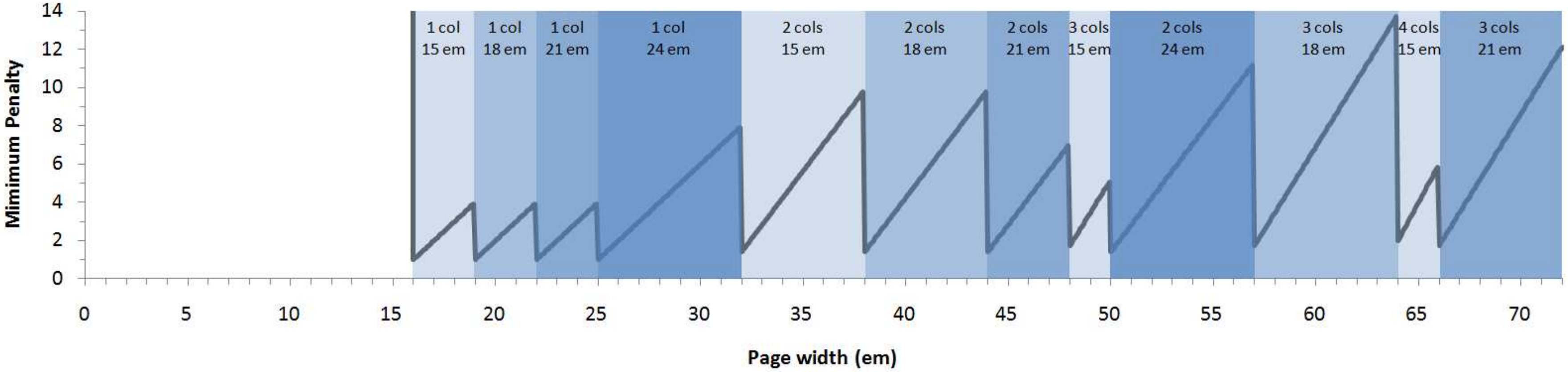


Figure 12: Graph showing the minimum penalty value of all galleys in a reflowable document, over a range of page widths. The particular document used contained four galleys; these were rendered at widths of 15, 18, 21 and 24 em, with a minimum gutter width of 1 em. Each vertical band highlights a range of page widths within which only the horizontal spacing of the page is altered. The boundaries between vertical bands represent a switch between galley renderings — the galley used and number of columns is as annotated on the graph.

If there are n lines of text in the document, this takes at most $k \cdot n$ operations, where k is some constant pertaining to the operations required to lay out one line of text, and therefore the view-time complexity is $O(n)$.

A first-fit (or greedy) layout algorithm, as used by most current ebook hardware and web browsers, also runs in $O(n)$, but proportional to the number of possible breakpoints rather than the number of lines of text. Due to using first-fit, the resultant layout will almost certainly be substandard in comparison to any high-quality pre-rendered layout, as used by the system described in this chapter. Conversely, to compute a higher-quality layout on the fly, (using, for example, the Knuth-Plass line breaking algorithm,[KP81, Knu99] or something even more complex) can take upwards of $O(n^2)$ [HL87, EG92, HLM09, PBB13].

2.8 SUMMARY

The malleable document system described in this chapter produces document layouts that are essentially indistinguishable from those produced by *troff* itself. Indeed, the layouts produced are comparable to those produced by any professional-level typesetting system that sets long text-based content into columns.

The drawbacks of this system are that the filesize is necessarily increased (since data for multiple layouts must be included), and that the galley rendering that is displayed may not fit the available page width as snugly as an algorithm that has been run on the fly (which would therefore be specifically tailored to the dimensions of the page).

The implementation of this algorithm within Adobe Acrobat is somewhat clunky — and certainly impractical to deploy on a real ebook platform — but it demonstrates that the concept of pre-rendering several variants is a viable means to producing well-typeset flowable layouts.

The one notable omission from this chapter is support for floating blocks (such as figures) — only non-floating blocks are supported. In the next chapter, both of these issues are addressed.

3 FLOATABLE BLOCKS

The system described in Chapter 2 supports only simple documents that are composed solely from text. Most documents contain figures, diagrams, illustrations, or tables, and so we must give consideration towards how these should be handled.

In this chapter, we extend the work of the previous chapter to allow floatable graphical blocks, whose absolute position within a document's text may vary depending upon the layout.

Furthermore, we reimplement the system described in the previous chapter, moving away from PDF *and Adobe Acrobat, towards a new system based in* HTML, CSS *and JavaScript that is much more portable and as such can be used on ebook hardware.*

The research in this chapter was previously published in [PBB13]

The implementation of the system described in the previous chapter is deeply rooted within PDF, and requires a custom-written plugin for Adobe Acrobat (see Section 2.4.4) to view the documents. Consequently, it is difficult to test that particular implementation on any device that is not running Microsoft Windows and does not have a fully licensed version of Adobe Acrobat, which effectively rules out any mobile ebook readers. Almost all ebook readers support the EPUB format, which is principally built upon HTML, CSS, and JavaScript. With this in mind, it was decided that the system should be reimplemented using these technologies, in order that it could be deployed on ebook hardware.

Amazon's Kindle *is one of the few contemporary devices that do not support EPUB*

3.1 DOCUMENT GENERATION

Since the new system no longer relies on COG-PDF, the reliance on *troff* and PDFDIT is no longer present. Consequently, a sensible approach is to produce a completely bespoke typesetting tool, allowing unneeded features to be removed, together with the provision of some finer-grained control over other aspects. For example, it is important for this system that the the line-breaking and hyphenation algorithms can easily be changed.

In the new system, the source document is described in terms of separate logical blocks: a block is either designated as a 'float', or as a 'paragraph'. (Listing 9 contains an excerpt from a sample source document.) Floats are currently limited to referencing images only (with an optional size parameter). Paragraphs, on the other hand, are described by their desired textual content. This is deliberately simple. It is envisaged that in a real system, the source document would have

```
3.1.3 pdfdit

Having generated the source document, it was processed with
ditroff to generate the intermediate code used to feed each
typesetter post-processor. This output is very expressive, and,
unlike TEX’s DVI, contains enough information that post-
processors are easily able to locate the start and end of lines
and paragraphs within the document. This meant that only minimal
changes were needed to be made to the pdfdit package described in
[1] to implement our design.

__FLOAT fig4.png

Figure 4: Sample renderings from the Acrobat plugin at page
widths of 42, 48, and 54 em.

__PARA

The first change necessary was to decrease the granularity of the
output COGs, producing them at the line level, rather than at the
paragraph level. Secondly, some method of generating the
requisite tree representing the document structure was required.
This was solved by simply using the point at which the original
version of pdfdit would have started a new paragraph-level COG,
and, instead, starting a new paragraph-level block entry in the
document structure tree. Each subsequent line-level COG produced
can then be added as a child of this block.

Once the entire output file has been parsed, the tree
representations of the various width galleys are amalgamated
per-paragraph, as indicated in figure 3, and finally the PDF file
is serialised, replete with COGs and content tree.
```

Listing 9: An excerpt from a sample source document, itself an excerpt from [PBB11]. The document is parsed from top to bottom. Paragraphs are separated by blank lines. Floats are specified by lines that begin `__FLOAT` and contain a reference to an image. Subsequent lines, until the next `__FLOAT` or `__PARA` marker, are interpreted as the float caption.

a richer language, perhaps using Markdown, XML, or a form similar to LaTeX source.

Next, the source document is passed through a program termed the *Paragraph Splitter* to produce the output that becomes the malleable document itself.

The Paragraph Splitter passes the text of each paragraph through an implementation of a line-breaking algorithm. The Knuth-Plass algorithm[KP81] was used by the Paragaph Splitter instance that produced the layouts shown in Appendix B, though it is trivial to replace this with any other line-breaking algorithm.

Each paragraph is rendered multiple times, once for each galley width, in order to produce the document’s multiple galley renderings. Each line of each rendering of every paragraph is converted into a list of its composite words. All of these words have an associated position offset value (as shown in Listing 11) which is later used when drawing the text, to ensure that each word is positioned on the line with the correct spacing. The general algorithm used is given in Listing 10.

The content of the floats is largely left unchanged. A reference to the image, along with its required dimensions, is simply passed through to the output. If dimensions were not explicitly specified in the source document, the pixel size of the image itself is used, at 96 DPI (i. e. 16 pixels becomes 12 points).

Finally, once the whole of the source document has been processed, the rendered content is output — in the form of the Galley Structure Tree shown in figure 10 on page 25 — encoded as a JavaScript Object Notation (JSON) string. This becomes the data representing the source document, which, in conjunction with the viewer defined in the next section, becomes a *malleable document*. A sample of this data is shown in Listing 11.

3.2 THE VIEWER

In order to circumvent the web browser's default text-layout algorithm, and to ensure that the "high quality" pre-computed text layout is used, the absolute position of every word on each line must be specified, in a manner not dissimilar to the internals of a PDF file. The Paragraph Splitter described in the previous section ensures that all the information needed to lay out the text is contained within the generated JSON string representing the Galley Structure Tree.

When the viewer is launched, it decides which is the most appropriate galley rendering to display, based on a metric of which rendering will be most aesthetically pleasing. Since it works well, the metric defined in Chapter 2 is used, which balances a penalty for excessive inter-column whitespace against a penalty for too many columns.

Although every galley is rendered in the same point size, this can be scaled up or down at view-time based on the preference of the user, to simulate point-size changes. All dimensions other than the page size, such as the gaps between words, are scaled proportionally, to allow the text to remain correctly justified.

3.2.1 *Floats with a Queue*

The first approach taken towards supporting floats took inspiration from TeX, which places floats into a queue until it finds somewhere it deems appropriate to place the first float [Pla81, Knu84]. In order to emulate this, two queues were defined: the *float queue*, and the *line queue*.

If both queues are empty, as they will be at the start of the layout process, the Galley Structure Tree is traversed, and when the first

```
galleyStrucTree renderDocument(documentContent, galleyWidths[]) {
    parasAndFloats[] = parseDocumentSource(documentContent);
    galleyStrucTree = empty tree;
    foreach (item in parasAndFloats) {
        if (item is a floatable object) {
            if (dimensions are not specified) {
                read pixel dimensions from file;
            }
            add floatable object to galleyStrucTree;
        } else { /* therefore item is a paragraph */
            create empty paragraph container;
            foreach (width in galleyWidths) {
                pass item text through linebreaker using width;
                create empty galley container;
                foreach (line returned by linebreaker) {
                    add words and positioning data of line to galley
                        container;
                }
                add galley container to paragraph container;
            }
            add paragraph container to galleyStrucTree;
        }
    }
    return galleyStrucTree;
}

parasAndFloats[] parseDocumentSource(documentContent) {
    step through documentContent line by line, returning the
        documentContent broken into an array of strings with one element
         per paragraph and per floatable object;
}
```

Listing 10: The algorithm followed by the Paragraph Splitter. Firstly the source of the document is parsed to break it into its initial logical blocks: one block per paragraph and one block per float, in the order encountered in the document source. These blocks are then processed further depending on their type. Floats may be probed for their pixel dimensions if no size was specified, and are then added to the Galley Structure Tree. Paragraphs have their content passed through a line breaking algorithm, once for each specified width.

```
[
  {
    "w": 952.5,
    "h": 342.75,
    "d": "<img style=\"width:100%\" src=\"fig0.png\" alt=\"Reflowable Documents
         Composed from\nPre-rendered Atomic Components\nAlexander J. Pinkney\
         nSteven R. Bagley\nDavid F. Brailsford\nDocument Engineering Lab.\nSchool
          of Computer Science\nUniversity of Nottingham\nNottingham, NG8 1BB, UK\n
         {azp|srb|dfb}@cs.nott.ac.uk\n\">"
  },
  [
    [
      [
        [0, "Abstract"]
      ]
    ],
    [
      [
        [0, "Abstract"]
      ]
    ],
    [
      [
        [0, "Abstract"]
      ]
    ]
  ],
  [
    [
      [
        [0, "Mobile"], [38.346, "eBook"], [73.356, "readers"]
      ],
      [
        [0, "are"], [17.334, "now"], [40.68, "commonplace"]
      ],
      [
        [0, "in"], [13.004, "today&#39;s"], [52, "society,"], [92.664, "but"]
      ],
      [
        [0, "their"], [26.334, "document"], [78, "layout"]
      ],
      [
        [0, "algorithms"], [53.736,"remain"], [89.46,"basic,"]
      ],
      [
        [0, "largely"],[35.724, "due"],[55.452, "to"],[67.188, "constraints"]
      ],
  /* truncated */
```

Listing 11: Excerpt from JavaScript data file representing a 3-galley document. Note that the title "Abstract" is treated as any normal paragraph and, as for any paragraph, is typeset once for each galley rendering (despite there being no difference between each rendering in this case). The first rendering of the first paragraph of the abstract begins below. For brevity's sake, subsequent renderings are not shown, but since the following galleys are typeset with a different measure, the spacing and words per line will differ. At the top is an object representing a float, which contains values for width, height, and data.

paragraph-level item (see Figure 10 on page 25) is encountered, its subcomponents (of the chosen galley rendering) are added to the requisite queue: lines to the line queue, and floats to the float queue.

When at least one of the queues is not empty, document layout begins. If the float queue is non-empty, and the first float in the queue will fit below the last typeset item, it is placed on the page. If not, items from the line queue are placed one by one, until no more will fit in the current column. When this happens, a new column is started, and the first float in the float queue is output. Whenever the line queue is depleted, and no floats in the float queue will fit at the current point on the page, all subcomponents of the next paragraph-level item from the Galley Structure Tree are queued. This process is illustrated in Figure 13.

Pagination is reasonably simple with this queueing system: as soon as a page is full, the layout can be restarted at the origin of the page using the current status of both queues and the Galley Structure Tree. Whilst this approach does produce professional-looking layouts, and handles floats well without the need for backtracking, it is not particularly conducive to producing layouts with floats that span multiple columns. The queue-based layout described above is rather simplistic: it knows about the size of each component that it lays out, but it does not remember the history of the positions of any of the components that are already laid out. This makes it difficult to have items that span more than one column, because there is no mechanism to mark space on the page as being reserved. In order to do this, another approach must be taken.

3.2.2 *A Grid-Based Layout*

One method for allowing regions of a page to be reserved is to break up the page into a grid. Grid-based layouts are useful in many situations; [Col91, Bri08] one example of particular note is that of modern-day newspapers (see figure 14). Following the example set by these newspapers, the grid used in this system is defined to have a row height of the leading of the document's body text, and a column width of the measure of one text column plus the required gutter space.

The viewer uses the dimensions of the float, as specified in the Galley Structure Tree (see Section 3.1), to determine how many columns it should span. The float is scaled to span the integer multiple of column widths that most closely matches its 'natural' size, though for reasons that should hopefully be obvious, this number is limited to a minimum of 1, and a maximum of the number of columns on the page. Additionally, checks are made to ensure that the scaling will not cause the height of the figure to exceed that of the page.

An advantage of this grid-based approach is that it no longer requires the use of queues, either for lines, or for floats. The viewer

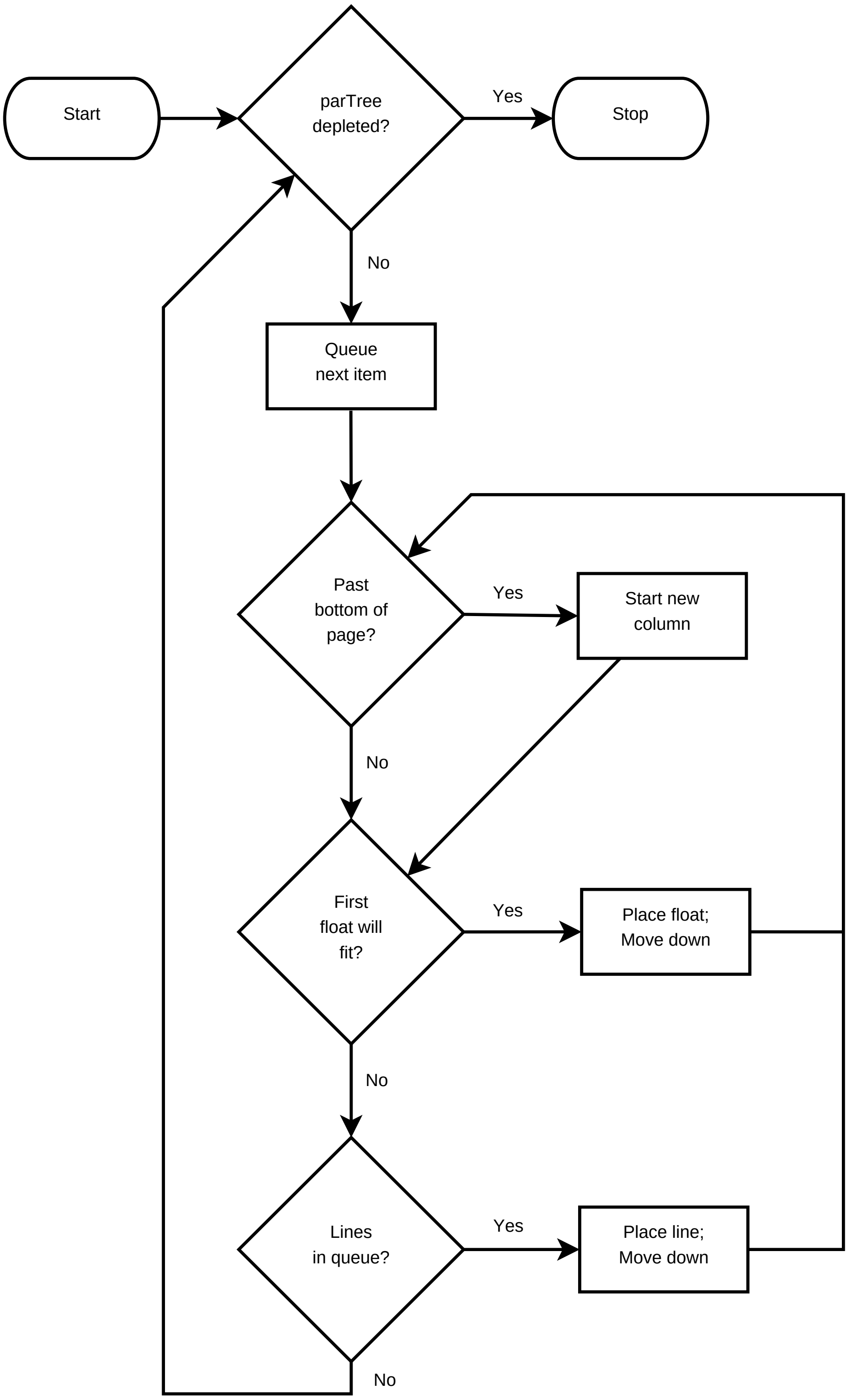
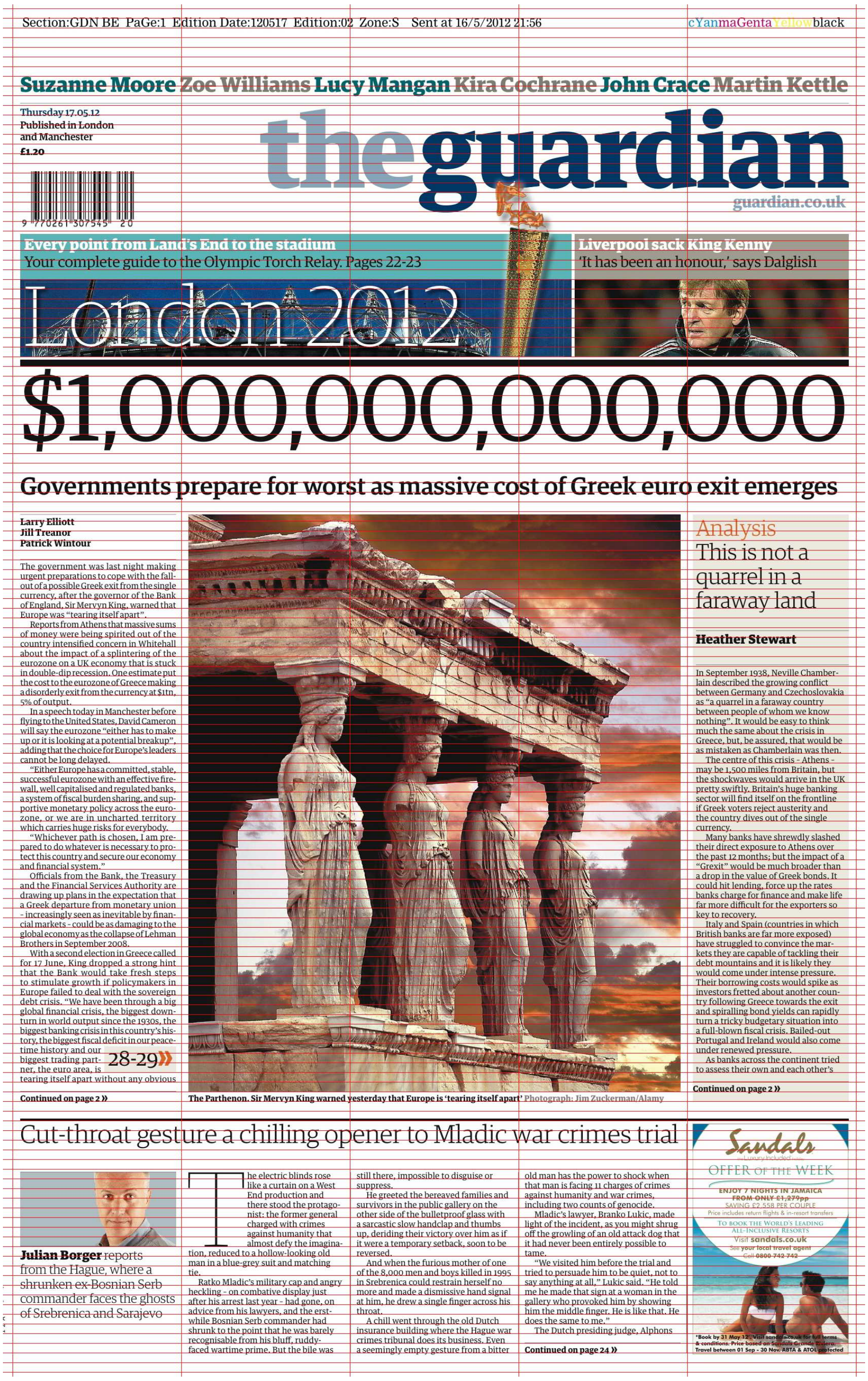


Figure 13: Flowchart describing the two-queue float algorithm. Section 3.2.1 explains this process in detail.



Section:GDN BE PaGe:1 Edition Date:120517 Edition:02 Zone:S Sent at 16/5/2012 21:56 cYanmaGentaYellowblack

Suzanne Moore Zoe Williams Lucy Mangan Kira Cochrane John Crace Martin Kettle

Thursday 17.05.12
Published in London
and Manchester
£1.20

9 770261 307545 20

theguardian
guardian.co.uk

Every point from Land's End to the stadium
Your complete guide to the Olympic Torch Relay. Pages 22-23

London 2012

Liverpool sack King Kenny
'It has been an honour,' says Dalglish

$1,000,000,000,000

Governments prepare for worst as massive cost of Greek euro exit emerges

Larry Elliott
Jill Treanor
Patrick Wintour

The government was last night making urgent preparations to cope with the fallout of a possible Greek exit from the single currency, after the governor of the Bank of England, Sir Mervyn King, warned that Europe was "tearing itself apart".

Reports from Athens that massive sums of money were being spirited out of the country intensified concern in Whitehall about the impact of a splintering of the eurozone on a UK economy that is stuck in double-dip recession. One estimate put the cost to the eurozone of Greece making a disorderly exit from the currency at $1tn, 5% of output.

In a speech today in Manchester before flying to the United States, David Cameron will say the eurozone "either has to make up or it is looking at a potential breakup", adding that the choice for Europe's leaders cannot be long delayed.

"Either Europe has a committed, stable, successful eurozone with an effective firewall, well capitalised and regulated banks, a system of fiscal burden sharing, and supportive monetary policy across the eurozone, or we are in uncharted territory which carries huge risks for everybody.

"Whichever path is chosen, I am prepared to do whatever is necessary to protect this country and secure our economy and financial system."

Officials from the Bank, the Treasury and the Financial Services Authority are drawing up plans in the expectation that a Greek departure from monetary union – increasingly seen as inevitable by financial markets – could be as damaging to the global economy as the collapse of Lehman Brothers in September 2008.

With a second election in Greece called for 17 June, King dropped a strong hint that the Bank would take fresh steps to stimulate growth if policymakers in Europe failed to deal with the sovereign debt crisis. "We have been through a big global financial crisis, the biggest downturn in world output since the 1930s, the biggest banking crisis in this country's history, the biggest fiscal deficit in our peacetime history and our biggest trading partner, the euro area, is tearing itself apart without any obvious

28-29»

Continued on page 2 »

The Parthenon. Sir Mervyn King warned yesterday that Europe is 'tearing itself apart' Photograph: Jim Zuckerman/Alamy

Analysis

This is not a quarrel in a faraway land

Heather Stewart

In September 1938, Neville Chamberlain described the growing conflict between Germany and Czechoslovakia as "a quarrel in a faraway country between people of whom we know nothing". It would be easy to think much the same about the crisis in Greece, but, be assured, that would be as mistaken as Chamberlain was then.

The centre of this crisis – Athens – may be 1,500 miles from Britain, but the shockwaves would arrive in the UK pretty swiftly. Britain's huge banking sector will find itself on the frontline if Greek voters reject austerity and the country dives out of the single currency.

Many banks have shrewdly slashed their direct exposure to Athens over the past 12 months; but the impact of a "Grexit" would be much broader than a drop in the value of Greek bonds. It could hit lending, force up the rates banks charge for finance and make life far more difficult for the exporters so key to recovery.

Italy and Spain (countries in which British banks are far more exposed) have struggled to convince the markets they are capable of tackling their debt mountains and it is likely they would come under intense pressure. Their borrowing costs would spike as investors fretted about another country following Greece towards the exit and spiralling bond yields can rapidly turn a tricky budgetary situation into a full-blown fiscal crisis. Bailed-out Portugal and Ireland would also come under renewed pressure.

As banks across the continent tried to assess their own and each other's

Continued on page 2 »

Cut-throat gesture a chilling opener to Mladic war crimes trial

Julian Borger reports from the Hague, where a shrunken ex-Bosnian Serb commander faces the ghosts of Srebrenica and Sarajevo

The electric blinds rose like a curtain on a West End production and there stood the protagonist: the former general charged with crimes against humanity that almost defy the imagination, reduced to a hollow-looking old man in a blue-grey suit and matching tie.

Ratko Mladic's military cap and angry heckling – on combative display just after his arrest last year – had gone, on advice from his lawyers, and the erstwhile Bosnian Serb commander had shrunk to the point that he was barely recognisable from his bluff, ruddy-faced wartime prime. But the bile was still there, impossible to disguise or suppress.

He greeted the bereaved families and survivors in the public gallery on the other side of the bulletproof glass with a sarcastic slow handclap and thumbs up, deriding their victory over him as if it were a temporary setback, soon to be reversed.

And when the furious mother of one of the 8,000 men and boys killed in 1995 in Srebrenica could restrain herself no more and made a dismissive hand signal at him, he drew a single finger across his throat.

A chill went through the old Dutch insurance building where the Hague war crimes tribunal does its business. Even a seemingly empty gesture from a bitter old man has the power to shock when that man is facing 11 charges of crimes against humanity and war crimes, including two counts of genocide.

Mladic's lawyer, Branko Lukic, made light of the incident, as you might shrug off the growling of an old attack dog that it had never been entirely possible to tame.

"We visited him before the trial and tried to persuade him to be quiet, not to say anything at all," Lukic said. "He told me he made that sign at a woman in the gallery who provoked him by showing him the middle finger. He is like that. He does the same to me."

The Dutch presiding judge, Alphons

Continued on page 24 »

Figure 14: An example of a grid-based layout in a UK newspaper. Note how all the baselines of the main body text are aligned to a common grid, and that all items span integer multiples of columns.

simply traverses the Galley Structure Tree, placing each item in the first available place in the grid. In the case of floats, or other items larger than multiples of the main leading, spaces in the grid can be marked as reserved, to prevent other items from trampling over their reserved space. If a float will not fit directly below the previous item to be placed, the grid is walked over until a gap of sufficient size can be found. Figure 15 shows the progressive stages of this algorithm, and Figure 16 shows a real example of a document laid out with this system.

Pagination becomes a little trickier when floats are allowed to span multiple columns. For example, if a float whose natural size would lead it to span n columns is encountered in the Galley Structure Tree when there are fewer than n columns remaining to be typeset on the page, it must be decided how best to handle the situation. Three obvious options present themselves: alter the float to span fewer columns; delay the placement of the float until the start of the next page; or backtrack and check whether there is room to move the float back one or more columns, by shunting non-floatable text lines forwards.

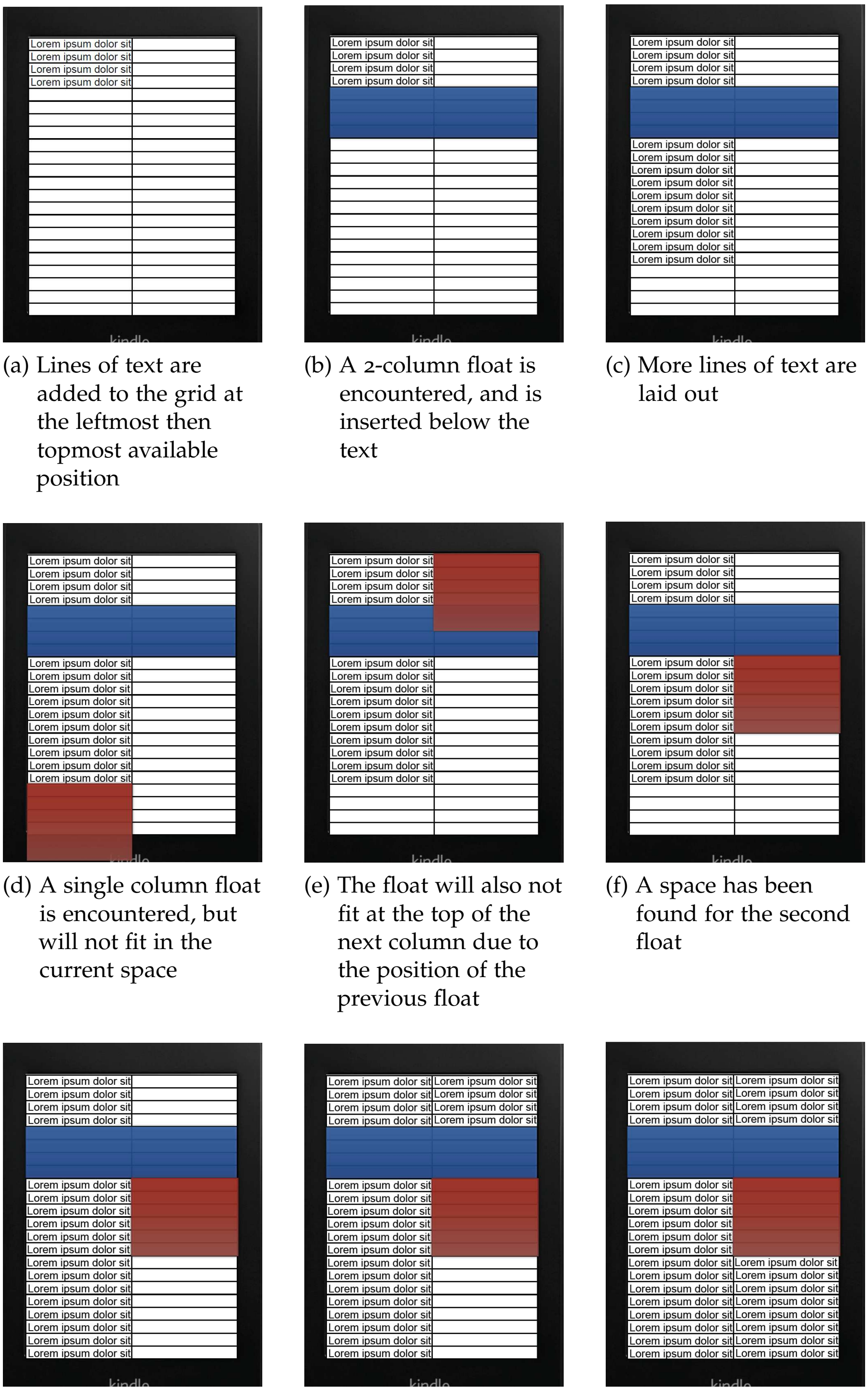
The first option is clearly not ideal behaviour, given that shrinking a float may well reduce its legibility. Additionally, if this becomes a common problem, it is likely to be noticeable that floats spanning into the rightmost column of the page appear shrunken.

The second option (delaying placement until the following page) is a reasonable compromise, though it will increase float-drift (whereby floats become separated from their callout points in the text), which again is not ideal.

The third option (backtracking and shunting) is likely to produce the most desirable output, although some computational overhead will be added. One approach is simply to check whether there is enough space immediately to the left (specifically a gap between other, already placed, floats) into which the current float can be placed, with the displaced lines of text being shunted forwards. This method will not produce layouts as optimal as methods that use full backtracking and check all possibilities, but will run in much quicker time. A combination of all three of the above options is likely to work best in practice.

3.3 SUMMARY

The reimplementation of the system in HTML is not without its pitfalls. One strong advantage of using PDF over HTML is that any PDF renderer that correctly implements the standard[Ado01] should display any given document in an identical manner to any other renderer. Regrettably (and much to the chagrin of web developers everywhere) this is very much not true of rival web browser layout engines. Though stand-



(a) Lines of text are added to the grid at the leftmost then topmost available position

(b) A 2-column float is encountered, and is inserted below the text

(c) More lines of text are laid out

(d) A single column float is encountered, but will not fit in the current space

(e) The float will also not fit at the top of the next column due to the position of the previous float

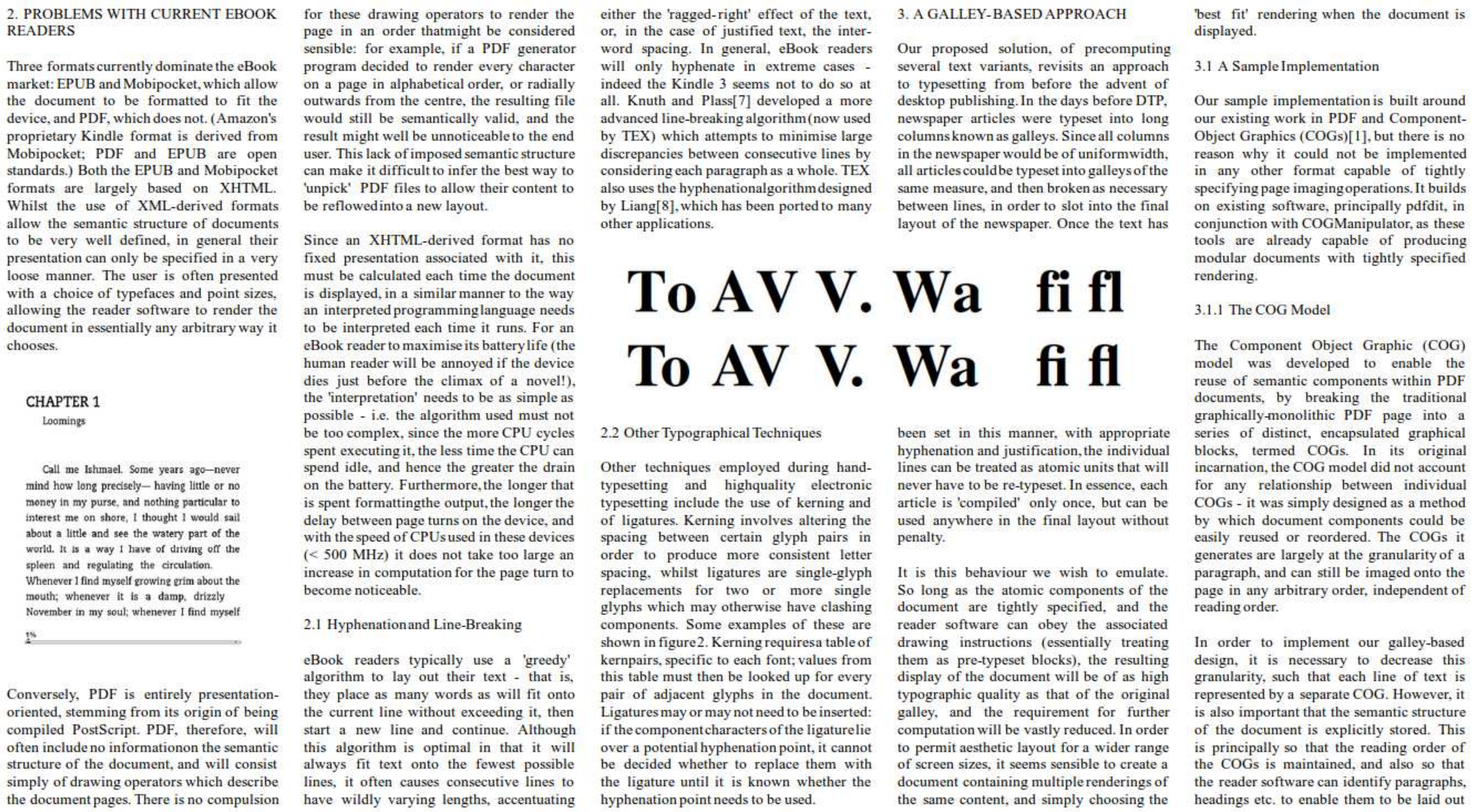
(f) A space has been found for the second float

(g) More text is laid out until the bottom of the column is reached

(h) Text begins to be laid out at the top of the next column, until the reserved space of the float is encountered

(i) Text is subsequently laid out below the two floats until the page is filled

Figure 15: A step-by-step example of how multi-column spanning floats are positioned using the grid-based layout described in Section 3.2.2



2. PROBLEMS WITH CURRENT EBOOK READERS

Three formats currently dominate the eBook market: EPUB and Mobipocket, which allow the document to be formatted to fit the device, and PDF, which does not. (Amazon's proprietary Kindle format is derived from Mobipocket; PDF and EPUB are open standards.) Both the EPUB and Mobipocket formats are largely based on XHTML. Whilst the use of XML-derived formats allow the semantic structure of documents to be very well defined, in general their presentation can only be specified in a very loose manner. The user is often presented with a choice of typefaces and point sizes, allowing the reader software to render the document in essentially any arbitrary way it chooses.

CHAPTER 1

Loomings

Call me Ishmael. Some years ago—never mind how long precisely— having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself

1%

Conversely, PDF is entirely presentation-oriented, stemming from its origin of being compiled PostScript. PDF, therefore, will often include no informationon the semantic structure of the document, and will consist simply of drawing operators which describe the document pages. There is no compulsion for these drawing operators to render the page in an order thatmight be considered sensible: for example, if a PDF generator program decided to render every character on a page in alphabetical order, or radially outwards from the centre, the resulting file would still be semantically valid, and the result might well be unnoticeable to the end user. This lack of imposed semantic structure can make it difficult to infer the best way to 'unpick' PDF files to allow their content to be reflowed into a new layout.

Since an XHTML-derived format has no fixed presentation associated with it, this must be calculated each time the document is displayed, in a similar manner to the way an interpreted programming language needs to be interpreted each time it runs. For an eBook reader to maximise its battery life (the human reader will be annoyed if the device dies just before the climax of a novel!), the 'interpretation' needs to be as simple as possible - i.e. the algorithm used must not be too complex, since the more CPU cycles spent executing it, the less time the CPU can spend idle, and hence the greater the drain on the battery. Furthermore, the longer that is spent formattingthe output, the longer the delay between page turns on the device, and with the speed of CPUs used in these devices (< 500 MHz) it does not take too large an increase in computation for the page turn to become noticeable.

2.1 Hyphenation and Line-Breaking

eBook readers typically use a 'greedy' algorithm to lay out their text - that is, they place as many words as will fit onto the current line without exceeding it, then start a new line and continue. Although this algorithm is optimal in that it will always fit text onto the fewest possible lines, it often causes consecutive lines to have wildly varying lengths, accentuating either the 'ragged-right' effect of the text, or, in the case of justified text, the inter-word spacing. In general, eBook readers will only hyphenate in extreme cases - indeed the Kindle 3 seems not to do so at all. Knuth and Plass[7] developed a more advanced line-breaking algorithm (now used by TEX) which attempts to minimise large discrepancies between consecutive lines by considering each paragraph as a whole. TEX also uses the hyphenationalgorithm designed by Liang[8], which has been ported to many other applications.

To AV V. Wa fi fl

To AV V. Wa fi fl

2.2 Other Typographical Techniques

Other techniques employed during hand-typesetting and highquality electronic typesetting include the use of kerning and of ligatures. Kerning involves altering the spacing between certain glyph pairs in order to produce more consistent letter spacing, whilst ligatures are single-glyph replacements for two or more single glyphs which may otherwise have clashing components. Some examples of these are shown in figure2. Kerning requiresa table of kernpairs, specific to each font; values from this table must then be looked up for every pair of adjacent glyphs in the document. Ligatures may or may not need to be inserted: if the componentcharacters of the ligaturelie over a potential hyphenation point, it cannot be decided whether to replace them with the ligature until it is known whether the hyphenation point needs to be used.

3. A GALLEY-BASED APPROACH

Our proposed solution, of precomputing several text variants, revisits an approach to typesetting from before the advent of desktop publishing. In the days before DTP, newspaper articles were typeset into long columns known as galleys. Since all columns in the newspaper would be of uniformwidth, all articles could be typeset into galleys of the same measure, and then broken as necessary between lines, in order to slot into the final layout of the newspaper. Once the text has been set in this manner, with appropriate hyphenation and justification, the individual lines can be treated as atomic units that will never have to be re-typeset. In essence, each article is 'compiled' only once, but can be used anywhere in the final layout without penalty.

It is this behaviour we wish to emulate. So long as the atomic components of the document are tightly specified, and the reader software can obey the associated drawing instructions (essentially treating them as pre-typeset blocks), the resulting display of the document will be of as high typographic quality as that of the original galley, and the requirement for further computation will be vastly reduced. In order to permit aesthetic layout for a wider range of screen sizes, it seems sensible to create a document containing multiple renderings of the same content, and simply choosing the 'best fit' rendering when the document is displayed.

3.1 A Sample Implementation

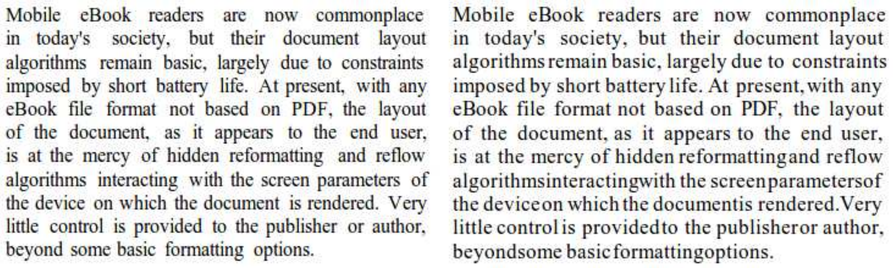
Our sample implementation is built around our existing work in PDF and Component-Object Graphics (COGs)[1], but there is no reason why it could not be implemented in any other format capable of tightly specifying page imaging operations. It builds on existing software, principally pdfdit, in conjunction with COGManipulator, as these tools are already capable of producing modular documents with tightly specified rendering.

3.1.1 The COG Model

The Component Object Graphic (COG) model was developed to enable the reuse of semantic components within PDF documents, by breaking the traditional graphically-monolithic PDF page into a series of distinct, encapsulated graphical blocks, termed COGs. In its original incarnation, the COG model did not account for any relationship between individual COGs - it was simply designed as a method by which document components could be easily reused or reordered. The COGs it generates are largely at the granularity of a paragraph, and can still be imaged onto the page in any arbitrary order, independent of reading order.

In order to implement our galley-based design, it is necessary to decrease this granularity, such that each line of text is represented by a separate COG. However, it is also important that the semantic structure of the document is explicitly stored. This is principally so that the reading order of the COGs is maintained, and also so that the reader software can identify paragraphs, headings etc. to enable them to be laid out

Figure 16: An excerpt from [PBB11], typeset and rendered by the system described in Chapter 3. Of the two floats (a screenshot from a Kindle, and a demonstration of some micro-typographical techniques), one spans only one column, and one spans two.



Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to the publisher or author, beyond some basic formatting options.

Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformattingand reflow algorithmsinteractingwith the screenparametersof the deviceon which the documentis rendered.Very little control is providedto the publisheror author, beyondsome basicformattingoptions.

Figure 17: Chrome and Safari's layout engine (WebKit) can be inconsistent when scaling fonts. The paragraph to the left is displayed "correctly", but the paragraph to the right (scaled up by 1% by the JavaScript) has clearly not scaled linearly. In contrast, Firefox's layout engine, Gecko, scales smoothly, and behaves as one might expect when zooming in on a PDF file, which was the intended behaviour. (Output from Gecko is not shown in this figure.)

ards do exist, published by the World Wide Web Consortium (W3C),[1] which are largely adhered to by all the major layout engines, there are certain areas that appear to be open to interpretation. One particular problem that is encountered, when using the system described in this chapter, is that of font scaling.

The layouts produced by this system will only retain their quality if they are laid out precisely as specified by the typesetting process. Viewing the malleable documents in Google Chrome and Mozilla Firefox (which use WebKit and Gecko respectively) produces noticeably different results when the user chooses to scale the page up or down. WebKit appears to be inconsistent and non-linear when scaling fonts, whereas Gecko does a better job, and manages to lay out the text as intended by the typesetter. Figure 17 shows an example of WebKit's font scaling problem. There is no real way around this issue, other than ensuring that whichever rendering engine is used in the reader device provides the required smooth scaling of fonts.

This is less of a problem on mobile devices: although in general their web browsers use the WebKit engine, their pixel-densities are much higher (many mobile phones have higher resolution screens than many laptops) so their displays are vastly scaled up, and fewer bad spacing issues such as those shown in Figure 17 occur.

In general, however, as a document layout engine, the system produces professional-looking layouts that scale to fit on many different screen sizes, and the reimplementation in XHTML, CSS, and JavaScript makes the system much more portable. One stand-out problem remains: that of bloated filesizes. The next chapter proposes several methods to keep file sizes to a minimum.

[1] See http://www.w3.org/standards/ for full details

4 DEALING WITH FILE BLOAT

In Chapter 3 we converted the implementation of our previously developed system into one that uses HTML, CSS, and JavaScript in place of PDF and Adobe Acrobat, which vastly increases its portability. We also added support for floating items, which allows us to produce document layouts that are extremly similar to those used in newspapers, magazines, and academic journals.

In this chapter, we turn around and address the elephant in the room: what effect does the inclusion of all these renderings have upon file size, and what can we do about it?

4.1 RATIONALE

In the system described so far, the emphasis has been firmly upon reducing the computational complexity of layout operations at document view-time, and therefore little consideration has been given to the filesize of the output malleable documents.

The tradeoff between filesize and required computation has previously been justified on the basis that storage is cheap, light, and small, and that batteries, although relatively inexpensive, already comprise a significant portion of the overall mass and volume of most portable devices that are suitable for reading ebooks. The consequence of this is that adding more storage would have little impact upon devices' aesthetics, but adding extra battery life (emerging nanotube battery technology notwithstanding) would result in vast increases in devices' overall bulk and mass.

as of 2015, one US dollar will buy around three gigabytes of NAND flash memory

Despite this, it seems perverse to make no attempt at all to keep filesizes as small as possible, as long as there are no (or limited) impacts upon the required computation at view-time.

Much like typesetting algorithms, few compression algorithms are designed with the minimisation of computation in mind. As a consequence of this, the result of compressing the data using some generic algorithm is likely to require significantly more computation to decompress than that of a carefully designed bespoke algorithm. The following section describes work towards such an algorithm.

4.2 IMPLEMENTATION

The most obvious saving that can be made is with the duplication of a document's textual content. The systems described in Chapters 2 and 3 both contain as many copies of the document text as there are pre-rendered galleys. In practice, there is no real need for more than one copy to be present in the file. Two approaches to this problem were considered.

4.2.1 *Pointers into the Source Text*

The first approach to be considered was to include the plaintext source of the document in its entirety, and for each rendering to contain only pointers to the relevant sections of text, instead of the words themselves. These pointers can either be absolute (in the form of a character offset from the start of the text) or logical (in the form *paragraph m, word n*). If the document text is to be included as a plaintext string, absolute pointers are easier to use than logical: logical pointers require either an auxiliary data structure to map the logical pointers to absolute ones, or for the document text to be stored in a format reflecting the logical structure, i. e. not in plain text.

The principal drawback of using this approach is that on occasion, the output of the linebreaking process does not precisely match the input: for example in the case where words are hyphenated (requiring one word to be broken into two parts, and the addition of a hyphen) or where certain glyphs may be substituted for others (such as with the use of ligatures, where a glyph pair or triplet may be replaced with a single glyph). For this reason, this approach was not considered further.

4.2.2 *Use of a Dictionary*

The second approach considered was the use of a dictionary to act as a lookup table for each word-level item produced by the linebreaking process.

A document's source text itself is likely to contain significant redundancy. In the 1930s, American linguist George Kingsley Zipf proposed *Zipf's law*,[Zip32] which (broadly) states that given a sizeable sample of text in any given language, the frequency of any word is inversely proportional to its rank in the frequency table. Stated another way, the most common word tends to appear twice as often as the second most common word, three times as often as the third most common word, and so on. There have been many subsequent studies on redundancy in English that have come to related conclusions, for example Claude Shannon estimated the redundancy of written English to be around 50% [Sha51, Hir88].

	Butterley		[PBB11]		Shakespeare		King James Bible		British National Corpus	
	103	the	233	the	23 197	the	62 099	the	5 375 304	the
	35	of	134	of	19 540	I	38 576	and	3 010 713	of
	33	and	116	to	18 263	and	34 445	of	2 541 227	to
	31	in	75	and	15 592	to	13 387	to	2 463 817	and
	25	to	75	a	15 507	of	12 735	And	2 015 815	a
	23	for	56	is	12 516	a	12 451	that	1 750 205	in
	23	a	52	be	10 825	my	12 167	in	949 677	that
	21	was	50	in	9 565	in	9 760	shall	924 763	is
	19	company	34	as	9 059	you	9 508	he	838 955	was
	17	Butterley	32	document	7 831	is	8 932	unto	816 193	for
total words	1 232		3 724		899 595		821 133		97 087 700	
unique words	628		1 436		67 107		33 446		1 733 032	

Table 1: Top 10 most frequent words in various documents. The total number of words and total unique words are also shown for each document. The data used to produce this table assumes a word is any contiguous block of (case-sensitive) non-whitespace characters; thus, “`and`” is distinct from “`And`”, and “`document`” is distinct from “`document.`”. The rationale behind this is that the data produced is more closely representative of a real dictionary of atomic “words” to be typeset in a malleable document.

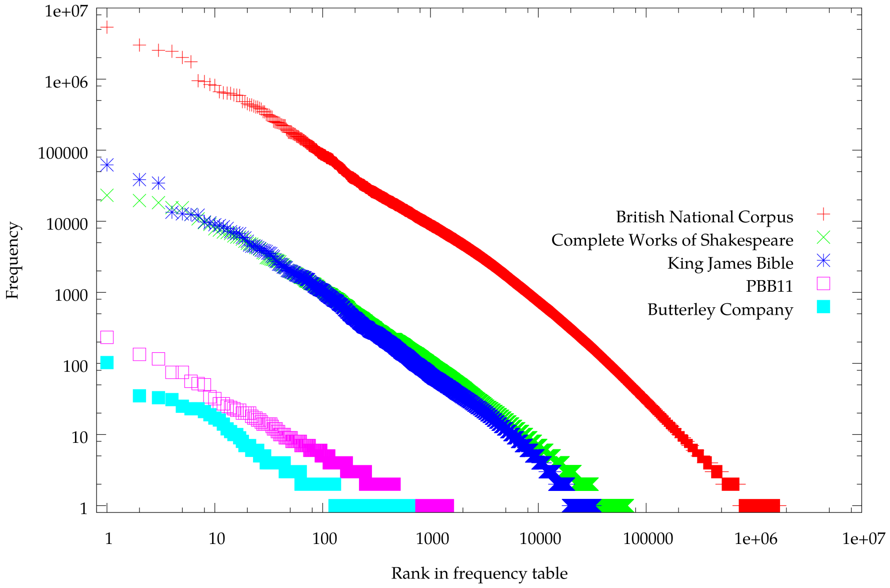


Figure 18: Word frequencies in various documents, plotted on a log-log scale. All of these documents, despite their varying lengths, appear to conform well with Zipf's Law, which manifests itself on a log-log scale as a straight line.

both Shakespeare and The Bible were obtained as plain text files from Project Gutenberg

Table 1 shows the ten most frequent words in four separate documents and for the British National Corpus[BNC07] as a whole.[1] The four documents used are the Wikipedia page for the *Butterley Company*[2], the author's 2011 paper *Reflowable Documents Composed from Pre-rendered Atomic Components*[PBB11], the Complete Works of Shakespeare, and the King James Version of the Bible. Figure 18 shows the word frequency data for the same documents plotted on a log-log scale. At the extremities, the data does not conform perfectly to Zipf's Law, though despite their hugely varying lengths, each document does display a clear Zipfian distribution.

This inherent redundancy in natural language can be exploited to produce a simple compression scheme through use of a dictionary. If, for example, the word "shall" appears multiple times in a document (in the King James Version of the Bible it appears 9760 times, and in the complete works of Shakespeare 3016 times) it is only stored once in the dictionary. As long as on average (i. e. over every occurrence of every word) each word's key is lexicographically shorter than the word itself, it can be guaranteed that some redundancy has been removed from the data.

[1] The British National Corpus describes itself as *"a 100 million word collection of samples of written and spoken language from a wide range of sources, designed to represent a wide cross-section of British English from the later part of the 20th century, both spoken and written"* and is available freely online at http://www.natcorp.ox.ac.uk/

[2] http://en.wikipedia.org/wiki/Butterley_Company

Figure 19: Cumulative distribution of word frequencies in various documents.

The HTML and JavaScript system described in Chapter 3 can be altered to use a dictionary-based lookup table with fairly few modifications. Firstly, since the data for the malleable document must be represented in JSON,[3] some means of including the dictionary must be devised. JSON supports two types of collection. The first is the *object*, which is defined formally as *an unordered set of name/value pairs*. This acts much like an associative array, though it does not guarantee the order of its elements. The second collection type supported by JSON is the *array*, defined as *an ordered collection of values*. Since both must be declared literally (in the forms `{"key1":"word1","key2":"word2"}` and `["word1","word2"]` respectively) rather than being populated programatically, using a plain array for the dictionary allows us to omit the keys from the dictionary itself, as they are implied by the order of the elements in the array. Additionally, since this forces the use of integers as keys, the Galley Structure Tree will not require the use of quote marks when the dictionary keys are referenced. If the keys were string values, each use of each key in the Galley Structure Tree would therefore necessitate two extra characters (e. g. `"key"` versus `401`).

It should also be noted that using integers as keys in JSON has different implications to using integers as keys in some more compact binary format. JSON is always stored in some textual encoding (perhaps ASCII, perhaps UTF-8), and there is no support for numeric representation in any base other than decimal. What might take up one 32 bit integer (i. e. 4 bytes) in a compiled language such as C might take as many as ten textual characters (10 bytes, assuming that

[3] see `http://www.json.org/` for full details

```
[
    [[0,982],[3.678,26],[3.678,93]],
    [[0,14],[2.682,1307],[2.682,558]],
    [[0,7],[3.668,557],[3.668,797],[3.668,226]],
    [[0,102],[4.338,9],[4.338,30]],
    [[0,112],[2.4,1013],[2.4,1068]],
    [[0,182],[2.4,1303],[2.4,2],[2.4,547]],
    [[0,308],[15.666,15],[15.666,1114]],
    [[0,177],[2.4,1173],[2.4,229],[2.4,733]],
    [[0,19],[7.336,81],[7.336,26],[7.336,143]],
    [[0,96],[7.116,33],[7.116,97],[7.116,16]],
    [[0,141],[9.444,0],[9.444,30],[9.444,1]],
    [[0,0],[8.78,89],[8.78,8],[8.78,11]],
    [[0,905],[10.008,2],[10.008,0],[10.008,66]],
    [[0,1125],[5.922,5],[5.922,34],[5.922,0],[5.922,1172]],
    [[0,1],[2.676,1053],[2.676,471]],
    [[0,3],[4.008,967],[4.008,112]],
    [[0,524],[10.338,19],[10.338,0]],
    [[0,126],[7.356,571],[7.356,1]],
    [[0,0],[6.896,197],[6.896,16],[6.896,18]],
    [[0,0],[2.4,9],[2.4,5],[2.4,691]],
    [[0,1249],[14.004,1046],[14.004,317]],
    [[0,5],[11.112,289],[11.112,2],[11.112,0]],
    [[0,273],[9.84,24],[9.84,859]],
    [[0,986],[11.34,144],[11.34,210]],
    [[0,263],[2.4,774]]
],
```

Listing 12: Excerpt from a JavaScript data file that uses position deltas in the Galley Structure Tree, representing one galley rendering of one paragraph. The first value in each pair is the position delta (in points) and the second is the dictionary key of the associated word.

whichever character encoding system is used represents low-ASCII characters with only one byte). Conversely, textual representation of integers can be more compact under certain conditions: namely, for values that use three or fewer characters, i. e. the numbers 0–999.

Referring back to Figure 18, to Figure 19, and to Zipf's law, it can be seen that even for extremely long documents, the number of words that are ranked in the top 1000 exceeds 60%, and so as long as the order of words in the dictionary is chosen carefully, using a textual representation of integers can be more compact than a naïve binary representation.

It was therefore decided to store the dictionary as an array, ordered such that the most frequently occurring words have the shortest keys.

4.2.3 *Further Compression Possibilities*

The techniques discussed thus far have focused mostly upon exploiting the inherent redundancy in natural language. A fairly large part of the

```
[["the",14.664],["of",9.996],["to",9.336],["and",17.328],["a",5.328],["
    is",8.004],["be",11.328],["in",9.336],["as",9.996],["document",47.3
    28],["that",18],["it",6.672],["page",22.656],["for",13.992],["are",
    14.652],["by",12],["on",12],["will",18.672],["which",29.328],["with
    ",21.336],["this",17.34],["The",18.66],["can",16.656],["an",11.328
    ],["or",9.996],["-",3.996],["eBook",31.332],["used",21.996],["PDF",
    22.008],["In",9.996],["layout",30],["have",22.656],["from",23.328
    ],["not",15.336],["at",8.664],["width",27.336],["This",21.336],["
    has",15.996],["then",20.664],["each",21.984],["was",18.66],["
    typesetting",52.668],["columns",40.668],["simply",32.676],["these",
    24.66],["text",18],["into",18.672],["hyphenation",59.328],["content
    ",35.328],["quality",33.336],["column",36],["lines",22.668],["only"
    ,21.336],["line",18],["ACM",27.336],["our",15.996],["its",11.34],["
    structure",41.988],["Document",49.992],["penalty",35.328],["between
    ",39.984],["galley",29.328],["order",25.32],["more",24.66],["COGs",
    30],["out",15.336],["end",17.328],["one",17.328],["use",15.996],["
    algorithm",46.668],["producing",48.66],["columns.",43.668],["
    galleys",33.996],["figure",28.656],["simple",32.004],["would",30],
```

Listing 13: Excerpt from the dictionary from a JavaScript data file that uses position deltas, where the width of each word is stored alongside the word itself. This is the dictionary from a rendering of [PBB11]: compare its ordering to that shown in Table 1 on page 53.

data contained within the Galley Structure Tree has been overlooked: the typesetting data itself.

All of the aforementioned encoding systems have used an absolute value for the x position of each word on each line; that is, each occurrence of each word has an associated value representing the required distance of its placement, in points, from the start of the line. An example of this can be seen in Listing 11 on page 43.

With a view to producing data that would be more easily compressible by a generic compression algorithm (that would perhaps be useful if HTTP compression or similar is used to transfer the document data to a device) it was decided to investigate a different approach to storing this data.

In any typeset document, most (if not all) occurrences of the same word will typeset identically upon the page. In particular, the amount of horizontal space reserved for a word will be the same for each occurrence of the word. Similarly, if a document's text is fully justified, the space between words on each individual line will be identical. If the document is left-justified, then each space between each pair of words on *every* line will be identical. This redundancy is present within all the previous encodings, but cannot be picked up by a generic compression algorithm, since it requires knowledge of the typesetting process. By separating the word widths from the spacing, this redundancy can be made more explicit, and therefore easier for a generic compression algorithm to take advantage of.

On the basis of the above observations, the decision was taken that the dictionary should be modified to store the width of each word alongside itself, and that the Galley Structure Tree should be modified so that each word to be typeset is now accompanied by the offset required from the end of the previous word (which will henceforth be referred to as *position deltas*) rather than the absolute offset required from the start of the line. This does of course necessitate two array lookups in the dictionary where previously there would have been one, but since array accesses run in constant time, this does not present a problem. Excerpts from a Galley Structure Tree and dictionary that use this encoding system are shown in Listings 12 and 13 respectively.

Further redundancy could be removed by exploiting the fact that words tend to be regularly spaced on each line. Whilst the encoding could be modified to allow only regular spacing of words, it was felt that this might be somewhat restrictive, and would detract from the appeal of the system as something that supports complex, arbitrary layouts.

Even without making further compression attempts beyond the encoding system shown in Listings 12 and 13 — the motivation for which, we must remember, was to produce an encoding that was more *compressible*, rather than more *compressed* — by pure chance, it turns out that even in its full form, this is the most compact representation yet devised!

4.2.4 *A Toy Example*

Here we see a very short document represented using each of the aforementioned compression schemes. Each of the following examples produces the same output document, which contains two galley renderings of one paragraph: “This is a short sentence”.

These will be rendered looking something like

This is a
short sen-
tence.

and

This is a short sentence.

The following is the original encoding, which uses absolute positioning and no dictionary:

```
{
   "galley_widths" : [72, 360],

   "paragraph_tree": [
      [
         [
            [
               [0, "This"], [38.538, "is"], [65.328, "a"]
            ],
            [
               [0, "short"], [51.276, "sen-"]
            ],
            [
               [0, "tence."]
            ]
         ],
         [
            [
               [0, "This"], [22.872, "is"], [33.9, "a"], [44.04, "short"
                   ], [71.964, "sentence."]
            ]
         ]
      ]
   ]
}
```

The following encoding uses a dictionary, which reduces redundancy by avoiding repetition of words. Since this document is so short, all its dictionary keys are the same length. For this reason, its corresponding version using an ordered dictionary (the next level of encoding devised) would be identical and is therefore omitted from this example.

```
{
   "galley_widths" : [72, 360],

   "paragraph_tree": [
      [
         [
            [
               [0, 0], [38.538, 1], [65.328, 2]
            ],
            [
               [0, 3], [51.276, 4]
            ],
            [
               [0, 5]
            ]
         ],
         [
            [
               [0, 0], [22.872, 1], [33.9, 2], [44.04, 3], [71.964, 6]
            ]
         ]
      ]
   ],
   "dict": ["This", "is", "a", "short", "sen-", "tence.", "sentence."]
}
```

This final encoding uses a dictionary that also contains the widths of words, so each word in the paragraph tree needs to store only its offset from the end of the preceding word.

```
//Ordered dictionary with position deltas
{
   "galley_widths" : [72, 360],

   "paragraph_tree": [
      [
         [
            [
               [0, 1], [19.134, 2], [19.134, 3]
            ],
            [
               [0, 0], [26.82, 6]
            ],
            [
               [0, 5]
            ]
         ],
         [
            [
               [0, 1], [3.468, 2], [3.468, 3], [3.468, 0], [3.468, 4]
            ]
         ]
      ]
   ],

   "dict": [
      ["short",     24.456],
      ["This",      19.404],
      ["is",         7.656],
      ["a",          6.672],
      ["sentence.", 46.368],
      ["tence.",    29.628],
      ["sen-",      20.724]
   ]
}
```

4.3 RESULTS

The following pages show the evolution of the encoding system, and how the filesizes vary according to the number of included galley renderings, for the same sample documents that are used for Figures 18 and 19:

- Figure 20 (page 63) shows the "original" encoding system described in Chapter 3, which does not make any attempt to minimise filesize.
- Figure 21 (page 64) shows the encoding system described in Section 4.2.2, using a dictionary ordered such that the earliest occurring words have the shortest keys. (The dictionary is therefore described as *unordered*.)
- Figure 22 (page 65) shows the encoding system described in Section 4.2.2, using a dictionary ordered such that the most frequently occurring words have the shortest keys. (The dictionary is therefore described as *ordered*.)
- Figure 23 (page 66) shows the encoding system described in Section 4.2.3, which not only uses a dictionary ordered such that the most frequently occurring words have the shortest keys, but also stores the width of each word in the dictionary, so that the Galley Structure Tree contains deltas rather than absolute positioning data.
- Figure 24 (page 67) shows a comparison of the filesizes produced by all encodings, and Figure 25 (page 68) shows the resultant filesizes when each rendering is further compressed with `gzip`.

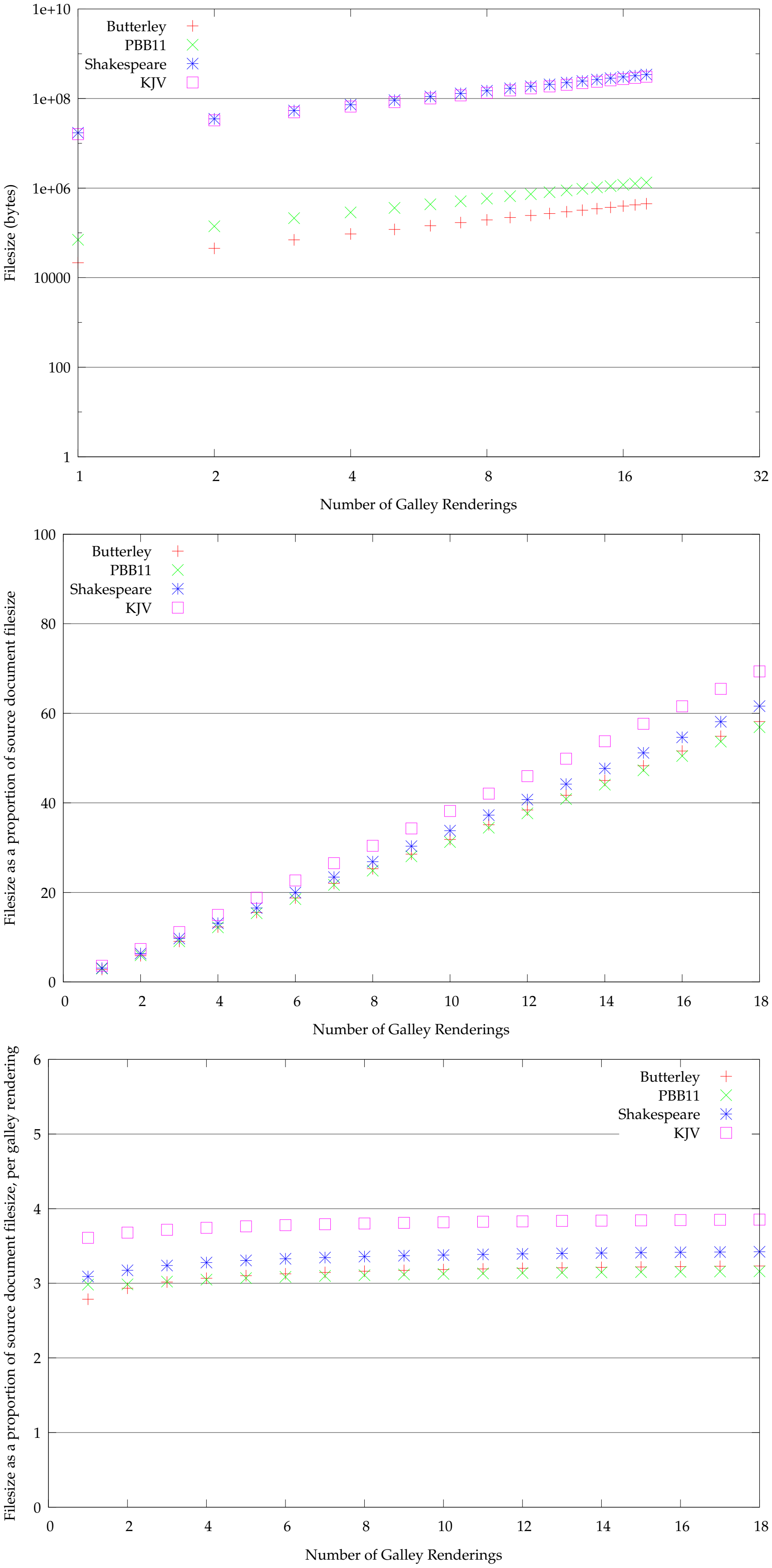


Figure 20: Filesizes of various documents, using the original encoding.

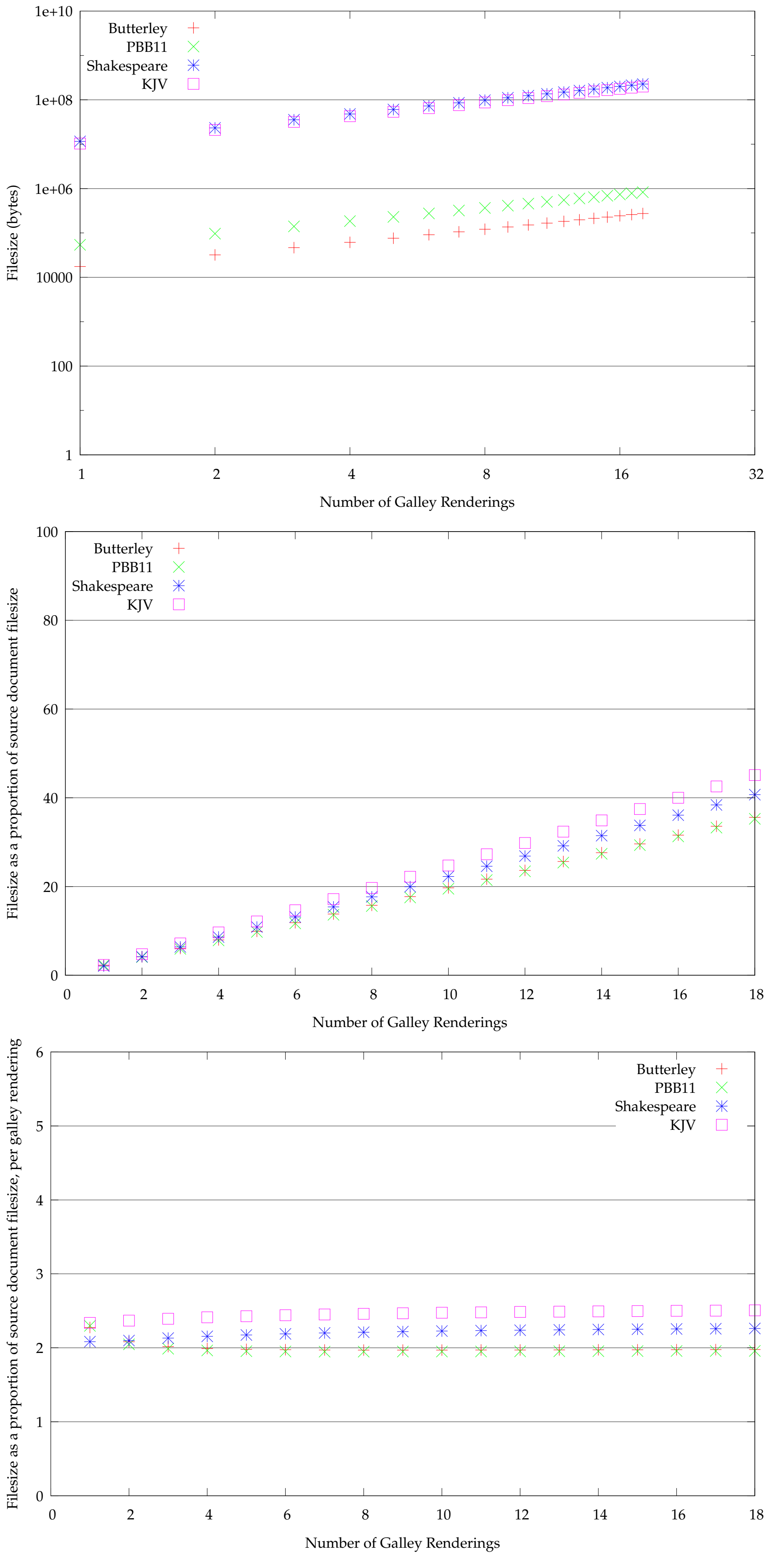


Figure 21: Filesizes of various documents, encoded using an unordered dictionary.

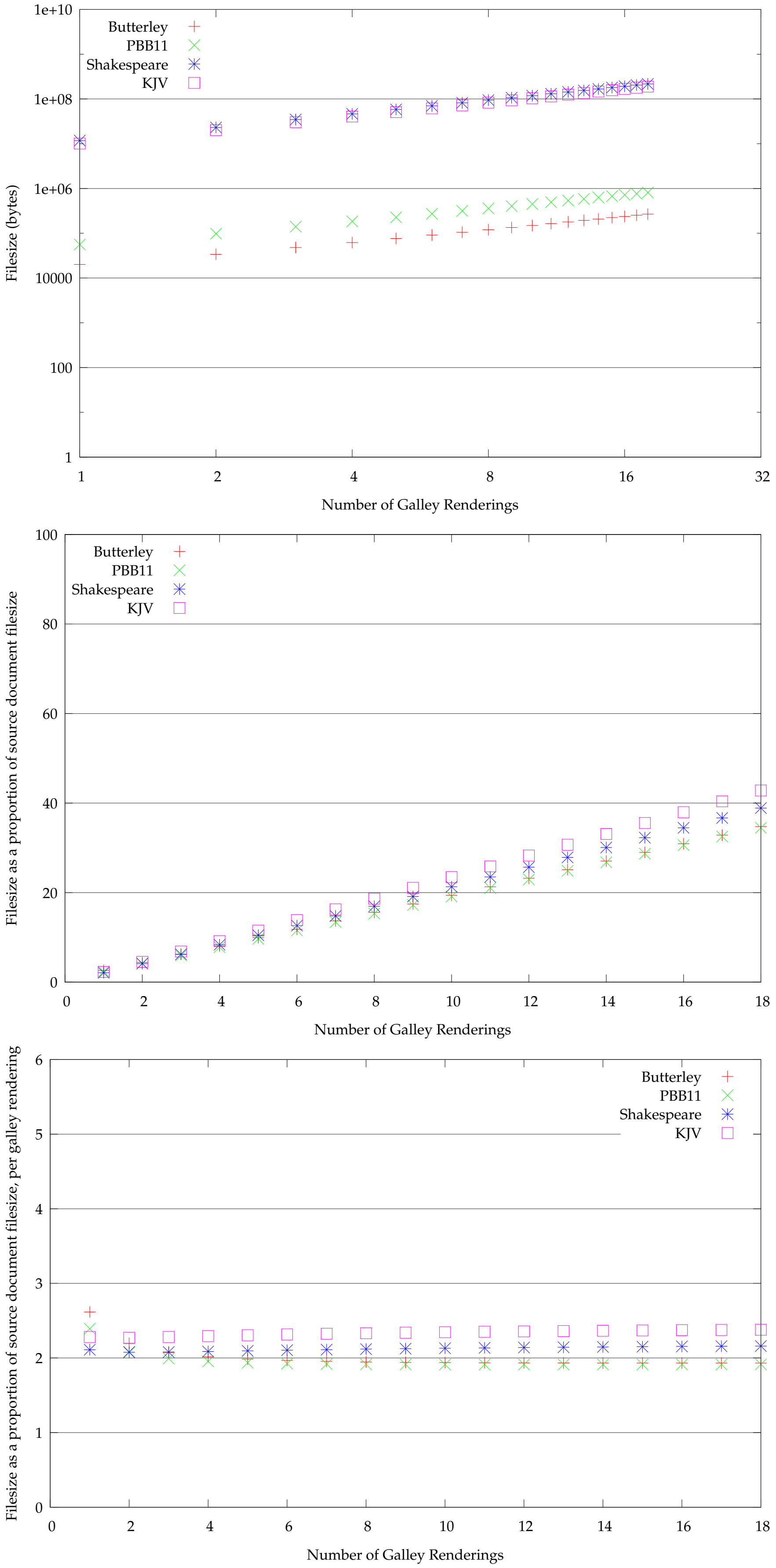


Figure 22: Filesizes of various documents, encoded using an ordered dictionary.

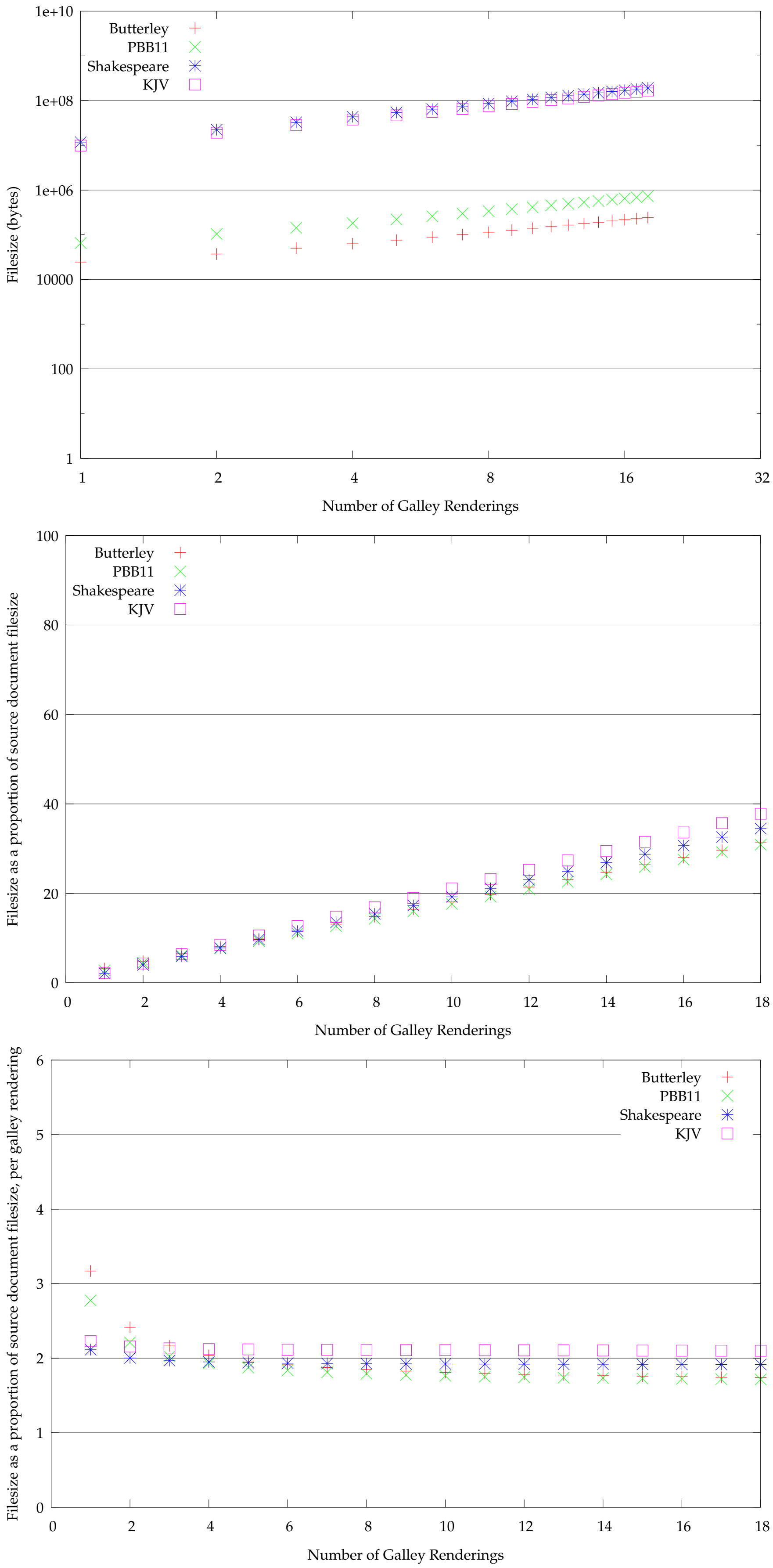


Figure 23: Filesizes of various documents, encoded using an ordered dictionary with word widths, and position deltas in the Galley Structure Tree.

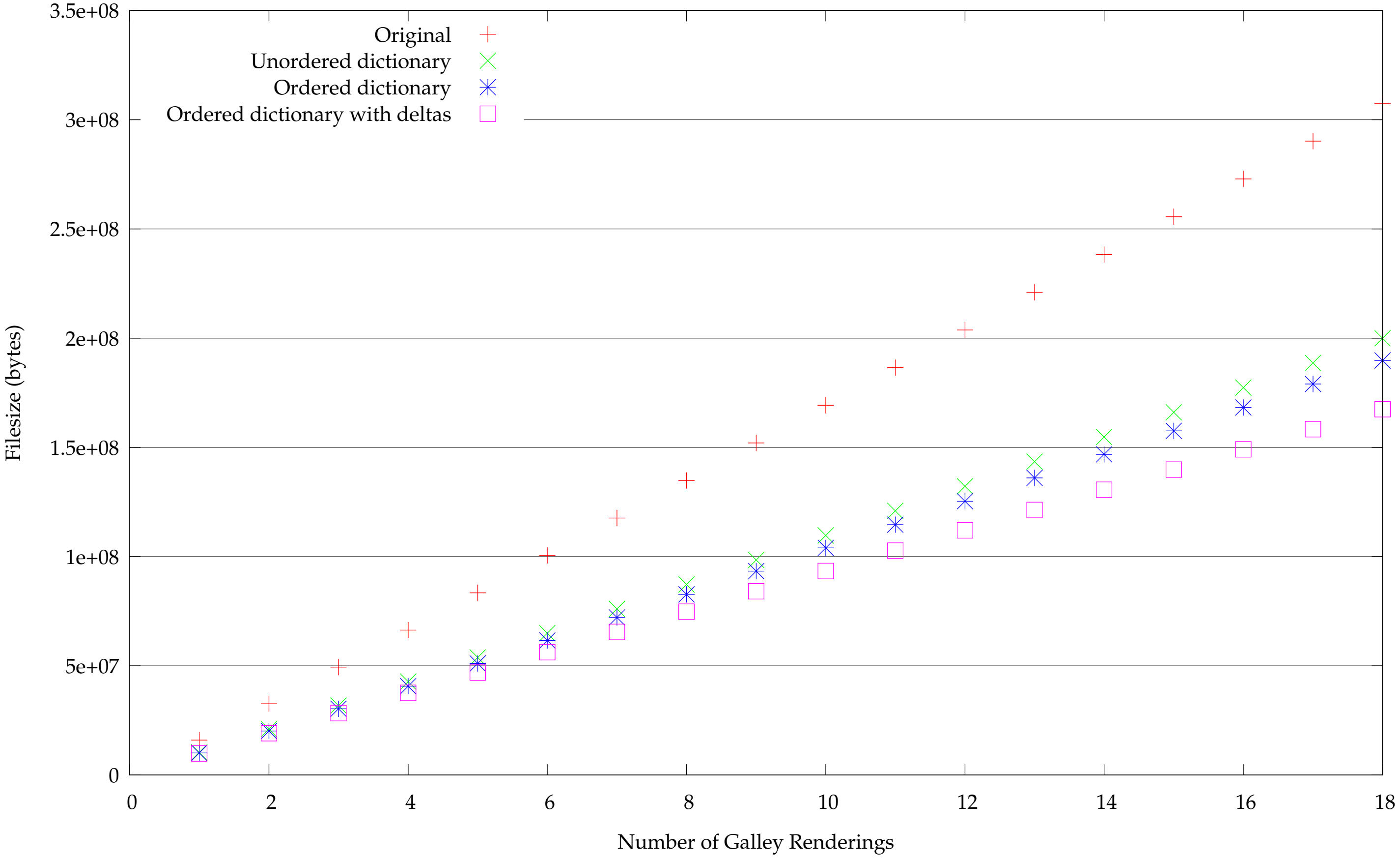


Figure 24: A comparison of filesizes produced by all encodings, using the King James Version of the Bible as a sample document.

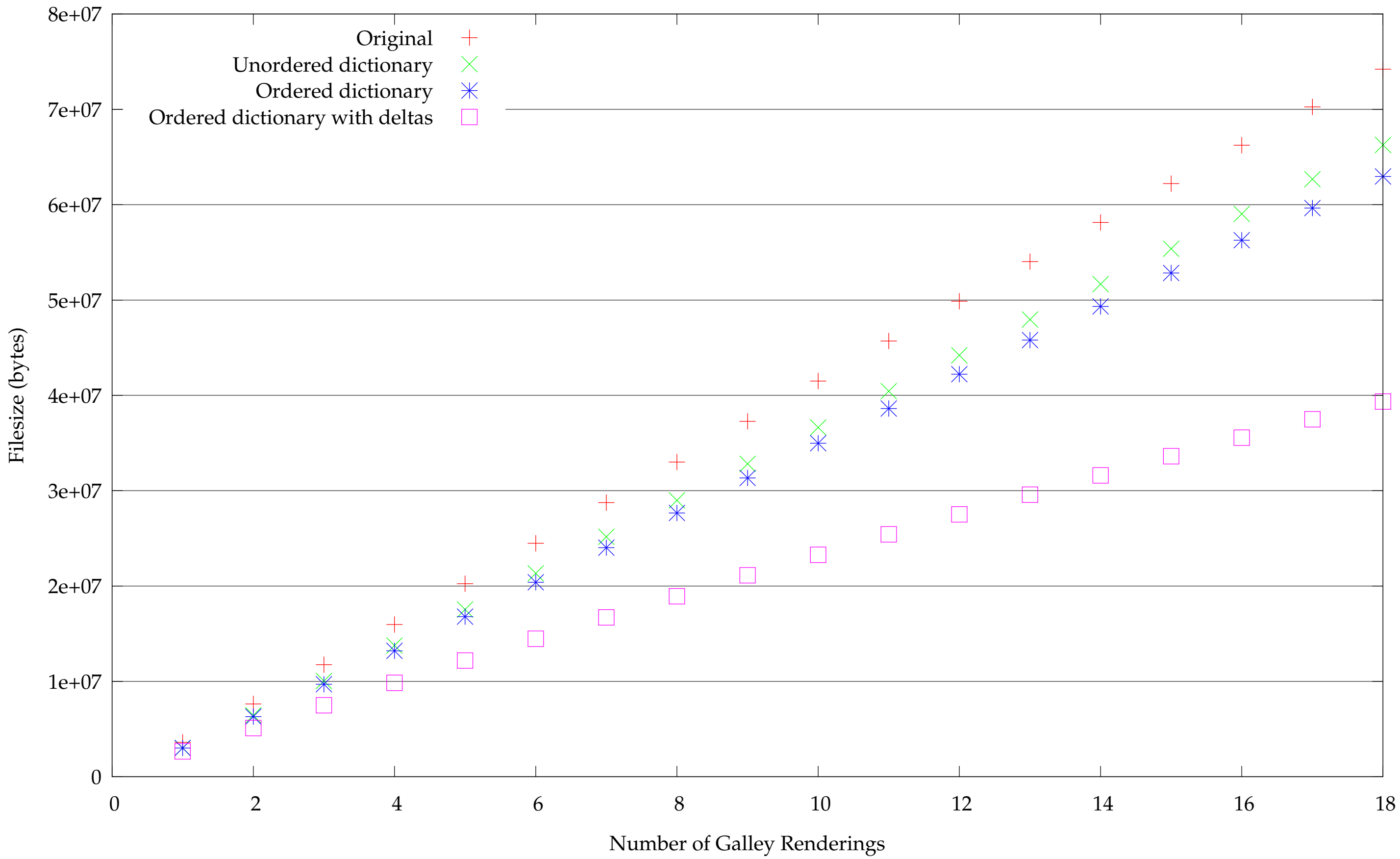


Figure 25: A comparison of filesizes of all encodings after gzipping, using the King James Version of the Bible as a sample document. Note the substantial improvement in compression of the encoding that uses position deltas, over the variants that use absolute positioning.

4.3.1 *Discussion*

It is fairly clear from the preceding graphs that so long as some thought is put in, a lot of redundancy can be squeezed out of the data, without the need to resort to aggressive compression methods that require significant computation during decompression.

Nevertheless, Figures 24 and 25 suggest that a significant amount of redundancy remains in each of the encoding schemes: on gzipping, the filesizes are reduced by some 66–75%. Some of this can be attributed to JSON's syntax, but it is likely that the blame lies more with the data itself. The dictionary itself is not particularly compact. No advantage is taken of words that share common substrings, nor of words that have identical widths. Algorithms that do take advantage of substrings, for example LZ77,[ZL77] tend not to have been designed with fast decompression in mind. It is vital that the complexity of any required decompression does not dominate that of the layout process itself.

A further manner in which the data could be compressed is to take advantage of the fact that the values of the position deltas are often repeated, particularly when lines have even spacing between words, which is in the vast majority of cases. This can be seen fairly clearly in Listing 12 on page 56. A second "dictionary" can be created in which to store these position deltas. Informal experimentation has suggested this would reduce filesizes by a further 10–15%, and gzipped filesizes by a further 5%.

4.4 SUMMARY

The result of the work in this chapter is a reasonably compact version of the document representation model developed in Chapter 3. As an example, a 7-galley malleable version of the King James Bible in the original representation was around 150 MB, and in the most compact representation around 57 MB. This is still considerably larger than the source document (around 4.3 MB as plain text) but does contain data that allows the content to be typeset to seven different widths. The graphs in Figure 23 show that each included rendering contributes approximately twice the size of the original plaintext source.

Since the compression used relies entirely on array lookups, the computational overhead for decompression is kept to a minimum. Dealing with moderately larger file sizes seems a reasonable price to pay.

Part III

ANALYSIS

5 ANALYSIS

We have now devised a fully-fledged document layout system that works across many platforms. In this chapter we discuss the layouts produced by the system both quantitatively and qualitatively, and put the system to the test in a user study.

5.1 QUANTITATIVE

Chapter 1 (and in particular Section 1.2) provides an overview of some of the operations required when laying out a document. This section examines this in further depth, and contrasts the operations required to view fixed and flowable documents against the operations required to view malleable documents.

5.1.1 *Fixed Document Formats*

Documents in a fixed format are rendered in a manner similar to the following:

```
parse and tokenise the document layout instructions;
foreach (layout instruction) {
    interpret and execute the instruction;
}
paint the results to the screen;
```

With the notable exception of PostScript, which is Turing complete, the majority of fixed document formats have only declarative layout instructions, and do not permit computation. This helps ensure that the document is always rendered identically [BBO07].

5.1.2 *Flowable Document Formats*

When a document in a flowable format is to be laid out, the process is as follows:

```
parse the source to identify flowable blocks;
foreach (flowable block) {
    apply a line-breaking algorithm;
}
paint the results to the screen;
```

This process generates information similar to that contained in fixed-format documents, which is then used to drive the painting of contents to the screen.

The line breaking algorithm can be as simple as or complex as desired. In most cases, the line-breaking algorithm used by flowable documents is reasonably simple, and will take a first fit approach. This will be of the form:

```
parse block to identify all possible breakpoints;
while (non-breakable items remain to be laid out) {
    place one item on the current line;
    if (there is not space for the next item) {
        adjust spacing between items to justify;
        move to a new line;
    }
}
```

As is noted in Section 1.2, first-fit algorithms do not generally result in well-typeset output. The simplest of these algorithms will not attempt to identify potential hyphenation points. More complex layout algorithms that search for "optimal" layouts usually require a considerable amount of backtracking and are thus more computationally demanding.

5.1.3 *Malleable Documents*

The layout algorithm for a malleable document is as follows. First, the penalties are calculated:

```
foreach (included galley rendering) {
    compute penalty for the rendering at current page width;
}
select the rendering with the minimum penalty;
```

Then, using the galley rendering that has the lowest penalty, the content is laid out onto the screen:

```
foreach (paragraph-level item in selected galley rendering) {
    foreach (line-level item in paragraph) {
        use precomputed data to place words;
    }
}
paint the results to the screen;
```

Crucially, a number of complex steps have been moved from view-time to compile-time (as they are for fixed-format documents) but without flowability being sacrificed:

- The source is already parsed into a form optimised for layout.

- The line-breaking has been entirely precomputed.

One step has been added: each included galley rendering must be examined once before layout, in order to ascertain its "penalty" for use. The penalty is based entirely on the width of the page, and the measure (width) of the galley rendering.

This penalty is calculated by taking the extra required horizontal whitespace (which can be envisaged as slack between columns) and weighting this to further penalise large numbers of columns. This weighting is achieved by multiplying by a smaller-than-linear function of the number of columns, such as a square root or logarithm.

Many low-power processors do not come with floating-point hardware as standard (for example the arm range of processors) which might suggest that these are poor choices of functions since they must be emulated using integer and bitwise operations only. This is not particularly important, for a number of reasons. Firstly, it has been shown previously[Lom03] that it is often possible to use mathematical analysis to find extremely good approximations for such functions. Secondly, the range of inputs to such a function would be limited to integers ranging from 1 to (in an extreme case) about 20, so the values could be pre-computed and stored in a lookup table. Thirdly, the penalty (and hence the root or logarithm) is calculated precisely once for each included galley rendering: in Sections 2.5 and 5.3 it is suggested that between three and ten galley renderings should be included in any one document. Given these facts, it is clear that there will be little impact from using such a function in the penalty calculation. In any case, as Don Knuth famously (and perhaps a little overdramatically) stated: "*premature optimization is the root of all evil*" [Knu74].

Most importantly, since the line-breaking algorithm has been moved to compile-time, there is no longer any requirement to limit its complexity. In fact, should the need (or desire) arise, the text layout can be hand-tuned, or *entirely hand-typeset*, with no consequences at view-time.

5.1.4 *Handling of Floats*

The grid-based layout system devised in Section 3.2.2 works in a similar manner to a first-fit line-breaking algorithm, in that it places elements on the page in order, in the first place they will fit. In the case of this system, each element is a floatable figure or line of text. Elements that are the same size as a single grid cell, such as lines of text set in the main point size, can simply be placed in the first empty slot in the current column, or the first empty slot in the next column, should there be no empty spaces. For the placement of elements that are larger than a single grid cell, there is some overhead required to step through the grid until a suitable position can be found. Once a position has been found, each grid cell that it overlaps must be marked as being reserved.

In the worst case, this algorithm does have a greater-than-linear time complexity. In practice, so long as the number of floats does not become excessive (which would cause the grid to be walked many times to search for suitably large gaps) the algorithm runs in linear time.

The placement of floats is subject to certain constraints: they must span integer multiples of columns, and can only be placed aligned to grid cells. This is very different to the model used for floats in HTML, whereby floats may be positioned arbitrarily, and text flowed around them. The result of this imposed "restrictiveness" on float placement (in comparison with that of HTML) is that the produced layouts are more regular. Each produced document as a whole fits together much better, as we shall see in Section 5.2.2.

5.2 QUALITATIVE

5.2.1 *Placement of Floats*

Since all text layout is precomputed, the only remaining concern is that the columns of text and floats are laid out in a pleasing manner.

Plass[Pla81] devised a system to perform optimal placement of floats within text, whereby float placement is penalised by the square of the distance from its intended position. He showed that this problem was NP-hard, but he also showed that a similar (but less "optimal") system using linear penalties could be made computationally tractable.

Brüggemann-Klein et al.[BKKW95] suggested that Plass's method is only optimal if one agrees with Plass's definition of "optimal". They proposed that a superior metric for float placement is to minimise the number of page turns that a reader must perform when reading the document from front to back. This is a desirable characteristic for a pagination algorithm that runs on an ebook reader, because page turns tend to be slow, particularly on devices with electronic paper displays. Unfortunately, the algorithm used runs in quadratic time, which limits its usefulness to this system.

In essence, the assumption that Plass's float placement algorithm produces the most optimal layouts may be slightly short-sighted: *other pagination schemes are available!* Clearly, using computationally complex algorithms such as those devised by Plass and Brüggemann-Klein et al. will have a significant impact on the demand for computation at view-time. As with many facets of the system described in this thesis, the float placement algorithm was chosen with efficiency in mind.

The float placement algorithm that has been developed for use with the malleable document system also aims to minimise the distance between the actual and intended positioning of floats: if a float can be placed directly at its intended position, then it will be, otherwise it will be placed in the next available space. (Figure 15 on page 48 demonstrates this process.)

Whilst this algorithm does not perform any lookahead or backtracking in order to place floats optimally, anecdotal evidence gained from using the system has shown that in most cases floats are placed directly in their intended position, or at the top of an adjacent column, and are rarely moved across page boundaries. Appendix B showcases some examples of layouts produced using this algorithm, alongside some comparative renderings of the same document, rendered in LaTeX and HTML. The output of the malleable document system looks very similar to that of LaTeX, and very different from that of the web browser.

As it stands, the algorithm does not avoid widowed or orphaned lines, nor single lines directly before or after floats. This can be seen clearly in the example in Appendix B.1.1 on the e-reader example (page 116) at the top of the fourth page, where the float has spanned both columns and taken up all the space on the page, with the exception of one line at the top of each column. It would be simple to add a constraint that states that single lines of text at the top or bottom of the page must never be allowed, which could be enforced by leaving extra lines blank and pushing the text forward, though it is possible that this may harm the balance of the page if it causes columns to have uneven lengths.

5.2.2 *Measures of Aesthetic Quality*

In their 2004 paper, Harrington et al.[HNJ+04] identified a number of aesthetic properties against which automated document layout systems may be measured. Many of these properties are inherently well satisfied by this system, due to its use of a grid to provide regular layout:

ALIGNMENT This property states that all content objects must have some commonality of alignment, based on their edges and/or centreline. Aligning content objects to a grid enforces this.

REGULARITY This property states that alignment positions must be regularly spaced — an inherent property of a grid.

BALANCE This property states that a page's visual weighting should ideally be around the centre of the page, and also that the page must be well balanced between left and right. Since the malleable document system produces pages with uniform density — there are no gaps left until the content has all been laid out — this property is well satisfied.

WHITESPACE FREE-FLOW This property states that whitespace should always be connected to the margins, and that there should never be islands of whitespace. The grid-based columnar layouts produced by the malleable document system ensure that any non-margin whitespace (for example the gaps between paragraphs or before and after floats) is always directly connected to the margin whitespace.

UNIFORMITY This property states that the visual density of the page should be consistent, i. e. that content objects should be distributed uniformly. The combination of the above measures (alignment, regularity, balance, and whitespace free-flow) mean that the page's visual density is inherently uniform.

Examining the grid layout float placement algorithm specified in Section 3.2.2 (in conjunction with the sample layouts shown in Appendix B) it can clearly be observed that all these characteristic properties are fulfilled by the malleable document system.

5.3 USER STUDY

In order to establish the most appropriate range of galleys to include in a malleable document, as discussed in Section 2.5, a user study was conducted. This was also used as a chance to get some general feedback about the malleable document system from a large pool of real-world users.

5.3.1 *Participants*

The study was carried out entirely online. Participants were recruited via email circulated around the researchers mailing list at the School of Computer Science, and via a number of posts on the web. No personal data was collected from any participants, though due to the recruitment method, it is a reasonable assumption to make that the majority of the participants had prior experience in reading electronic documents on a screen. In all, the study had 41 participants.

5.3.2 *Methodology*

Obtaining data that can be interpreted quantitatively from a process that is inherently qualitative can be a challenge. It can be difficult to turn *How good did you think this was?* and *Which one of these is better?* into something from which statistical significance can be taken. With enough respondents and the use of some cunning, this can certainly be achieved.

	inches	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	5.5	6.0
	characters	22	28	35	42	49	56	63	70	77	84
A	315 KB	✓	✓		✓						
B	392 KB	✓	✓		✓		✓				
C	538 KB	✓	✓		✓		✓		✓		✓
D	549 KB	✓	✓	✓	✓	✓	✓				
E	839 KB	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
F	386 KB	✓	✓				✓				✓

Table 2: This table shows both the pool of galley widths that were used in the generation of malleable document instances A–F, which renderings each instance contained, and the filesize of each instance. The number of characters corresponding to the width in inches has been taken from a lookup table in *The Elements of Typographic Style*[Bri08]. These values are for 12 point Times Roman.

In this user study, each participant was given a malleable document to read, and then asked for their feedback. Each participant was randomly assigned into one of six groups, labelled from A to F. Participants were not made aware that they had been assigned to a group. Each group was presented with a separate instance of a malleable document. These all had the same textual and graphical content, but contained different ranges of galley renderings.

As mentioned in Section 2.5, according a consensus among professional typographers is that lines of text should not normally be less than 30 characters wide, nor wider than 75, and that 40–50 characters is reasonable for multi-column work. The pool of galley widths from which each malleable document instance's included renderings were chosen is shown in Table 2. These were chosen to extend slightly beyond this range, whilst providing good coverage across it. The table also shows which of these renderings was included in each group's document.

All documents contained the narrowest two galley renderings. This was to ensure that it would be possible to display the document on an extremely narrow screen and/or with a high scaling factor.

Document A was deliberately designed to provide a poor user experience. It had only three galleys, and only one of these was within the range specified above. Documents B and D both contained galleys up to 56 characters in width, and documents C, E and F contained galleys up to 84 characters in width. Documents B and C contained a coarser range of galleys (and hence fewer) than documents D and E. Document F contained the coarsest range of galleys. These ranges and coarsenesses were chosen in order to assess how their changes affect perceived user experience.

5.3.3 *Preamble*

At the start of the study, each user was presented with the following:

```
This study is designed to evaluate a variety of document
layouts, and the way these layouts are interacted with by
users. You will be asked to read a short document with a
particular layout, and then fill in a questionnaire about
your reading experience. Don’t worry, this is not a
comprehension exercise! You may use any device with a
modern web browser (eg desktop, laptop, tablet, mobile
phone) though it is preferable that you use a mobile phone
or tablet, since this is primarily aimed at portable
devices with small screens. For this reason, if you’re
using a laptop or desktop, you may find it helpful if your
web browser window is not maximised.

Very shortly, you’ll be given some text to read (taken from
the Wikipedia article on The Great Fire of London) that
will be customised to fit the screen you’re reading it on.
Feel free to try changing your browser window size (or
screen orientation if you’re using a phone or tablet). If
you don’t want to read the whole document, it’s fine to
stop once you feel you have looked through it enough to be
able to answer some questions on the document’s layout.

Before you begin, here are some quick instructions:

If you’re using a laptop/desktop computer with a keyboard:
 *  Use the up/down arrow keys to increase/decrease
    the font size
 *  Use the right/left arrow keys to turn to the
    next/previous page

If you’re using a device with a touchscreen (phone/tablet
etc):
 *  Swipe up/down on the screen to decrease/increase the
    font size
 *  Swipe left/right turn to the next/previous page
```

5.3.4 *Questions*

Once each user had read the preamble, and had had enough of reading about The Great Fire of London, they were asked for their feedback in the following manner:

```
Please rate the following statements based on how closely
```

```
you agree or disagree with them. For each, choose a number
from 0 to 10, where 0 is "I strongly disagree", 5 is
neutral, and 10 is "I strongly agree".

Q1. The document layouts produced were well-adapted to fit
    my screen.

Q2. I could adequately customise the document layout to
    suit my personal reading preferences.

Q3. I would preferentially choose a document layout system
    like this to read long text-based web pages.

Q4. I would preferentially choose a document layout system
    like this to read PDFs.

Q5. I would preferentially choose a document layout system
    like this to read ebooks.
```

Questions 1 and 2 were designed to assess overall impressions of the system, and questions 3–5 to ascertain whether users would feel comfortable using a similar system to read documents in different contexts. Participants were also given the opportunity to provide written feedback.

5.3.5 *Discussion of Results*

Table 3 and Figure 26 show the aggregated response data for each group. It is clear that, especially for questions 1 and 2, renderings D and E are by far and away the most consistently highly rated. These were the two documents that had the least coarse range of galley renderings, which suggests that, as seems intuitive, the inclusion of a greater number of galley renderings within a given range provides a better user experience. It is also clear that the smaller maximum galley width used in rendering D has not had a vast adverse affect on its user rating.

As shown in Table 2, rendering D's filesize is approximately 65% of that of rendering E. Figure 27 shows the filesizes of each rendering plotted against their mean scores for questions 1 and 2. This quite neatly illustrates each rendering's tradeoff between filesize and perceived quality, and it is easy to identify that rendering D here makes the tradeoff work best.

As to questions 3, 4, and 5 (*"I would use this for {web pages, PDFs, ebooks}"*) there does not seem to be much consensus between any of the renderings. Averaging over all responses, the results are web pages: 5.8, PDFS: 6.7, and ebooks: 6.9. This does show a weak preference for the

group	range (inches)	step (inches)	n	respondents	q1	q2	q3	q4	q5
A	1.5, $2-3$	1.0	3	9	6.8	6.3	5.7	7.1	6.8
B	1.5, $2-4$	1.0	4	6	7.7	6.7	6.0	6.3	5.8
C	1.5, $2-6$	1.0	6	8	7.1	6.6	4.5	6.0	5.8
D	1.5, $2-4$	0.5	6	7	8.1	7.7	5.9	5.9	7.4
E	1.5, $2-6$	0.5	10	5	8.6	8.0	8.6	8.0	8.4
F	1.5, $2-6$	2.0	4	6	6.5	7.2	5.3	7.0	8.0

Table 3: A summary of the data obtained from the user study. For each group A–F, the included renderings are shown (columns *range*, *step* and *n*, respectively the narrowest and widest galleys, the size of steps between them, and the total number of included galleys). The mean value of the responses for questions 1–5 (detailed in Section 5.3.4) is shown for each group. Figure 26 shows a plot of this data in greater detail.

malleable document system for all three, with a stronger preference for PDFs and ebooks.

5.3.6 *User Comments*

Some comments from users follow:

> "Initially the unconventional method of adjusting the type size by swiping vertically seemed out of place and a little jarring when it is standard practice to swipe vertically to scroll webpages etc on an iPhone - swiping to turn pages, typically reserved for ebooks does work well in this application of the gesture. However, without the instructional prompt before reading the article I would have probably taken some extra attempts to realise swiping ebook-style would progress through the article. The way the text flowed into columns etc while adjusting the size was fluid and ensured any preference could easily be achieved. Overall I liked the way it worked."

> "The thing that put me off the viewer most was the fact that it enforced a page model on a continuous flow document and in so doing broke the browser's native scroll ability. This meant that I couldn't use the mouse wheel to scroll, and I couldn't jump to arbitrary points in the document using the scrollbar. I would have much preferred a single continuous column that my browser could scroll through normally, and this is also true if I were browsing on a mobile device."

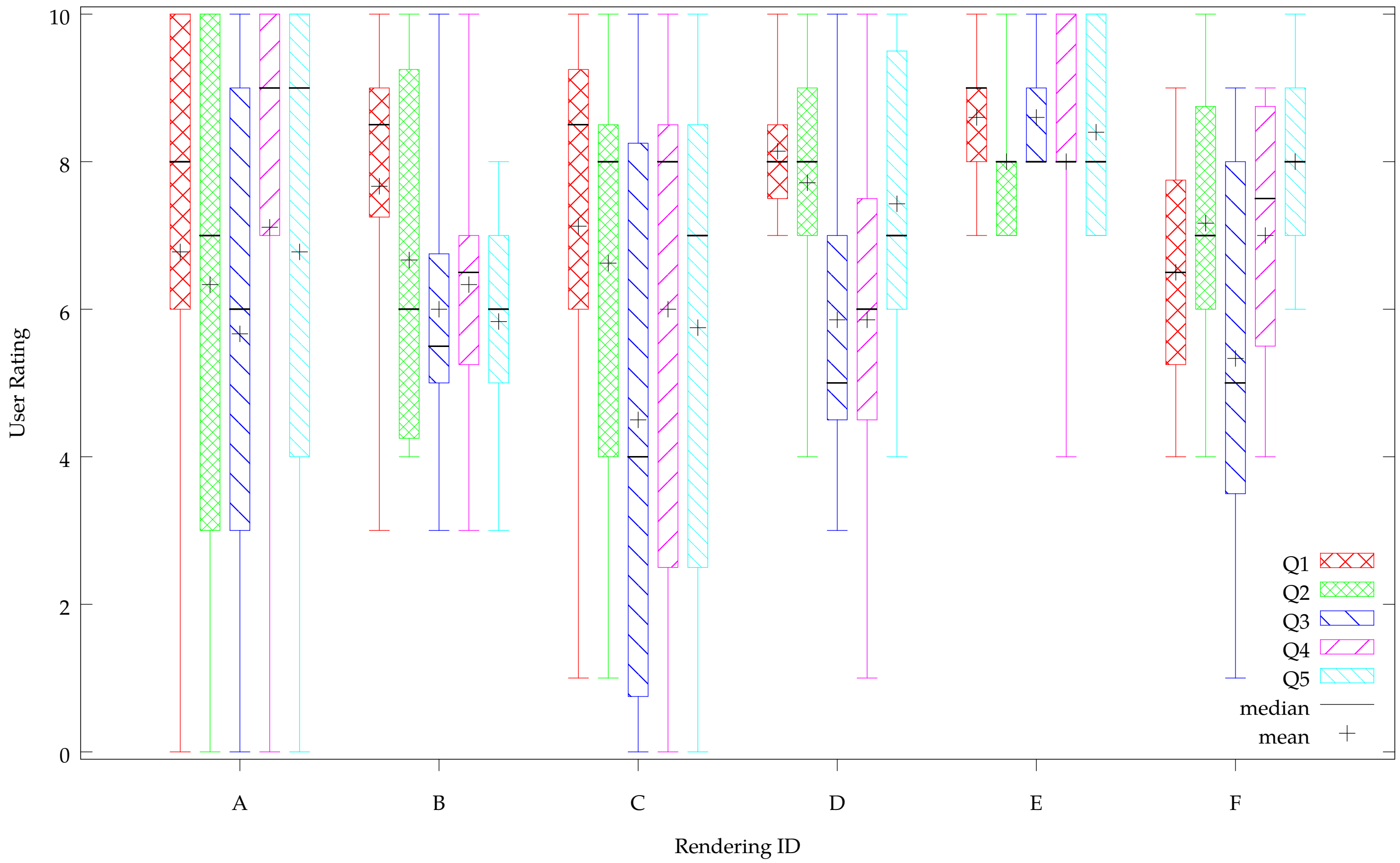


Figure 26: This plot shows the distributions of responses to each survey question for each group of respondents. The boxes mark the interquartile ranges, and the whiskers show the full range. The mean and median values are also shown (as indicated in the key). Questions 1 and 2 are of the most import: respectively, these assess whether the user thought the layouts fitted well to their screen, and whether the user thought they could customise the layout to suit their reading preferences.

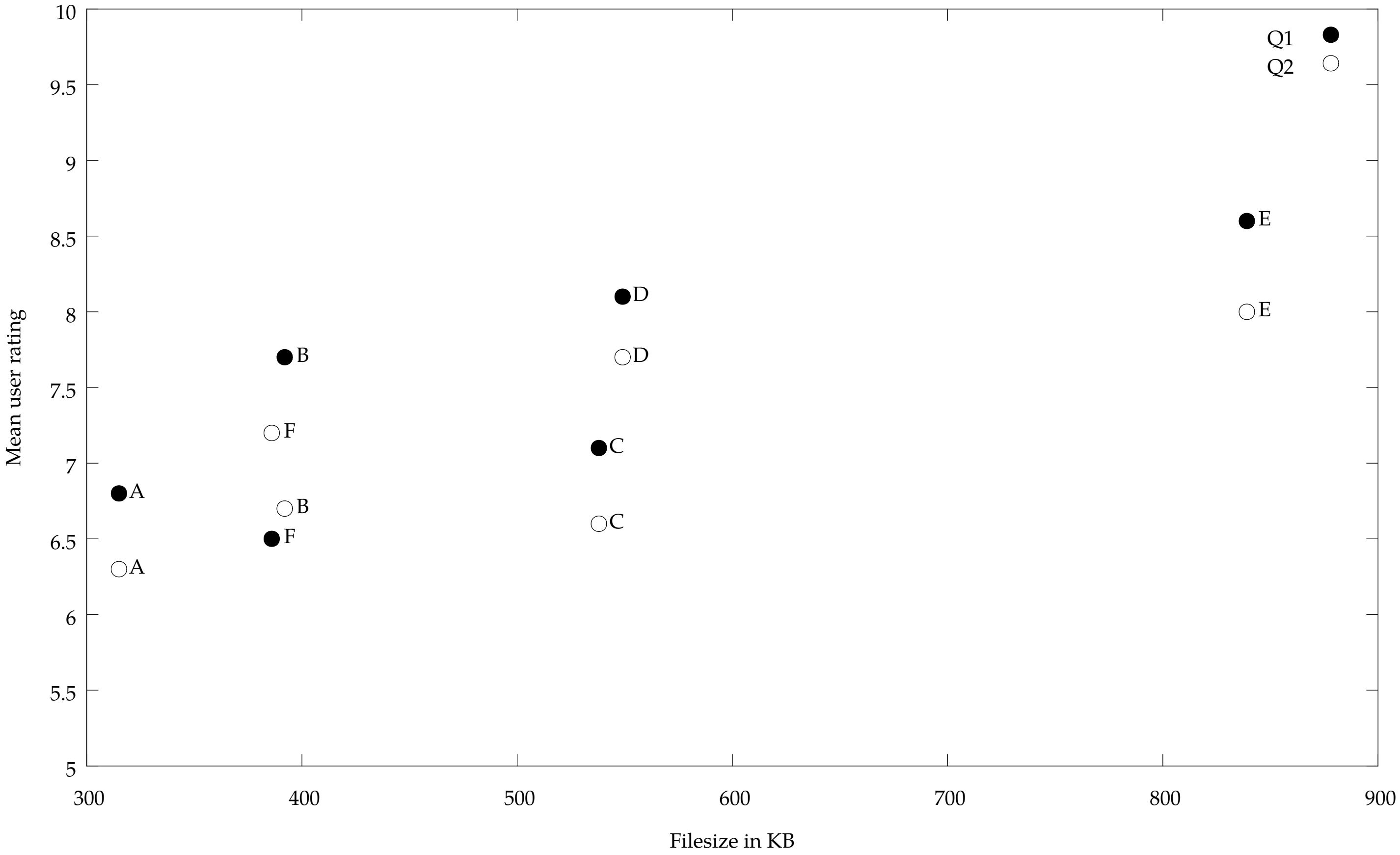


Figure 27: This plot shows the mean of each group's responses to questions 1 and 2 against the filesize of that group's document. Although group E scored highest, group D's scores are not far behind, and its filesize is some 65% of group E's.

> "In general it seems to be working fine. My main issue is that on each change (orientation/font size/etc...) it changes amount of content being displayed. Which means that if I switch devices or orientation on device or even resize my browser - it will be really hard to find place where to continue reading."

> The only issue is with long documents if you wish to go back to a previous page or section then you have to keep flipping through the pages one by one. The ability to jump to a particular section via a list of contents would be helful.

5.4 SUMMARY

The malleable document system devised in this thesis was designed to be used for linear documents whose content is primarily text. Examples of such documents would be novels and scientific papers, but not reference books or graphic-heavy documents such as comics or children's picture books.

The layouts produced by this system are visually very similar to those of both newspapers and scientific papers, and can be flowed to fit virtually any page size. For smaller screen sizes, where single- or double-column spreads occur, the layouts closely resemble those of physical books and magazines.

6 FINAL THOUGHTS

The intention at the outset of this project was to devise an efficient method to provide flowability to documents whilst maintaining their typographic quality — to investigate the middle ground between fixed formats and flowable formats. This area, as far as the author is aware, has previously been left unexplored.

Much research into automated layouts[JMPS96, Gol02, PHOF03, BHW09] has been geared towards static page sizes, and does not provide support for reflow at view-time. Other research into automated layouts that is designed with view-time reflow in mind[JLS+03, SDJ+08] does so at the expense of typesetting: generally, the text must be considered completely flowable in order to fit into the layouts devised by the systems.

When text is to be typeset, the choice must be made between *computationally cheap* and *typographically good*. The fact that both computational cheapness and typographical quality are desirable characteristics for ebook readers suggests that ebook readers are not the correct place to compute text layout.

6.1 CONTRIBUTION

The system devised in this thesis, whereby line breaking is precomputed but the final binding of layout is delayed until view-time, removes the need to make any compromises on typographical quality. Precomputing multiple variants of line breaking, at differing widths, allows the text to fit to a multitude of screen sizes, by displaying one or more columns of whichever galley width best fits the page.

Consideration has been given to compression of the resultant documents: 'pre-compression' provided by squeezing as much redundancy out of the data as possible whilst still in its uncompressed form that allows constant-time data access, and for 'post-compression', by making the uncompressed form as amenable as possible to packing by some generic compression algorithm, which can be used during file transfers or long-term storage.

Since the line breaking is precomputed, the display device does not need any knowledge of the algorithm: the only guarantee that is needed is that the device must be able to correctly interpret the rendering instructions. Because of this, each individual malleable document can use any text rendering algorithm — the system was deliberately designed to be modular, so that the text rendering algorithm can easily be changed.

The Malleable Document system shows real promise. It can represent documents whose text has been typeset by any manner of exotic line breaker. It can produce layouts that are essentially indistinguishable from those produced by professional typesetting systems. It can adapt its output to a vast range of screen sizes, with minimal computation. This is the beginning of an exciting new era for electronic document representation.

6.2 SYSTEM EXTENSIONS

The implementation of the Malleable Document presented in this thesis should be considered only a prototype. There are numerous areas in which it could be modified: some are reasonably straightforward changes, while others are fairly major overhauls.

6.2.1 *Improved Support for Floats*

The system described in chapter 3 provides only very basic support for floats. A particular limitation is that unlike text, each float has only one rendering, which must be scaled up or down as required, to fit across multiples of columns. Whilst for image-based figures or illustrations, this is probably already the desired behaviour, other types of floats, such as tables or code listings, would almost certainly benefit from the inclusion of multiple width renderings, with the choice of which rendering to display to be made at view-time. As with the text layout, these renderings could be hand-tuned, or produced by some automated process.

6.2.2 *Improved Vertical Layout*

As mentioned in Chapter 5, a naïve algorithm is used for vertical layout, which makes no attempt to avoid orphaned or widowed lines. Kernighan and Van Wyk[KVW89] described the solution to a similar problem, designed at improving the output of the *troff* typesetting package, providing better methods of pagination, figure placement, footnote handling, and so on. Care must be taken, of course, that any extensions do not impact upon the computational demands of the system as a whole, but certainly, improvements can be made.

6.2.3 *Postponing Layout*

Precisely when the precomputed aspects of layout should be precomputed is an interesting question. There are three key points where this could be performed: at the time when the document is created; directly before the document is transferred to an ebook reader device;

and directly after the document has been transferred to an ebook reader device. Each offers its own advantages.

If the precomputation is performed at creation time, the publisher and author have full control over all renderings, which is certainly beneficial from the point of view of quality control.

The second juncture at which the the precomputation can be performed is directly before the document is transferred to the reader device. This would allow knowledge of the reader device to be taken into account, allowing the output to be more closely tailored to the device. It is envisaged that such a system would utilise an intermediary program to transparently perform the text layout as the document is transferred to the device, using a similar model to that of Calibre or iTunes, or the model Amazon uses, whereby users can email documents to their Kindle, which then arrive in Kindle format.

The last point at which the precomputation can be performed is on the device itself. At first glance this approach might seem counterproductive, and in conflict with the underlying philosophy of this thesis. Instead of precomputing several variants all at once, the system can be redesigned so that it only computes text layouts when necessary, but, crucially, caches the layout to disk for later reuse. Though the layout would be performed by the ebook reader device itself, it would only ever be calculated once for each rendering of the document, and not each time the document is displayed.

6.2.4 *Moving Nearer to the Metal*

The two implementations of the malleable document system that were developed in this thesis in Chapters 2 and 3 were built upon PDF and HTML respectively. Both of these require the layout instructions to be parsed and interpreted, and rely upon third-party systems (e. g. Acrobat and WebKit/Gecko) to display their content.

It has been shown previously[Bag10] that it is possible to compile PDF to machine code, which can then be run natively on the processor of the display device. Using a method such as this would dispense with all the unneeded overhead associated with using an off-the-shelf system for display, and in general would be likely to run faster. Clearly this would require the output to be tailored to each device (or class of comparable devices), but as is discussed in the previous section, this is perfectly feasible.

6.3 OPEN RESEARCH QUESTIONS

More generally, the development of the malleable document system has highlighted a number of questions that suggest areas for future research:

- Pre-rendering text layout necessitates that the typeface is chosen ahead of time. Should each document be rendered in a certain set of typefaces, for example for accessibility purposes? Is there a particular subset of typefaces or classes of typeface that provides maximum flexibility, both in terms of user preference and accessibility?
- In a similar vein, what should be the sampling frequency within the range of included galley renderings? Should this be linear, or would some other sampling frequency that attempts to avoid simple multiples result in a smoother "sawtooth" penalty graph (such as that in Figure 8 on page 22)?
- Is there any benefit in attempting to coordinate breakpoints between different galley renderings, to allow switching between different width galley renderings, for example to support floats that span half-columns? Would this cause the typography to suffer, or would it provide more benefits by giving more flexibility to its layouts?
- Should some limited computation be allowed at view-time, for example to adjust letter-spacing or glyph widths in order to provide a better fit for a galley? How much should be allowed before the benefits are outweighed by the computation itself?
- Many documents — particularly academic work — contain cross-references and footnotes. What is the best way to handle these?
- Would use of Just In Time compilation on display devices (i. e. computing but then caching layouts) positively or negatively affect user experience?

6.4 CONCLUDING REMARKS

Linear, primarily text-based documents, such as novels, newspaper articles, and scientific papers, make up a large proportion of published, typeset documents. These documents typically have their text rendered to fit rectangular apertures, and so long as this text is well-typeset, its precise final layout is not of enormous importance.

It is the documents that fall into this category, in their uncountable millions, that benefit most from the system described in this thesis.

Ebook readers are beginning to reach critical mass. We must ensure that we do not stumble blindly into a future where substandard typography becomes an accepted norm.

REFERENCES

[Ado98] Adobe Systems Incorporated. *Adobe Font Metrics File Format Specification*. 1998.

[Ado01] Adobe Systems Incorporated. *Portable Document Format Reference Manual*. Addison-Wesley, third edition, 2001.

[Bag04] Steven Bagley. *A Component-Based Model for Creating and Manipulating Digital Documents*. PhD thesis, University of Nottingham, 2004.

[Bag06] Steven R. Bagley. COG extractor. In *Proceedings of the 2006 ACM Symposium on Document Engineering*, page 31. ACM Press, 2006.

[Bag10] Steven R. Bagley. Lessons from the dragon: compiling PDF to machine code. In *Proceedings of the 10th ACM symposium on Document engineering*, DocEng '10, pages 65–68, New York, NY, USA, 2010. ACM.

[BB05] Steven R. Bagley and David F. Brailsford. Demo abstract: The COG scrapbook. In *Proceedings of the 2005 ACM Symposium on Document Engineering*, pages 233–234. ACM Press, 2005.

[BBH03] Steven R. Bagley, David F. Brailsford, and Matthew R. B. Hardy. Creating reusable well-structured PDF as a sequence of component object graphic (COG) elements. In *Proceedings of the 2003 ACM Symposium on Document Engineering*, pages 58–67. ACM Press, 2003.

[BBO07] Steven R. Bagley, David F. Brailsford, and James A. Ollis. Extracting reusable document components for variable data printing. In *Proceedings of the 2007 ACM Symposium on Document Engineering*, pages 44–52, New York, NY, USA, 2007. ACM.

[BHW09] Helen Y. Balinsky, Jonathan R. Howes, and Anthony J. Wiley. Aesthetically-driven layout engine. In *Proceedings of the 2009 ACM Symposium on Document Engineering*, pages 119–122, 2009.

[BKKW95] Anne Brüggemann-Klein, Rolf Klein, and Stefan Wohlfeil. Pagination reconsidered, 1995.

[BMM$^+$09] Cameron Braganza, Kim Marriott, Peter Moulder, Michael Wybrow, and Tim Dwyer. Scrolling behaviour with single-

and multi-column layout. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 831–840, New York, NY, USA, 2009. ACM.

[BNC07] BNC Consortium. *The British National Corpus, version 3 (BNC XML Edition)*. Oxford University Computing Services on behalf of the BNC Consortium, 2007.

[Bri08] Robert Bringhurst. *The Elements of Typographic Style (v 3.2)*. Hartley & Marks, 2008.

[Col91] David Collier. *Collier's Rules for Desktop Design and Typography*. Addison-Wesley, 1991.

[EG92] D Eppstein and Z Galil. Sparse dynamic programming II: Convex and concave cost functions. *J. ACM*, 39(3):546–567, 1992.

[Gol02] Eldan Goldenberg. Automatic layout of variable-content print data. Master's thesis, University of Sussex, 2002.

[hex12] hexus.net. Kindle book sales overtake print at Amazon. http://hexus.net/mobile/news/e-readers/43365-kindle-book-sales-overtake-print-amazon/, August 2012.

[Hil99] Bill Hill. The magic of reading. Technical report, Microsoft, 1999.

[Hir88] William Hirst, editor. *The Making of Cognitive Science: Essays in Honor of George Armitage Miller*. Cambridge University Press, 1988.

[HL87] D S Hirschberg and L L Larmore. The least weight subsequence problem. *SIAM J. Comput.*, 16(4):628–638, 1987.

[HLM09] Nathan Hurst, Wilmot Li, and Kim Marriott. Review of automatic document formatting. In *Proceedings of the 2009 ACM Symposium on Document Engineering*, 2009.

[HNJ+04] Steven J Harrington, J. Fernando Naveda, Rhys Price Jones, Paul Roetling, and Nishant Thakkar. Aesthetic measures for automated document layout. In *Proceedings of the 2004 ACM Symposium on Document Engineering*, pages 109–111. ACM Press, 2004.

[IDP11] IDPF. *EPUB 3.0 Specification*. International Digital Publishing Forum, 2011.

[JLS+03] Charles Jacobs, Wilmot Li, Evan Schrier, David Bargeron, and David Salesin. Adaptive grid-based document layout. *ACM Trans. Graph.*, 22(3):838–847, July 2003.

[JMPS96] Ramesh Johari, Joe Marks, Ali Partovi, and Stuart Shieber. Automatic yellow-pages pagination and layout. Technical report, Mitsubish Electric Research Laboratories, 1996.

[Ker82] Brian W. Kernighan. Computing science technical report no. 97, *A Typesetter Independent TROFF*. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey 07974, 1982.

[Knu74] Donald E. Knuth. Structured programming with go to statements. *Computing Surveys*, 6(4), December 1974.

[Knu84] Donald E. Knuth. *The TEXbook*. Addison Wesley, 1984.

[Knu99] Donald E. Knuth. *Digital Typography*. CSLI Publications, 1999.

[KP81] D. E. Knuth and M. F. Plass. Breaking paragraphs into lines. *Software — Practice and Experience*, 11:1119–1184, 1981.

[KVW89] Brian W. Kernighan and Christopher J. Van Wyk. Page makeup by postprocessing text formatter output. Technical report, AT&T Bell Laboratories, 1989.

[LB95] William S. Lovegrove and David F. Brailsford. Document analysis of PDF files: methods, results and implications. *Electronic Publishing — Origination, Dissemination and Design*, 8(2 & 3):207–220, 1995.

[LB11] Gordon E. Legge and Charles A. Bigelow. Does print size matter for reading? A review of findings from vision science and typography. *Journal of Vision*, 11(5):8:1–22, 2011.

[Lia83] Franklin Mark Liang. *Word Hy-phen-a-tion by Com-put-er*. PhD thesis, Stanford University, 1983.

[Lom03] Chris Lomont. Fast inverse square root. Technical report, 2003.

[Mar13] Simone Marinai. Reflowing and annotating scientific papers on ebook readers. In *Proceedings of the 13th ACM Symposium on Document Engineering*, DocEng '13, New York, NY, USA, 2013. ACM.

[MBB05] Alexander J. Macdonald, David F. Brailsford, and Steven R. Bagley. Encapsulating and manipulating component object graphics (COGs) using SVG. In *Proceedings of the 2005 ACM Symposium on Document Engineering*, pages 61–63. ACM Press, 2005.

[MR91] F Mittelbach and C Rowley. The pursuit of quality — how can automated typesetting achieve the highest standards of craft typography? In *EP92 (Proceedings of Electronic Publishing)*, pages 261–273. Cambridge University Press, 1991.

[PBB11] Alexander J. Pinkney, Steven R. Bagley, and David F. Brailsford. Reflowable documents composed from pre-rendered atomic components. In *Proceedings of the 11th ACM Symposium on Document Engineering*, DocEng '11, pages 163–166, New York, NY, USA, 2011. ACM.

[PBB13] Alexander J. Pinkney, Steven R. Bagley, and David F. Brailsford. No need to justify your choice: Pre-compiling line breaks to improve eBook readability. In *Proceedings of the 13th ACM Symposium on Document Engineering*, DocEng '13, New York, NY, USA, 2013. ACM.

[PHOF03] Lisa Purvis, Steven Harrington, Barry O'Sullivan, and Eugene C. Freuder. Creating personalized documents: an optimization approach. In *Proceedings of the 2003 ACM Symposium on Document Engineering*, pages 68–77. ACM Press, 2003.

[Pla81] Michael Frederick Plass. *Optimal Pagination Techniques for Automatic Typesetting Systems*. PhD thesis, Stanford University, 1981.

[SB95] Philip N. Smith and David F. Brailsford. Towards structured, block-based PDF. *Electronic Publishing — Origination, Dissemination and Design*, 8(2 & 3):153–165, June/September 1995.

[SDJ+08] Evan Schrier, Mira Dontcheva, Charles Jacobs, Geraldine Wade, and David Salesin. Adaptive layout for dynamically aggregated documents. In *Proceedings of the 13th international conference on Intelligent user interfaces*, IUI '08, pages 99–108, New York, NY, USA, 2008. ACM.

[Sha51] Claude Shannon. The redundancy of english. In *Cybernetics: Circular Causal and Feedback Mechanisms in Biological and Social Systems*. Josiah Macy, Jr. Foundation, 1951.

[Voo11] Garret Voorhees. Congeniality of reading on digital devices. Master's thesis, Rochester Institute Of Technology, 2011.

[War91] John Warnock. The Camelot project. Technical report, Adobe, 1991.

[Zip32] George Kingsley Zipf. *Selected Studies of the Principle of Relative Frequency in Language*. Harvard university press, 1932.

[ZL77] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

GLOSSARY

COG Component Object Graphic. 21, 23, 25, 28, 30, 31, 39

HTML Hypertext Markup Language. 9, 10, 12, 13, 39, 47, 51, 55, 76, 77, 89, 122

JSON JavaScript Object Notation. 41, 55, 69

PDF Portable Document Format. 9–12, 19, 21, 23, 25, 26, 28–31, 39, 47, 50, 51, 81, 82, 89

W3C World Wide Web Consortium. 50

galley A metal tray into which physical type is set. 15, 20, 21, 23, 25, 28, 29, 31, 34, 35, 37, 40, 41, 44, 52, 59, 62, 69, 74, 75, 78, 79, 81, 87, 90, 103, 113

glyph The smallest typesettable item. Generally a glyph is a single character, but compounds of characters (such as ligatures and dipthongs) are also logical glyphs, though they may represent two separate characters. 8, 9, 20, 52, 90

justification The process of adjusting spacing between (or within) words on a line to effect some change in the alignment of the text with respect to its left and right margins. 20

kerning Small adjustments to the spacing between specific pairs of letters in order to achieve spacing that appears more even to the eye (compare AVATAR with AVATAR). 6, 9, 20, 100

leading The vertical distance from the baseline of one line of type to the next. In physical typesetting, this was altered by using thin strips of lead, hence the name. 44, 47

measure The standard length of a line of text. 6, 20, 21, 43, 44, 75

point
A common unit of measure in typesetting, usually defined as $\frac{1}{72}$ of an inch. 10, 21, 41, 57, 75

ragged right
Text that is aligned to the left margin but not the right is said to be set *flush left, ragged right*. Commonly also known as *left justified* text. Similarly, *right justified* text may also be referred to as being set *ragged left, flush right*. 5, 34

tracking
Also known as letter-spacing. Similar to kerning, though tracking is applied between every pair of characters identically. Tracking is used to alter the perceived density of a body of text. 9

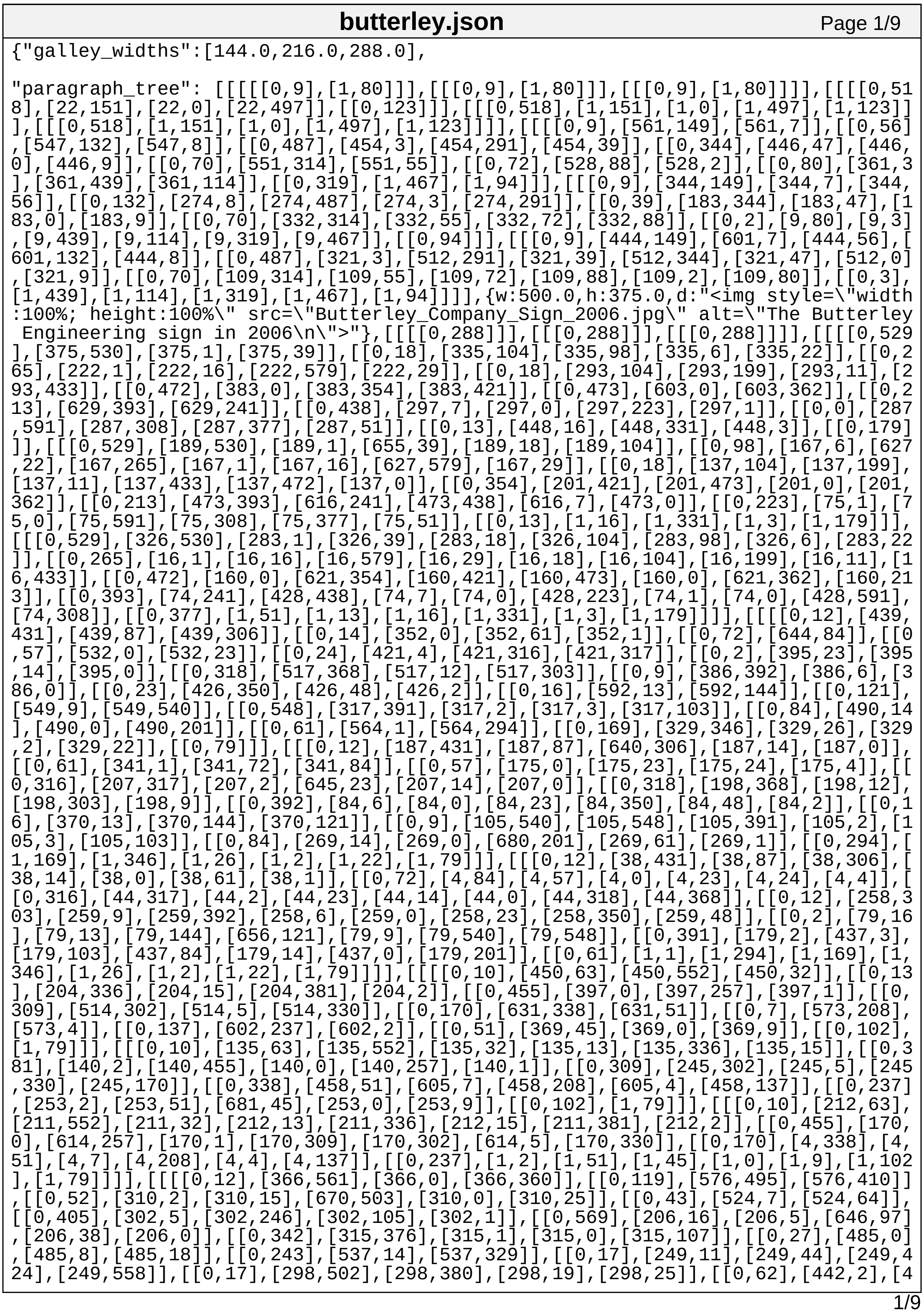
Part IV

APPENDICES

A
A SAMPLE MALLEABLE DOCUMENT

The following pages contain the full layout data of a reasonably short (~1200 word) document: the Wikipedia entry for The Butterley Company. (See http://en.wikipedia.org/wiki/Butterley_Company).

It contains three galley renderings, and has ordered dictionaries both for words, and for position deltas.

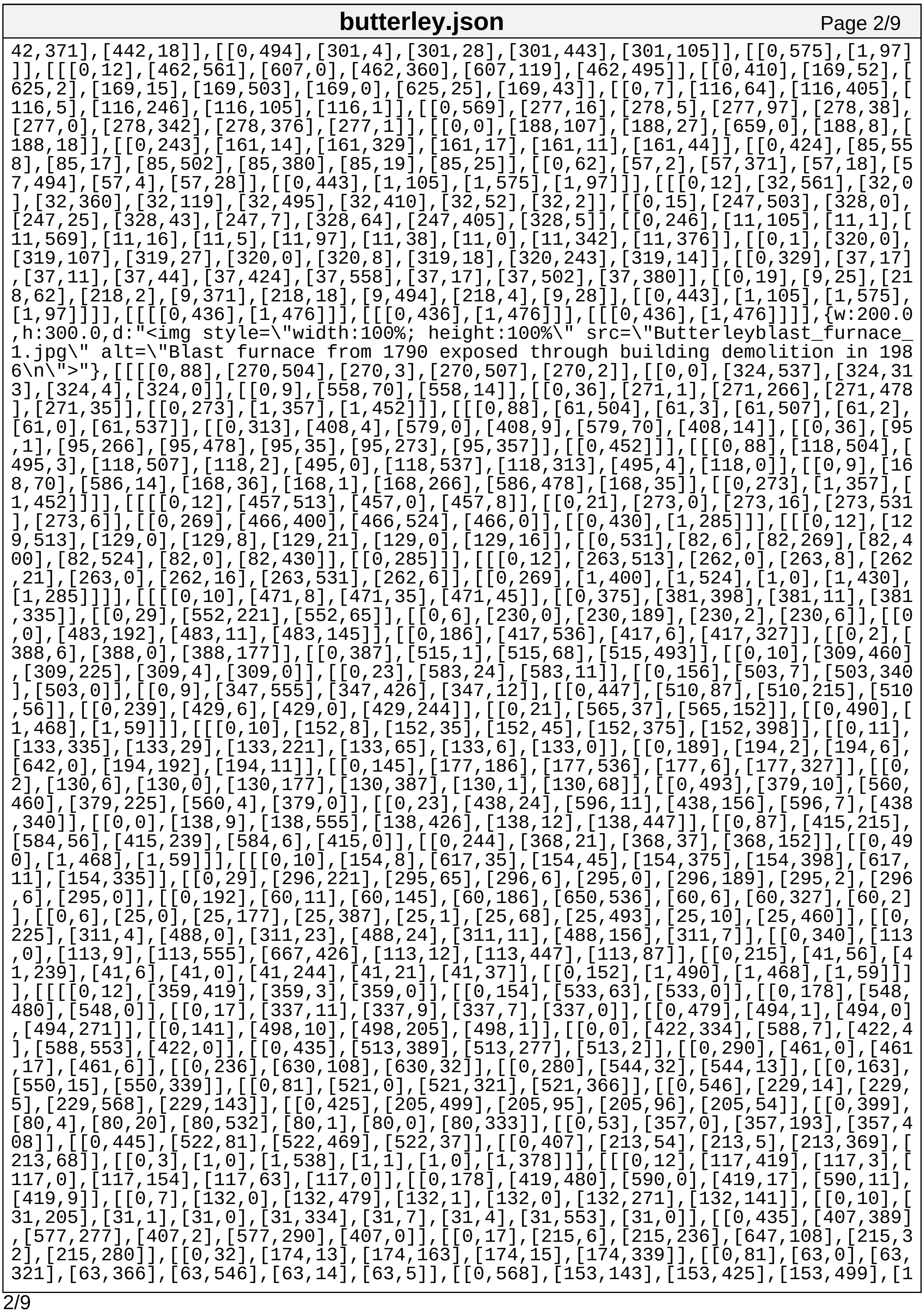


butterley.json Page 1/9

```
{"galley_widths":[144.0,216.0,288.0],

"paragraph_tree": [[[[[0,9],[1,80]]],[[[0,9],[1,80]]],[[[0,9],[1,80]]]],[[[[0,51
8],[22,151],[22,0],[22,497]],[[0,123]]],[[[0,518],[1,151],[1,0],[1,497],[1,123]]
],[[[0,518],[1,151],[1,0],[1,497],[1,123]]]],[[[[0,9],[561,149],[561,7]],[[0,56]
,[547,132],[547,8]],[[0,487],[454,3],[454,291],[454,39]],[[0,344],[446,47],[446,
0],[446,9]],[[0,70],[551,314],[551,55]],[[0,72],[528,88],[528,2]],[[0,80],[361,3
],[361,439],[361,114]],[[0,319],[1,467],[1,94]]],[[[0,9],[344,149],[344,7],[344,
56]],[[0,132],[274,8],[274,487],[274,3],[274,291]],[[0,39],[183,344],[183,47],[1
83,0],[183,9]],[[0,70],[332,314],[332,55],[332,72],[332,88]],[[0,2],[9,80],[9,3]
,[9,439],[9,114],[9,319],[9,467]],[[0,94]]],[[[0,9],[444,149],[601,7],[444,56],[
601,132],[444,8]],[[0,487],[321,3],[512,291],[321,39],[512,344],[321,47],[512,0]
,[321,9]],[[0,70],[109,314],[109,55],[109,72],[109,88],[109,2],[109,80]],[[0,3],
[1,439],[1,114],[1,319],[1,467],[1,94]]]],{w:500.0,h:375.0,d:"<img style=\"width
:100%; height:100%\" src=\"Butterley_Company_Sign_2006.jpg\" alt=\"The Butterley
 Engineering sign in 2006\n\">"},[[[[0,288]]],[[[0,288]]],[[[0,288]]]],[[[[0,529
],[375,530],[375,1],[375,39]],[[0,18],[335,104],[335,98],[335,6],[335,22]],[[0,2
65],[222,1],[222,16],[222,579],[222,29]],[[0,18],[293,104],[293,199],[293,11],[2
93,433]],[[0,472],[383,0],[383,354],[383,421]],[[0,473],[603,0],[603,362]],[[0,2
13],[629,393],[629,241]],[[0,438],[297,7],[297,0],[297,223],[297,1]],[[0,0],[287
,591],[287,308],[287,377],[287,51]],[[0,13],[448,16],[448,331],[448,3]],[[0,179]
]],[[[0,529],[189,530],[189,1],[655,39],[189,18],[189,104]],[[0,98],[167,6],[627
,22],[167,265],[167,1],[167,16],[627,579],[167,29]],[[0,18],[137,104],[137,199],
[137,11],[137,433],[137,472],[137,0]],[[0,354],[201,421],[201,473],[201,0],[201,
362]],[[0,213],[473,393],[616,241],[473,438],[616,7],[473,0]],[[0,223],[75,1],[7
5,0],[75,591],[75,308],[75,377],[75,51]],[[0,13],[1,16],[1,331],[1,3],[1,179]]],
[[[0,529],[326,530],[283,1],[326,39],[283,18],[326,104],[283,98],[326,6],[283,22
]],[[0,265],[16,1],[16,16],[16,579],[16,29],[16,18],[16,104],[16,199],[16,11],[1
6,433]],[[0,472],[160,0],[621,354],[160,421],[160,473],[160,0],[621,362],[160,21
3]],[[0,393],[74,241],[428,438],[74,7],[74,0],[428,223],[74,1],[74,0],[428,591],
[74,308]],[[0,377],[1,51],[1,13],[1,16],[1,331],[1,3],[1,179]]]],[[[[0,12],[439,
431],[439,87],[439,306]],[[0,14],[352,0],[352,61],[352,1]],[[0,72],[644,84]],[[0
,57],[532,0],[532,23]],[[0,24],[421,4],[421,316],[421,317]],[[0,2],[395,23],[395
,14],[395,0]],[[0,318],[517,368],[517,12],[517,303]],[[0,9],[386,392],[386,6],[3
86,0]],[[0,23],[426,350],[426,48],[426,2]],[[0,16],[592,13],[592,144]],[[0,121],
[549,9],[549,540]],[[0,548],[317,391],[317,2],[317,3],[317,103]],[[0,84],[490,14
],[490,0],[490,201]],[[0,61],[564,1],[564,294]],[[0,169],[329,346],[329,26],[329
,2],[329,22]],[[0,79]]],[[[0,12],[187,431],[187,87],[640,306],[187,14],[187,0]],
[[0,61],[341,1],[341,72],[341,84]],[[0,57],[175,0],[175,23],[175,24],[175,4]],[[
0,316],[207,317],[207,2],[645,23],[207,14],[207,0]],[[0,318],[198,368],[198,12],
[198,303],[198,9]],[[0,392],[84,6],[84,0],[84,23],[84,350],[84,48],[84,2]],[[0,1
6],[370,13],[370,144],[370,121]],[[0,9],[105,540],[105,548],[105,391],[105,2],[1
05,3],[105,103]],[[0,84],[269,14],[269,0],[680,201],[269,61],[269,1]],[[0,294],[
1,169],[1,346],[1,26],[1,2],[1,22],[1,79]]],[[[0,12],[38,431],[38,87],[38,306],[
38,14],[38,0],[38,61],[38,1]],[[0,72],[4,84],[4,57],[4,0],[4,23],[4,24],[4,4]],[
[0,316],[44,317],[44,2],[44,23],[44,14],[44,0],[44,318],[44,368]],[[0,12],[258,3
03],[259,9],[259,392],[258,6],[259,0],[258,23],[258,350],[259,48]],[[0,2],[79,16
],[79,13],[79,144],[656,121],[79,9],[79,540],[79,548]],[[0,391],[179,2],[437,3],
[179,103],[437,84],[179,14],[437,0],[179,201]],[[0,61],[1,1],[1,294],[1,169],[1,
346],[1,26],[1,2],[1,22],[1,79]]]],[[[[0,10],[450,63],[450,552],[450,32]],[[0,13
],[204,336],[204,15],[204,381],[204,2]],[[0,455],[397,0],[397,257],[397,1]],[[0,
309],[514,302],[514,5],[514,330]],[[0,170],[631,338],[631,51]],[[0,7],[573,208],
[573,4]],[[0,137],[602,237],[602,2]],[[0,51],[369,45],[369,0],[369,9]],[[0,102],
[1,79]]],[[[0,10],[135,63],[135,552],[135,32],[135,13],[135,336],[135,15]],[[0,3
81],[140,2],[140,455],[140,0],[140,257],[140,1]],[[0,309],[245,302],[245,5],[245
,330],[245,170]],[[0,338],[458,51],[605,7],[458,208],[605,4],[458,137]],[[0,237]
,[253,2],[253,51],[681,45],[253,0],[253,9]],[[0,102],[1,79]]],[[[0,10],[212,63],
[211,552],[211,32],[212,13],[211,336],[212,15],[211,381],[212,2]],[[0,455],[170,
0],[614,257],[170,1],[170,309],[170,302],[614,5],[170,330]],[[0,170],[4,338],[4,
51],[4,7],[4,208],[4,4],[4,137]],[[0,237],[1,2],[1,51],[1,45],[1,0],[1,9],[1,102
],[1,79]]]],[[[[0,12],[366,561],[366,0],[366,360]],[[0,119],[576,495],[576,410]]
,[[0,52],[310,2],[310,15],[670,503],[310,0],[310,25]],[[0,43],[524,7],[524,64]],
[[0,405],[302,5],[302,246],[302,105],[302,1]],[[0,569],[206,16],[206,5],[646,97]
,[206,38],[206,0]],[[0,342],[315,376],[315,1],[315,0],[315,107]],[[0,27],[485,0]
,[485,8],[485,18]],[[0,243],[537,14],[537,329]],[[0,17],[249,11],[249,44],[249,4
24],[249,558]],[[0,17],[298,502],[298,380],[298,19],[298,25]],[[0,62],[442,2],[4
```

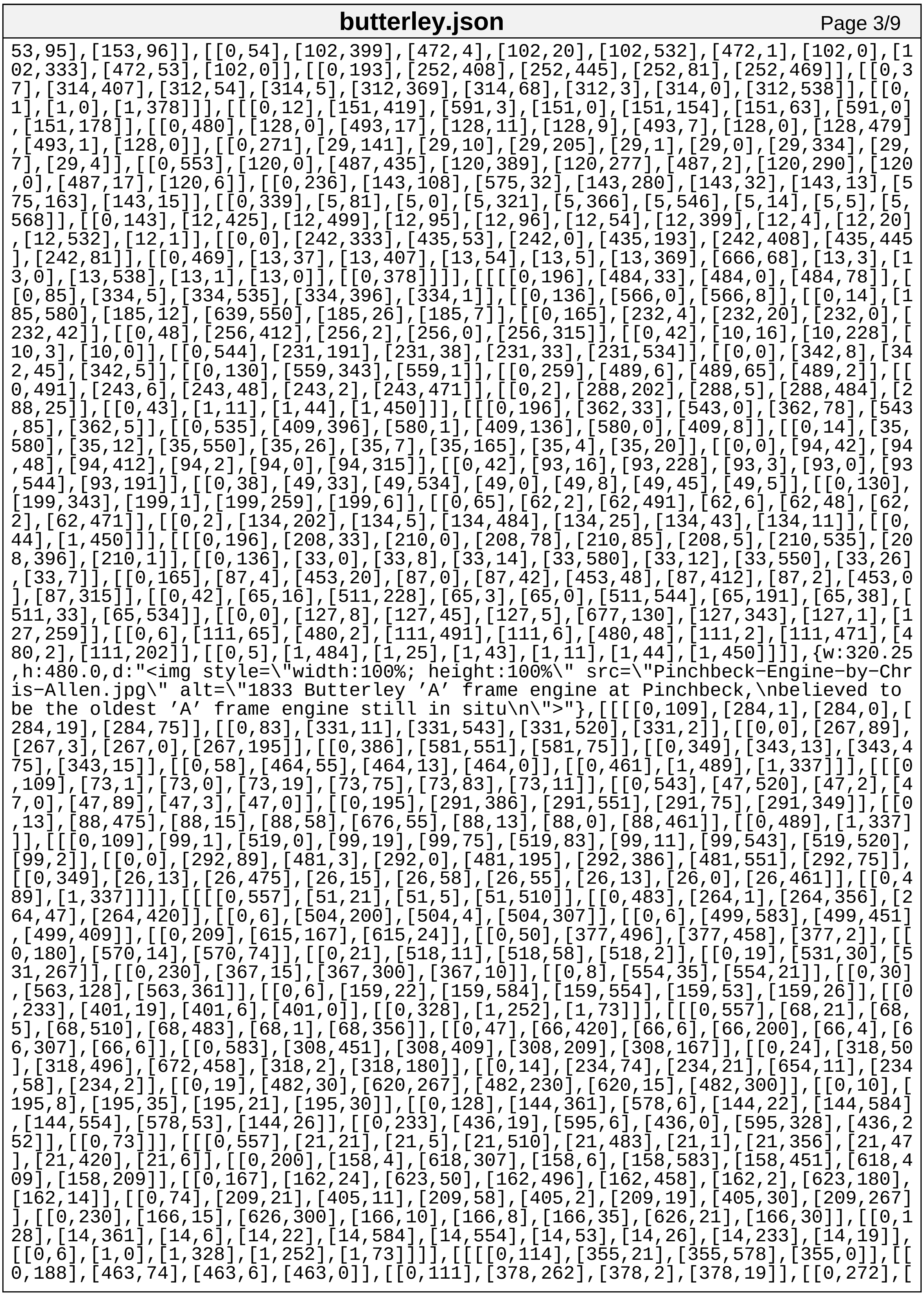
1/9



butterley.json Page 2/9

```
42,371],[442,18]],[[0,494],[301,4],[301,28],[301,443],[301,105]],[[0,575],[1,97]
]],[[[0,12],[462,561],[607,0],[462,360],[607,119],[462,495]],[[0,410],[169,52],[
625,2],[169,15],[169,503],[169,0],[625,25],[169,43]],[[0,7],[116,64],[116,405],[
116,5],[116,246],[116,105],[116,1]],[[0,569],[277,16],[278,5],[277,97],[278,38],
[277,0],[278,342],[278,376],[277,1]],[[0,0],[188,107],[188,27],[659,0],[188,8],[
188,18]],[[0,243],[161,14],[161,329],[161,17],[161,11],[161,44]],[[0,424],[85,55
8],[85,17],[85,502],[85,380],[85,19],[85,25]],[[0,62],[57,2],[57,371],[57,18],[5
7,494],[57,4],[57,28]],[[0,443],[1,105],[1,575],[1,97]]],[[[0,12],[32,561],[32,0
],[32,360],[32,119],[32,495],[32,410],[32,52],[32,2]],[[0,15],[247,503],[328,0],
[247,25],[328,43],[247,7],[328,64],[247,405],[328,5]],[[0,246],[11,105],[11,1],[
11,569],[11,16],[11,5],[11,97],[11,38],[11,0],[11,342],[11,376]],[[0,1],[320,0],
[319,107],[319,27],[320,0],[320,8],[319,18],[320,243],[319,14]],[[0,329],[37,17]
,[37,11],[37,44],[37,424],[37,558],[37,17],[37,502],[37,380]],[[0,19],[9,25],[21
8,62],[218,2],[9,371],[218,18],[9,494],[218,4],[9,28]],[[0,443],[1,105],[1,575],
[1,97]]]],[[[[0,436],[1,476]]],[[[0,436],[1,476]]],[[[0,436],[1,476]]]],{w:200.0
,h:300.0,d:"<img style=\"width:100%; height:100%\" src=\"Butterleyblast_furnace_
1.jpg\" alt=\"Blast furnace from 1790 exposed through building demolition in 198
6\n\">"},[[[[0,88],[270,504],[270,3],[270,507],[270,2]],[[0,0],[324,537],[324,31
3],[324,4],[324,0]],[[0,9],[558,70],[558,14]],[[0,36],[271,1],[271,266],[271,478
],[271,35]],[[0,273],[1,357],[1,452]]],[[[0,88],[61,504],[61,3],[61,507],[61,2],
[61,0],[61,537]],[[0,313],[408,4],[579,0],[408,9],[579,70],[408,14]],[[0,36],[95
,1],[95,266],[95,478],[95,35],[95,273],[95,357]],[[0,452]]],[[[0,88],[118,504],[
495,3],[118,507],[118,2],[495,0],[118,537],[118,313],[495,4],[118,0]],[[0,9],[16
8,70],[586,14],[168,36],[168,1],[168,266],[586,478],[168,35]],[[0,273],[1,357],[
1,452]]]],[[[[0,12],[457,513],[457,0],[457,8]],[[0,21],[273,0],[273,16],[273,531
],[273,6]],[[0,269],[466,400],[466,524],[466,0]],[[0,430],[1,285]]],[[[0,12],[12
9,513],[129,0],[129,8],[129,21],[129,0],[129,16]],[[0,531],[82,6],[82,269],[82,4
00],[82,524],[82,0],[82,430]],[[0,285]]],[[[0,12],[263,513],[262,0],[263,8],[262
,21],[263,0],[262,16],[263,531],[262,6]],[[0,269],[1,400],[1,524],[1,0],[1,430],
[1,285]]]],[[[[0,10],[471,8],[471,35],[471,45]],[[0,375],[381,398],[381,11],[381
,335]],[[0,29],[552,221],[552,65]],[[0,6],[230,0],[230,189],[230,2],[230,6]],[[0
,0],[483,192],[483,11],[483,145]],[[0,186],[417,536],[417,6],[417,327]],[[0,2],[
388,6],[388,0],[388,177]],[[0,387],[515,1],[515,68],[515,493]],[[0,10],[309,460]
,[309,225],[309,4],[309,0]],[[0,23],[583,24],[583,11]],[[0,156],[503,7],[503,340
],[503,0]],[[0,9],[347,555],[347,426],[347,12]],[[0,447],[510,87],[510,215],[510
,56]],[[0,239],[429,6],[429,0],[429,244]],[[0,21],[565,37],[565,152]],[[0,490],[
1,468],[1,59]]],[[[0,10],[152,8],[152,35],[152,45],[152,375],[152,398]],[[0,11],
[133,335],[133,29],[133,221],[133,65],[133,6],[133,0]],[[0,189],[194,2],[194,6],
[642,0],[194,192],[194,11]],[[0,145],[177,186],[177,536],[177,6],[177,327]],[[0,
2],[130,6],[130,0],[130,177],[130,387],[130,1],[130,68]],[[0,493],[379,10],[560,
460],[379,225],[560,4],[379,0]],[[0,23],[438,24],[596,11],[438,156],[596,7],[438
,340]],[[0,0],[138,9],[138,555],[138,426],[138,12],[138,447]],[[0,87],[415,215],
[584,56],[415,239],[584,6],[415,0]],[[0,244],[368,21],[368,37],[368,152]],[[0,49
0],[1,468],[1,59]]],[[[0,10],[154,8],[617,35],[154,45],[154,375],[154,398],[617,
11],[154,335]],[[0,29],[296,221],[295,65],[296,6],[295,0],[296,189],[295,2],[296
,6],[295,0]],[[0,192],[60,11],[60,145],[60,186],[650,536],[60,6],[60,327],[60,2]
],[[0,6],[25,0],[25,177],[25,387],[25,1],[25,68],[25,493],[25,10],[25,460]],[[0,
225],[311,4],[488,0],[311,23],[488,24],[311,11],[488,156],[311,7]],[[0,340],[113
,0],[113,9],[113,555],[667,426],[113,12],[113,447],[113,87]],[[0,215],[41,56],[4
1,239],[41,6],[41,0],[41,244],[41,21],[41,37]],[[0,152],[1,490],[1,468],[1,59]]]
],[[[[0,12],[359,419],[359,3],[359,0]],[[0,154],[533,63],[533,0]],[[0,178],[548,
480],[548,0]],[[0,17],[337,11],[337,9],[337,7],[337,0]],[[0,479],[494,1],[494,0]
,[494,271]],[[0,141],[498,10],[498,205],[498,1]],[[0,0],[422,334],[588,7],[422,4
],[588,553],[422,0]],[[0,435],[513,389],[513,277],[513,2]],[[0,290],[461,0],[461
,17],[461,6]],[[0,236],[630,108],[630,32]],[[0,280],[544,32],[544,13]],[[0,163],
[550,15],[550,339]],[[0,81],[521,0],[521,321],[521,366]],[[0,546],[229,14],[229,
5],[229,568],[229,143]],[[0,425],[205,499],[205,95],[205,96],[205,54]],[[0,399],
[80,4],[80,20],[80,532],[80,1],[80,0],[80,333]],[[0,53],[357,0],[357,193],[357,4
08]],[[0,445],[522,81],[522,469],[522,37]],[[0,407],[213,54],[213,5],[213,369],[
213,68]],[[0,3],[1,0],[1,538],[1,1],[1,0],[1,378]]],[[[0,12],[117,419],[117,3],[
117,0],[117,154],[117,63],[117,0]],[[0,178],[419,480],[590,0],[419,17],[590,11],
[419,9]],[[0,7],[132,0],[132,479],[132,1],[132,0],[132,271],[132,141]],[[0,10],[
31,205],[31,1],[31,0],[31,334],[31,7],[31,4],[31,553],[31,0]],[[0,435],[407,389]
,[577,277],[407,2],[577,290],[407,0]],[[0,17],[215,6],[215,236],[647,108],[215,3
2],[215,280]],[[0,32],[174,13],[174,163],[174,15],[174,339]],[[0,81],[63,0],[63,
321],[63,366],[63,546],[63,14],[63,5]],[[0,568],[153,143],[153,425],[153,499],[1
```

2/9

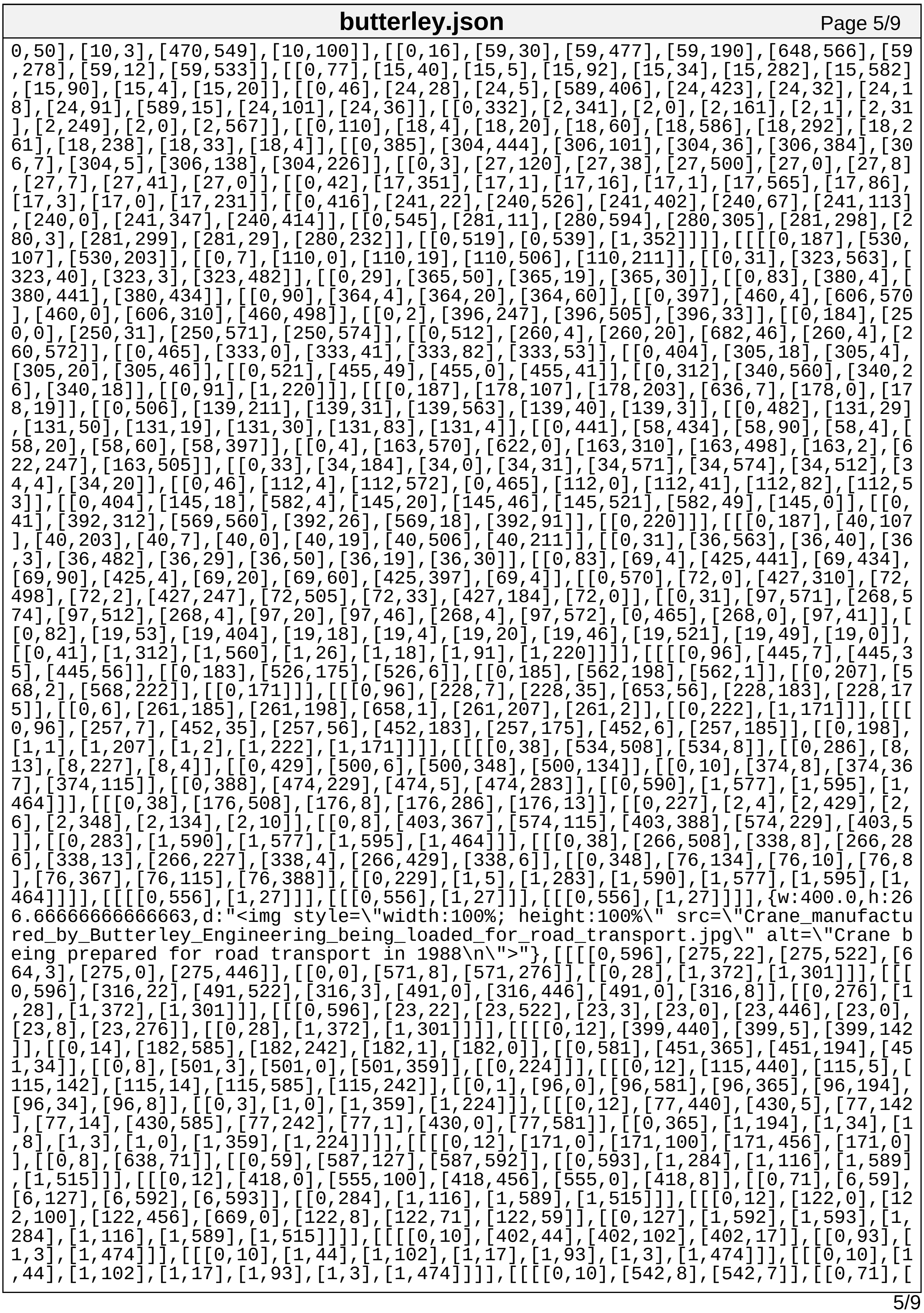


butterley.json Page 3/9

```
53,95],[153,96]],[[0,54],[102,399],[472,4],[102,20],[102,532],[472,1],[102,0],[1
02,333],[472,53],[102,0]],[[0,193],[252,408],[252,445],[252,81],[252,469]],[[0,3
7],[314,407],[312,54],[314,5],[312,369],[314,68],[312,3],[314,0],[312,538]],[[0,
1],[1,0],[1,378]]],[[[0,12],[151,419],[591,3],[151,0],[151,154],[151,63],[591,0]
,[151,178]],[[0,480],[128,0],[493,17],[128,11],[128,9],[493,7],[128,0],[128,479]
,[493,1],[128,0]],[[0,271],[29,141],[29,10],[29,205],[29,1],[29,0],[29,334],[29,
7],[29,4]],[[0,553],[120,0],[487,435],[120,389],[120,277],[487,2],[120,290],[120
,0],[487,17],[120,6]],[[0,236],[143,108],[575,32],[143,280],[143,32],[143,13],[5
75,163],[143,15]],[[0,339],[5,81],[5,0],[5,321],[5,366],[5,546],[5,14],[5,5],[5,
568]],[[0,143],[12,425],[12,499],[12,95],[12,96],[12,54],[12,399],[12,4],[12,20]
,[12,532],[12,1]],[[0,0],[242,333],[435,53],[242,0],[435,193],[242,408],[435,445
],[242,81]],[[0,469],[13,37],[13,407],[13,54],[13,5],[13,369],[666,68],[13,3],[1
3,0],[13,538],[13,1],[13,0]],[[0,378]]]],[[[[0,196],[484,33],[484,0],[484,78]],[
[0,85],[334,5],[334,535],[334,396],[334,1]],[[0,136],[566,0],[566,8]],[[0,14],[1
85,580],[185,12],[639,550],[185,26],[185,7]],[[0,165],[232,4],[232,20],[232,0],[
232,42]],[[0,48],[256,412],[256,2],[256,0],[256,315]],[[0,42],[10,16],[10,228],[
10,3],[10,0]],[[0,544],[231,191],[231,38],[231,33],[231,534]],[[0,0],[342,8],[34
2,45],[342,5]],[[0,130],[559,343],[559,1]],[[0,259],[489,6],[489,65],[489,2]],[[
0,491],[243,6],[243,48],[243,2],[243,471]],[[0,2],[288,202],[288,5],[288,484],[2
88,25]],[[0,43],[1,11],[1,44],[1,450]]],[[[0,196],[362,33],[543,0],[362,78],[543
,85],[362,5]],[[0,535],[409,396],[580,1],[409,136],[580,0],[409,8]],[[0,14],[35,
580],[35,12],[35,550],[35,26],[35,7],[35,165],[35,4],[35,20]],[[0,0],[94,42],[94
,48],[94,412],[94,2],[94,0],[94,315]],[[0,42],[93,16],[93,228],[93,3],[93,0],[93
,544],[93,191]],[[0,38],[49,33],[49,534],[49,0],[49,8],[49,45],[49,5]],[[0,130],
[199,343],[199,1],[199,259],[199,6]],[[0,65],[62,2],[62,491],[62,6],[62,48],[62,
2],[62,471]],[[0,2],[134,202],[134,5],[134,484],[134,25],[134,43],[134,11]],[[0,
44],[1,450]]],[[[0,196],[208,33],[210,0],[208,78],[210,85],[208,5],[210,535],[20
8,396],[210,1]],[[0,136],[33,0],[33,8],[33,14],[33,580],[33,12],[33,550],[33,26]
,[33,7]],[[0,165],[87,4],[453,20],[87,0],[87,42],[453,48],[87,412],[87,2],[453,0
],[87,315]],[[0,42],[65,16],[511,228],[65,3],[65,0],[511,544],[65,191],[65,38],[
511,33],[65,534]],[[0,0],[127,8],[127,45],[127,5],[677,130],[127,343],[127,1],[1
27,259]],[[0,6],[111,65],[480,2],[111,491],[111,6],[480,48],[111,2],[111,471],[4
80,2],[111,202]],[[0,5],[1,484],[1,25],[1,43],[1,11],[1,44],[1,450]]]],{w:320.25
,h:480.0,d:"<img style=\"width:100%; height:100%\" src=\"Pinchbeck−Engine−by−Chr
is−Allen.jpg\" alt=\"1833 Butterley ’A’ frame engine at Pinchbeck,\nbelieved to
be the oldest ’A’ frame engine still in situ\n\">"},[[[[0,109],[284,1],[284,0],[
284,19],[284,75]],[[0,83],[331,11],[331,543],[331,520],[331,2]],[[0,0],[267,89],
[267,3],[267,0],[267,195]],[[0,386],[581,551],[581,75]],[[0,349],[343,13],[343,4
75],[343,15]],[[0,58],[464,55],[464,13],[464,0]],[[0,461],[1,489],[1,337]]],[[[0
,109],[73,1],[73,0],[73,19],[73,75],[73,83],[73,11]],[[0,543],[47,520],[47,2],[4
7,0],[47,89],[47,3],[47,0]],[[0,195],[291,386],[291,551],[291,75],[291,349]],[[0
,13],[88,475],[88,15],[88,58],[676,55],[88,13],[88,0],[88,461]],[[0,489],[1,337]
]],[[[0,109],[99,1],[519,0],[99,19],[99,75],[519,83],[99,11],[99,543],[519,520],
[99,2]],[[0,0],[292,89],[481,3],[292,0],[481,195],[292,386],[481,551],[292,75]],
[[0,349],[26,13],[26,475],[26,15],[26,58],[26,55],[26,13],[26,0],[26,461]],[[0,4
89],[1,337]]]],[[[[0,557],[51,21],[51,5],[51,510]],[[0,483],[264,1],[264,356],[2
64,47],[264,420]],[[0,6],[504,200],[504,4],[504,307]],[[0,6],[499,583],[499,451]
,[499,409]],[[0,209],[615,167],[615,24]],[[0,50],[377,496],[377,458],[377,2]],[[
0,180],[570,14],[570,74]],[[0,21],[518,11],[518,58],[518,2]],[[0,19],[531,30],[5
31,267]],[[0,230],[367,15],[367,300],[367,10]],[[0,8],[554,35],[554,21]],[[0,30]
,[563,128],[563,361]],[[0,6],[159,22],[159,584],[159,554],[159,53],[159,26]],[[0
,233],[401,19],[401,6],[401,0]],[[0,328],[1,252],[1,73]]],[[[0,557],[68,21],[68,
5],[68,510],[68,483],[68,1],[68,356]],[[0,47],[66,420],[66,6],[66,200],[66,4],[6
6,307],[66,6]],[[0,583],[308,451],[308,409],[308,209],[308,167]],[[0,24],[318,50
],[318,496],[672,458],[318,2],[318,180]],[[0,14],[234,74],[234,21],[654,11],[234
,58],[234,2]],[[0,19],[482,30],[620,267],[482,230],[620,15],[482,300]],[[0,10],[
195,8],[195,35],[195,21],[195,30]],[[0,128],[144,361],[578,6],[144,22],[144,584]
,[144,554],[578,53],[144,26]],[[0,233],[436,19],[595,6],[436,0],[595,328],[436,2
52]],[[0,73]]],[[[0,557],[21,21],[21,5],[21,510],[21,483],[21,1],[21,356],[21,47
],[21,420],[21,6]],[[0,200],[158,4],[618,307],[158,6],[158,583],[158,451],[618,4
09],[158,209]],[[0,167],[162,24],[623,50],[162,496],[162,458],[162,2],[623,180],
[162,14]],[[0,74],[209,21],[405,11],[209,58],[405,2],[209,19],[405,30],[209,267]
],[[0,230],[166,15],[626,300],[166,10],[166,8],[166,35],[626,21],[166,30]],[[0,1
28],[14,361],[14,6],[14,22],[14,584],[14,554],[14,53],[14,26],[14,233],[14,19]],
[[0,6],[1,0],[1,328],[1,252],[1,73]]]],[[[[0,114],[355,21],[355,578],[355,0]],[[
0,188],[463,74],[463,6],[463,0]],[[0,111],[378,262],[378,2],[378,19]],[[0,272],[
```

3/9

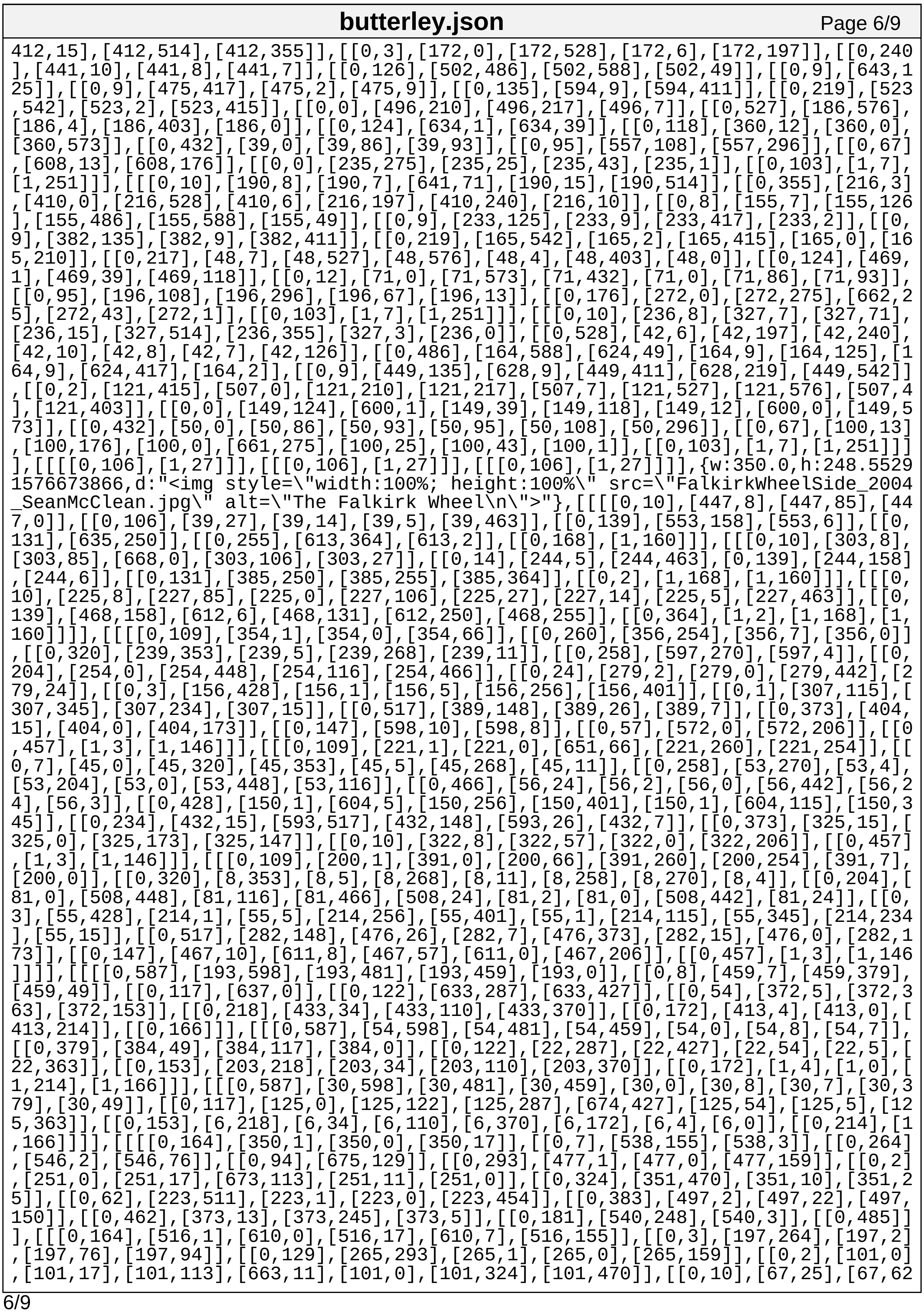
```
539,382],[539,0]],[[0,235],[529,253],[529,2]],[[0,216],[505,547],[505,281],[505,
2]],[[0,0],[387,23],[387,2],[387,562]],[[0,509],[414,73],[414,597],[414,295]],[[
0,89],[285,6],[285,0],[285,358],[285,395]],[[0,3],[376,311],[376,99],[376,11]],[
[0,182],[632,564],[632,541]],[[0,133]]],[[[0,114],[443,21],[599,578],[443,0],[59
9,188],[443,74]],[[0,6],[3,0],[3,111],[3,262],[3,2],[3,19],[3,272]],[[0,382],[22
4,0],[224,235],[224,253],[224,2]],[[0,216],[276,547],[276,281],[665,2],[276,0],[
276,23]],[[0,2],[353,562],[535,509],[353,73],[535,597],[353,295]],[[0,89],[86,6]
,[86,0],[86,358],[86,395],[86,3],[86,311]],[[0,99],[1,11],[1,182],[1,564],[1,541
],[1,133]]],[[[0,114],[290,21],[289,578],[290,0],[289,188],[290,74],[289,6],[290
,0],[289,111]],[[0,262],[89,2],[89,19],[89,272],[89,382],[89,0],[89,235]],[[0,25
3],[46,2],[46,216],[46,547],[46,281],[46,2],[46,0]],[[0,23],[237,2],[238,562],[2
37,509],[238,73],[237,597],[238,295],[237,89],[238,6]],[[0,0],[78,358],[440,395]
,[78,3],[78,311],[440,99],[78,11],[78,182],[440,564],[78,541]],[[0,133]]]],[[[[0
,10],[349,8],[349,7],[349,449]],[[0,4],[192,374],[192,3],[192,0],[192,111]],[[0,
69],[406,82],[406,6],[406,31]],[[0,140],[541,3],[541,413]],[[0,453],[486,36],[48
6,1],[486,523]],[[0,174],[313,34],[313,525],[313,52],[313,5]],[[0,325],[393,47],
[393,112],[393,488]],[[0,69],[286,390],[286,5],[286,492],[286,1]],[[0,37],[520,1
57],[520,37],[520,304]],[[0,10],[416,274],[416,13],[416,394]],[[0,28],[220,0],[2
20,78],[220,3],[220,99]],[[0,4],[345,326],[345,0],[345,297]],[[0,212],[1,1],[1,0
],[1,162]]],[[[0,10],[148,8],[585,7],[148,449],[148,4],[148,374],[585,3],[148,0]
],[[0,111],[173,69],[173,82],[173,6],[173,31]],[[0,140],[107,3],[107,413],[107,4
53],[107,36],[107,1],[107,523]],[[0,174],[90,34],[90,525],[90,52],[657,5],[90,32
5],[90,47],[90,112]],[[0,488],[126,69],[126,390],[126,5],[126,492],[126,1],[126,
37]],[[0,157],[147,37],[147,304],[147,10],[147,274],[147,13]],[[0,394],[70,28],[
70,0],[70,78],[70,3],[70,99],[70,4]],[[0,326],[142,0],[142,297],[142,212],[142,1
],[142,0]],[[0,162]]],[[[0,10],[91,8],[506,7],[91,449],[91,4],[506,374],[91,3],[
91,0],[506,111],[91,69]],[[0,82],[255,6],[336,31],[255,140],[336,3],[255,413],[3
36,453],[255,36],[336,1]],[[0,523],[20,174],[20,34],[20,525],[20,52],[20,5],[20,
325],[20,47],[20,112],[20,488]],[[0,69],[28,390],[28,5],[28,492],[28,1],[28,37],
[28,157],[28,37],[28,304]],[[0,10],[124,274],[492,13],[124,394],[124,28],[492,0]
,[124,78],[124,3],[492,99],[124,4]],[[0,326],[1,0],[1,297],[1,212],[1,1],[1,0],[
1,162]]]]],{w:250.0,h:187.5,d:"<img style=\"width:100%; height:100%\" src=\"Butte
rley_co_plate.jpg\" alt=\"Butterley Company plate in St Pancras station\n\">"},[
[[[0,289],[525,323],[525,13]],[[0,422],[330,52],[330,15],[330,0],[330,66]],[[0,2
63],[423,112],[423,559],[423,279]],[[0,12],[420,76],[420,501],[420,77]],[[0,40],
[51,5],[51,92],[51,1]],[[0,64],[545,5],[545,418],[683,322]],[[0,16],[300,516],[3
00,98],[300,55],[300,0]],[[0,9],[400,437],[400,50],[400,3]],[[0,549],[398,100],[
398,16],[398,30]],[[0,477],[567,190],[567,566]],[[0,278],[456,12],[456,533],[456
,77]],[[0,40],[339,5],[339,92],[339,34]],[[0,282],[248,582],[248,90],[248,4],[24
8,20]],[[0,46],[299,28],[299,5],[299,406],[299,423]],[[0,32],[294,18],[294,91],[
294,15],[294,101]],[[0,36],[371,332],[371,341],[371,0]],[[0,161],[5,1],[5,31],[5
,249]],[[0,0],[202,567],[202,110],[202,4],[202,20]],[[0,60],[536,586],[536,292]]
,[[0,261],[434,238],[434,33],[434,4]],[[0,385],[390,444],[390,101],[390,36]],[[0
,384],[358,7],[358,5],[358,138]],[[0,226],[424,3],[424,120],[424,38]],[[0,500],[
394,0],[394,8],[394,7]],[[0,41],[7,0],[7,42],[7,351],[7,1]],[[0,16],[157,1],[157
,565],[157,86],[157,3],[157,0]],[[0,231],[348,416],[348,22],[348,526]],[[0,402],
[363,67],[363,113],[363,0]],[[0,347],[246,414],[246,545],[246,11],[246,594]],[[0
,305],[431,298],[431,3],[431,299]],[[0,29],[556,232],[556,519],[0,539]],[[0,352]
]],[[[0,289],[119,323],[119,13],[119,422],[119,52],[119,15],[119,0]],[[0,66],[21
9,263],[219,112],[649,559],[219,279],[219,12]],[[0,76],[184,501],[184,77],[184,4
0],[184,5]],[[0,92],[146,1],[146,64],[146,5],[146,418],[0,322],[146,16]],[[0,516
],[103,98],[103,55],[103,0],[103,9],[103,437],[103,50]],[[0,3],[92,549],[92,100]
,[92,16],[92,30],[92,477],[92,190]],[[0,566],[478,278],[110,12],[478,533],[110,7
7],[478,40]],[[0,5],[64,92],[64,34],[64,282],[652,582],[64,90],[64,4],[64,20]],[
[0,46],[114,28],[114,5],[114,406],[114,423],[114,32],[114,18]],[[0,91],[83,15],[
83,101],[83,36],[83,332],[83,341],[83,0]],[[0,161],[7,1],[7,31],[7,249],[7,0],[7
,567],[7,110],[7,4]],[[0,20],[191,60],[191,586],[191,292],[191,261]],[[0,238],[1
04,33],[104,4],[104,385],[104,444],[104,101],[104,36]],[[0,384],[141,7],[141,5],
[141,138],[141,226],[141,3]],[[0,120],[479,38],[619,500],[479,0],[619,8],[479,7]
],[[0,41],[98,0],[98,42],[98,351],[660,1],[98,16],[98,1],[98,565]],[[0,86],[3,3]
,[3,0],[3,231],[3,416],[3,22],[3,526]],[[0,402],[226,67],[226,113],[678,0],[226,
347],[226,414]],[[0,545],[136,11],[136,594],[136,305],[136,298],[136,3],[136,299
]],[[0,29],[1,232],[1,519],[0,539],[1,352]]],[[[0,289],[43,323],[43,13],[43,422]
,[43,52],[43,15],[43,0],[43,66]],[[0,263],[123,112],[123,559],[123,279],[671,12]
,[123,76],[123,501],[123,77]],[[0,40],[108,5],[108,92],[108,1],[679,64],[108,5],
[108,418],[0,322],[108,16]],[[0,516],[10,98],[470,55],[10,0],[10,9],[470,437],[1
```



butterley.json Page 5/9

```
0,50],[10,3],[470,549],[10,100]],[[0,16],[59,30],[59,477],[59,190],[648,566],[59
,278],[59,12],[59,533]],[[0,77],[15,40],[15,5],[15,92],[15,34],[15,282],[15,582]
,[15,90],[15,4],[15,20]],[[0,46],[24,28],[24,5],[589,406],[24,423],[24,32],[24,1
8],[24,91],[589,15],[24,101],[24,36]],[[0,332],[2,341],[2,0],[2,161],[2,1],[2,31
],[2,249],[2,0],[2,567]],[[0,110],[18,4],[18,20],[18,60],[18,586],[18,292],[18,2
61],[18,238],[18,33],[18,4]],[[0,385],[304,444],[306,101],[304,36],[306,384],[30
6,7],[304,5],[306,138],[304,226]],[[0,3],[27,120],[27,38],[27,500],[27,0],[27,8]
,[27,7],[27,41],[27,0]],[[0,42],[17,351],[17,1],[17,16],[17,1],[17,565],[17,86],
[17,3],[17,0],[17,231]],[[0,416],[241,22],[240,526],[241,402],[240,67],[241,113]
,[240,0],[241,347],[240,414]],[[0,545],[281,11],[280,594],[280,305],[281,298],[2
80,3],[281,299],[281,29],[280,232]],[[0,519],[0,539],[1,352]]]],[[[[0,187],[530,
107],[530,203]],[[0,7],[110,0],[110,19],[110,506],[110,211]],[[0,31],[323,563],[
323,40],[323,3],[323,482]],[[0,29],[365,50],[365,19],[365,30]],[[0,83],[380,4],[
380,441],[380,434]],[[0,90],[364,4],[364,20],[364,60]],[[0,397],[460,4],[606,570
],[460,0],[606,310],[460,498]],[[0,2],[396,247],[396,505],[396,33]],[[0,184],[25
0,0],[250,31],[250,571],[250,574]],[[0,512],[260,4],[260,20],[682,46],[260,4],[2
60,572]],[[0,465],[333,0],[333,41],[333,82],[333,53]],[[0,404],[305,18],[305,4],
[305,20],[305,46]],[[0,521],[455,49],[455,0],[455,41]],[[0,312],[340,560],[340,2
6],[340,18]],[[0,91],[1,220]]],[[[0,187],[178,107],[178,203],[636,7],[178,0],[17
8,19]],[[0,506],[139,211],[139,31],[139,563],[139,40],[139,3]],[[0,482],[131,29]
,[131,50],[131,19],[131,30],[131,83],[131,4]],[[0,441],[58,434],[58,90],[58,4],[
58,20],[58,60],[58,397]],[[0,4],[163,570],[622,0],[163,310],[163,498],[163,2],[6
22,247],[163,505]],[[0,33],[34,184],[34,0],[34,31],[34,571],[34,574],[34,512],[3
4,4],[34,20]],[[0,46],[112,4],[112,572],[0,465],[112,0],[112,41],[112,82],[112,5
3]],[[0,404],[145,18],[582,4],[145,20],[145,46],[145,521],[582,49],[145,0]],[[0,
41],[392,312],[569,560],[392,26],[569,18],[392,91]],[[0,220]]],[[[0,187],[40,107
],[40,203],[40,7],[40,0],[40,19],[40,506],[40,211]],[[0,31],[36,563],[36,40],[36
,3],[36,482],[36,29],[36,50],[36,19],[36,30]],[[0,83],[69,4],[425,441],[69,434],
[69,90],[425,4],[69,20],[69,60],[425,397],[69,4]],[[0,570],[72,0],[427,310],[72,
498],[72,2],[427,247],[72,505],[72,33],[427,184],[72,0]],[[0,31],[97,571],[268,5
74],[97,512],[268,4],[97,20],[97,46],[268,4],[97,572],[0,465],[268,0],[97,41]],[
[0,82],[19,53],[19,404],[19,18],[19,4],[19,20],[19,46],[19,521],[19,49],[19,0]],
[[0,41],[1,312],[1,560],[1,26],[1,18],[1,91],[1,220]]]],[[[[0,96],[445,7],[445,3
5],[445,56]],[[0,183],[526,175],[526,6]],[[0,185],[562,198],[562,1]],[[0,207],[5
68,2],[568,222]],[[0,171]]],[[[0,96],[228,7],[228,35],[653,56],[228,183],[228,17
5]],[[0,6],[261,185],[261,198],[658,1],[261,207],[261,2]],[[0,222],[1,171]]],[[[
0,96],[257,7],[452,35],[257,56],[452,183],[257,175],[452,6],[257,185]],[[0,198],
[1,1],[1,207],[1,2],[1,222],[1,171]]]],[[[[0,38],[534,508],[534,8]],[[0,286],[8,
13],[8,227],[8,4]],[[0,429],[500,6],[500,348],[500,134]],[[0,10],[374,8],[374,36
7],[374,115]],[[0,388],[474,229],[474,5],[474,283]],[[0,590],[1,577],[1,595],[1,
464]]],[[[0,38],[176,508],[176,8],[176,286],[176,13]],[[0,227],[2,4],[2,429],[2,
6],[2,348],[2,134],[2,10]],[[0,8],[403,367],[574,115],[403,388],[574,229],[403,5
]],[[0,283],[1,590],[1,577],[1,595],[1,464]]],[[[0,38],[266,508],[338,8],[266,28
6],[338,13],[266,227],[338,4],[266,429],[338,6]],[[0,348],[76,134],[76,10],[76,8
],[76,367],[76,115],[76,388]],[[0,229],[1,5],[1,283],[1,590],[1,577],[1,595],[1,
464]]]],[[[[0,556],[1,27]]],[[[0,556],[1,27]]],[[[0,556],[1,27]]]],{w:400.0,h:26
6.666666666663,d:"<img style=\"width:100%; height:100%\" src=\"Crane_manufactu
red_by_Butterley_Engineering_being_loaded_for_road_transport.jpg\" alt=\"Crane b
eing prepared for road transport in 1988\n\">"},[[[[0,596],[275,22],[275,522],[6
64,3],[275,0],[275,446]],[[0,0],[571,8],[571,276]],[[0,28],[1,372],[1,301]]],[[[
0,596],[316,22],[491,522],[316,3],[491,0],[316,446],[491,0],[316,8]],[[0,276],[1
,28],[1,372],[1,301]]],[[[0,596],[23,22],[23,522],[23,3],[23,0],[23,446],[23,0],
[23,8],[23,276]],[[0,28],[1,372],[1,301]]]],[[[[0,12],[399,440],[399,5],[399,142
]],[[0,14],[182,585],[182,242],[182,1],[182,0]],[[0,581],[451,365],[451,194],[45
1,34]],[[0,8],[501,3],[501,0],[501,359]],[[0,224]]],[[[0,12],[115,440],[115,5],[
115,142],[115,14],[115,585],[115,242]],[[0,1],[96,0],[96,581],[96,365],[96,194],
[96,34],[96,8]],[[0,3],[1,0],[1,359],[1,224]]],[[[0,12],[77,440],[430,5],[77,142
],[77,14],[430,585],[77,242],[77,1],[430,0],[77,581]],[[0,365],[1,194],[1,34],[1
,8],[1,3],[1,0],[1,359],[1,224]]]],[[[[0,12],[171,0],[171,100],[171,456],[171,0]
],[[0,8],[638,71]],[[0,59],[587,127],[587,592]],[[0,593],[1,284],[1,116],[1,589]
,[1,515]]],[[[0,12],[418,0],[555,100],[418,456],[555,0],[418,8]],[[0,71],[6,59],
[6,127],[6,592],[6,593]],[[0,284],[1,116],[1,589],[1,515]]],[[[0,12],[122,0],[12
2,100],[122,456],[669,0],[122,8],[122,71],[122,59]],[[0,127],[1,592],[1,593],[1,
284],[1,116],[1,589],[1,515]]]],[[[[0,10],[402,44],[402,102],[402,17]],[[0,93],[
1,3],[1,474]]],[[[0,10],[1,44],[1,102],[1,17],[1,93],[1,3],[1,474]]],[[[0,10],[1
,44],[1,102],[1,17],[1,93],[1,3],[1,474]]]],[[[[0,10],[542,8],[542,7]],[[0,71],[
```

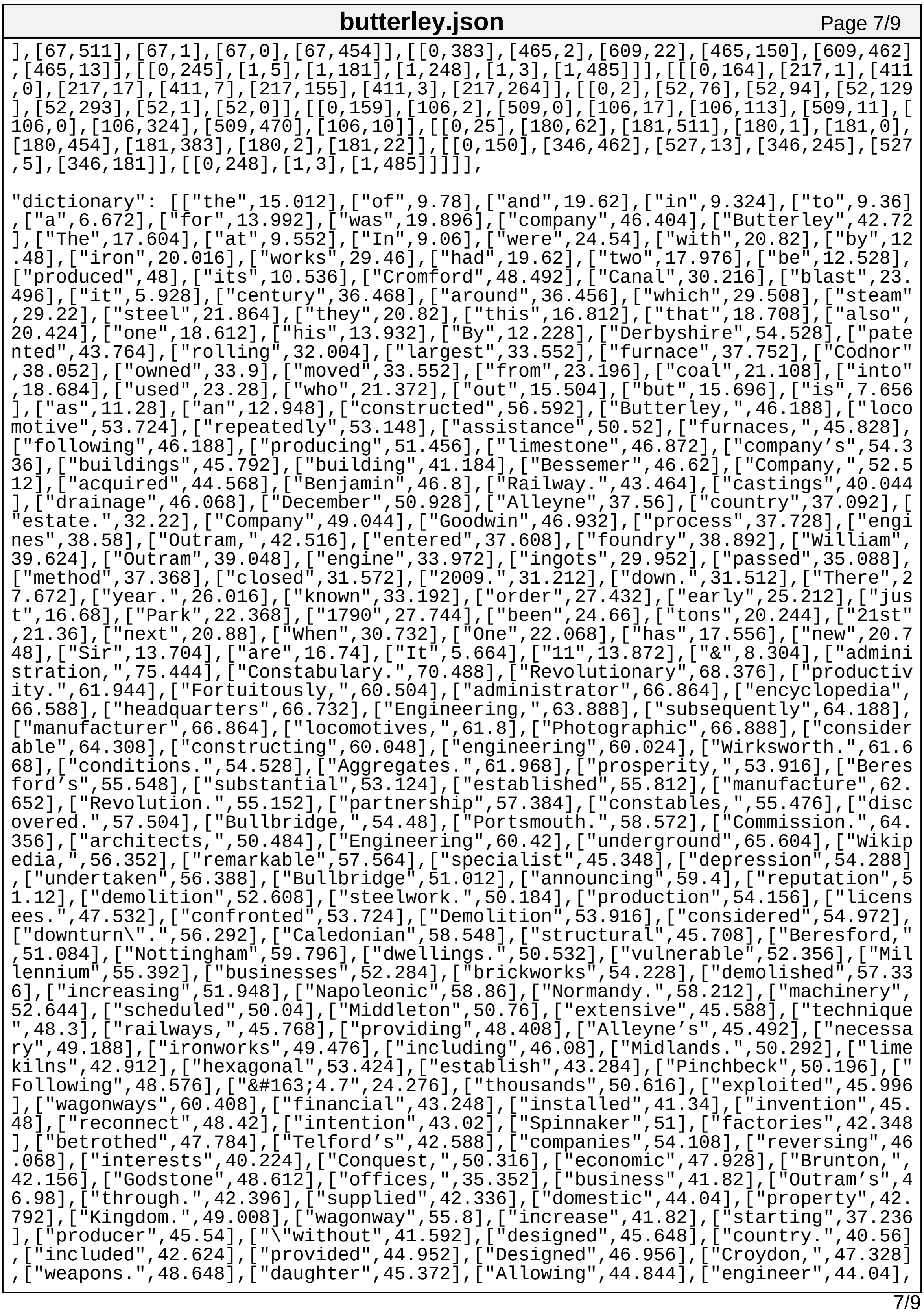
5/9



butterley.json Page 6/9

```
412,15],[412,514],[412,355]],[[0,3],[172,0],[172,528],[172,6],[172,197]],[[0,240
],[441,10],[441,8],[441,7]],[[0,126],[502,486],[502,588],[502,49]],[[0,9],[643,1
25]],[[0,9],[475,417],[475,2],[475,9]],[[0,135],[594,9],[594,411]],[[0,219],[523
,542],[523,2],[523,415]],[[0,0],[496,210],[496,217],[496,7]],[[0,527],[186,576],
[186,4],[186,403],[186,0]],[[0,124],[634,1],[634,39]],[[0,118],[360,12],[360,0],
[360,573]],[[0,432],[39,0],[39,86],[39,93]],[[0,95],[557,108],[557,296]],[[0,67]
,[608,13],[608,176]],[[0,0],[235,275],[235,25],[235,43],[235,1]],[[0,103],[1,7],
[1,251]]],[[[0,10],[190,8],[190,7],[641,71],[190,15],[190,514]],[[0,355],[216,3]
,[410,0],[216,528],[410,6],[216,197],[410,240],[216,10]],[[0,8],[155,7],[155,126
],[155,486],[155,588],[155,49]],[[0,9],[233,125],[233,9],[233,417],[233,2]],[[0,
9],[382,135],[382,9],[382,411]],[[0,219],[165,542],[165,2],[165,415],[165,0],[16
5,210]],[[0,217],[48,7],[48,527],[48,576],[48,4],[48,403],[48,0]],[[0,124],[469,
1],[469,39],[469,118]],[[0,12],[71,0],[71,573],[71,432],[71,0],[71,86],[71,93]],
[[0,95],[196,108],[196,296],[196,67],[196,13]],[[0,176],[272,0],[272,275],[662,2
5],[272,43],[272,1]],[[0,103],[1,7],[1,251]]],[[[0,10],[236,8],[327,7],[327,71],
[236,15],[327,514],[236,355],[327,3],[236,0]],[[0,528],[42,6],[42,197],[42,240],
[42,10],[42,8],[42,7],[42,126]],[[0,486],[164,588],[624,49],[164,9],[164,125],[1
64,9],[624,417],[164,2]],[[0,9],[449,135],[628,9],[449,411],[628,219],[449,542]]
,[[0,2],[121,415],[507,0],[121,210],[121,217],[507,7],[121,527],[121,576],[507,4
],[121,403]],[[0,0],[149,124],[600,1],[149,39],[149,118],[149,12],[600,0],[149,5
73]],[[0,432],[50,0],[50,86],[50,93],[50,95],[50,108],[50,296]],[[0,67],[100,13]
,[100,176],[100,0],[661,275],[100,25],[100,43],[100,1]],[[0,103],[1,7],[1,251]]]
],[[[[0,106],[1,27]]],[[[0,106],[1,27]]],[[[0,106],[1,27]]]],{w:350.0,h:248.5529
1576673866,d:"<img style=\"width:100%; height:100%\" src=\"FalkirkWheelSide_2004
_SeanMcClean.jpg\" alt=\"The Falkirk Wheel\n\">"},[[[[0,10],[447,8],[447,85],[44
7,0]],[[0,106],[39,27],[39,14],[39,5],[39,463]],[[0,139],[553,158],[553,6]],[[0,
131],[635,250]],[[0,255],[613,364],[613,2]],[[0,168],[1,160]]],[[[0,10],[303,8],
[303,85],[668,0],[303,106],[303,27]],[[0,14],[244,5],[244,463],[0,139],[244,158]
,[244,6]],[[0,131],[385,250],[385,255],[385,364]],[[0,2],[1,168],[1,160]]],[[[0,
10],[225,8],[227,85],[225,0],[227,106],[225,27],[227,14],[225,5],[227,463]],[[0,
139],[468,158],[612,6],[468,131],[612,250],[468,255]],[[0,364],[1,2],[1,168],[1,
160]]]],[[[[0,109],[354,1],[354,0],[354,66]],[[0,260],[356,254],[356,7],[356,0]]
,[[0,320],[239,353],[239,5],[239,268],[239,11]],[[0,258],[597,270],[597,4]],[[0,
204],[254,0],[254,448],[254,116],[254,466]],[[0,24],[279,2],[279,0],[279,442],[2
79,24]],[[0,3],[156,428],[156,1],[156,5],[156,256],[156,401]],[[0,1],[307,115],[
307,345],[307,234],[307,15]],[[0,517],[389,148],[389,26],[389,7]],[[0,373],[404,
15],[404,0],[404,173]],[[0,147],[598,10],[598,8]],[[0,57],[572,0],[572,206]],[[0
,457],[1,3],[1,146]]],[[[0,109],[221,1],[221,0],[651,66],[221,260],[221,254]],[[
0,7],[45,0],[45,320],[45,353],[45,5],[45,268],[45,11]],[[0,258],[53,270],[53,4],
[53,204],[53,0],[53,448],[53,116]],[[0,466],[56,24],[56,2],[56,0],[56,442],[56,2
4],[56,3]],[[0,428],[150,1],[604,5],[150,256],[150,401],[150,1],[604,115],[150,3
45]],[[0,234],[432,15],[593,517],[432,148],[593,26],[432,7]],[[0,373],[325,15],[
325,0],[325,173],[325,147]],[[0,10],[322,8],[322,57],[322,0],[322,206]],[[0,457]
,[1,3],[1,146]]],[[[0,109],[200,1],[391,0],[200,66],[391,260],[200,254],[391,7],
[200,0]],[[0,320],[8,353],[8,5],[8,268],[8,11],[8,258],[8,270],[8,4]],[[0,204],[
81,0],[508,448],[81,116],[81,466],[508,24],[81,2],[81,0],[508,442],[81,24]],[[0,
3],[55,428],[214,1],[55,5],[214,256],[55,401],[55,1],[214,115],[55,345],[214,234
],[55,15]],[[0,517],[282,148],[476,26],[282,7],[476,373],[282,15],[476,0],[282,1
73]],[[0,147],[467,10],[611,8],[467,57],[611,0],[467,206]],[[0,457],[1,3],[1,146
]]]],[[[[0,587],[193,598],[193,481],[193,459],[193,0]],[[0,8],[459,7],[459,379],
[459,49]],[[0,117],[637,0]],[[0,122],[633,287],[633,427]],[[0,54],[372,5],[372,3
63],[372,153]],[[0,218],[433,34],[433,110],[433,370]],[[0,172],[413,4],[413,0],[
413,214]],[[0,166]]],[[[0,587],[54,598],[54,481],[54,459],[54,0],[54,8],[54,7]],
[[0,379],[384,49],[384,117],[384,0]],[[0,122],[22,287],[22,427],[22,54],[22,5],[
22,363]],[[0,153],[203,218],[203,34],[203,110],[203,370]],[[0,172],[1,4],[1,0],[
1,214],[1,166]]],[[[0,587],[30,598],[30,481],[30,459],[30,0],[30,8],[30,7],[30,3
79],[30,49]],[[0,117],[125,0],[125,122],[125,287],[674,427],[125,54],[125,5],[12
5,363]],[[0,153],[6,218],[6,34],[6,110],[6,370],[6,172],[6,4],[6,0]],[[0,214],[1
,166]]]],[[[[0,164],[350,1],[350,0],[350,17]],[[0,7],[538,155],[538,3]],[[0,264]
,[546,2],[546,76]],[[0,94],[675,129]],[[0,293],[477,1],[477,0],[477,159]],[[0,2]
,[251,0],[251,17],[673,113],[251,11],[251,0]],[[0,324],[351,470],[351,10],[351,2
5]],[[0,62],[223,511],[223,1],[223,0],[223,454]],[[0,383],[497,2],[497,22],[497,
150]],[[0,462],[373,13],[373,245],[373,5]],[[0,181],[540,248],[540,3]],[[0,485]]
],[[[0,164],[516,1],[610,0],[516,17],[610,7],[516,155]],[[0,3],[197,264],[197,2]
,[197,76],[197,94]],[[0,129],[265,293],[265,1],[265,0],[265,159]],[[0,2],[101,0]
,[101,17],[101,113],[663,11],[101,0],[101,324],[101,470]],[[0,10],[67,25],[67,62
```

6/9

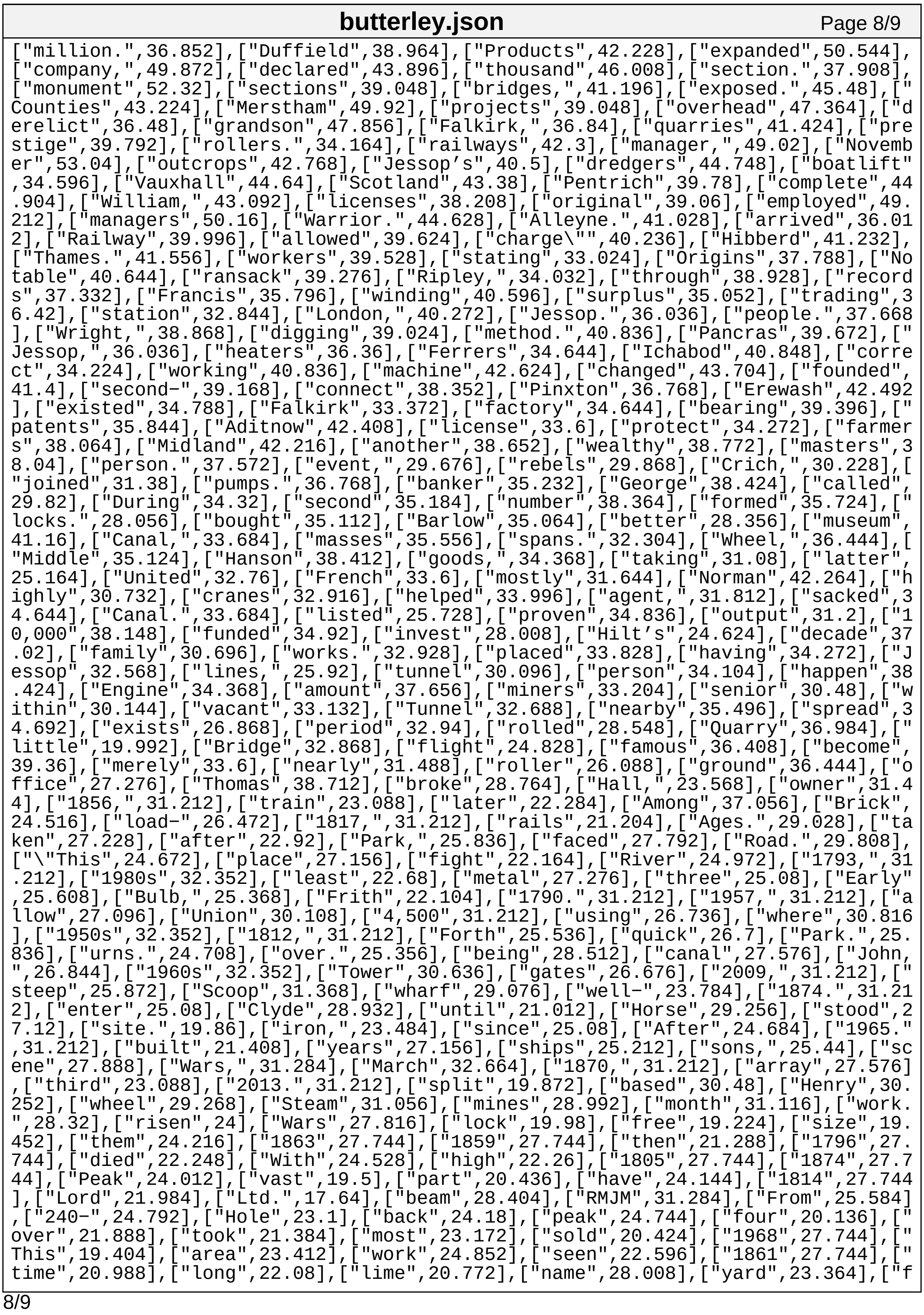


butterley.json Page 7/9

```
],[67,511],[67,1],[67,0],[67,454]],[[0,383],[465,2],[609,22],[465,150],[609,462]
,[465,13]],[[0,245],[1,5],[1,181],[1,248],[1,3],[1,485]]],[[[0,164],[217,1],[411
,0],[217,17],[411,7],[217,155],[411,3],[217,264]],[[0,2],[52,76],[52,94],[52,129
],[52,293],[52,1],[52,0]],[[0,159],[106,2],[509,0],[106,17],[106,113],[509,11],[
106,0],[106,324],[509,470],[106,10]],[[0,25],[180,62],[181,511],[180,1],[181,0],
[180,454],[181,383],[180,2],[181,22]],[[0,150],[346,462],[527,13],[346,245],[527
,5],[346,181]],[[0,248],[1,3],[1,485]]]]],

"dictionary": [["the",15.012],["of",9.78],["and",19.62],["in",9.324],["to",9.36]
,["a",6.672],["for",13.992],["was",19.896],["company",46.404],["Butterley",42.72
],["The",17.604],["at",9.552],["In",9.06],["were",24.54],["with",20.82],["by",12
.48],["iron",20.016],["works",29.46],["had",19.62],["two",17.976],["be",12.528],
["produced",48],["its",10.536],["Cromford",48.492],["Canal",30.216],["blast",23.
496],["it",5.928],["century",36.468],["around",36.456],["which",29.508],["steam"
,29.22],["steel",21.864],["they",20.82],["this",16.812],["that",18.708],["also",
20.424],["one",18.612],["his",13.932],["By",12.228],["Derbyshire",54.528],["pate
nted",43.764],["rolling",32.004],["largest",33.552],["furnace",37.752],["Codnor"
,38.052],["owned",33.9],["moved",33.552],["from",23.196],["coal",21.108],["into"
,18.684],["used",23.28],["who",21.372],["out",15.504],["but",15.696],["is",7.656
],["as",11.28],["an",12.948],["constructed",56.592],["Butterley,",46.188],["loco
motive",53.724],["repeatedly",53.148],["assistance",50.52],["furnaces,",45.828],
["following",46.188],["producing",51.456],["limestone",46.872],["company’s",54.3
36],["buildings",45.792],["building",41.184],["Bessemer",46.62],["Company,",52.5
12],["acquired",44.568],["Benjamin",46.8],["Railway.",43.464],["castings",40.044
],["drainage",46.068],["December",50.928],["Alleyne",37.56],["country",37.092],[
"estate.",32.22],["Company",49.044],["Goodwin",46.932],["process",37.728],["engi
nes",38.58],["Outram,",42.516],["entered",37.608],["foundry",38.892],["William",
39.624],["Outram",39.048],["engine",33.972],["ingots",29.952],["passed",35.088],
["method",37.368],["closed",31.572],["2009.",31.212],["down.",31.512],["There",2
7.672],["year.",26.016],["known",33.192],["order",27.432],["early",25.212],["jus
t",16.68],["Park",22.368],["1790",27.744],["been",24.66],["tons",20.244],["21st"
,21.36],["next",20.88],["When",30.732],["One",22.068],["has",17.556],["new",20.7
48],["Sir",13.704],["are",16.74],["It",5.664],["11",13.872],["&",8.304],["admini
stration,",75.444],["Constabulary.",70.488],["Revolutionary",68.376],["productiv
ity.",61.944],["Fortuitously,",60.504],["administrator",66.864],["encyclopedia",
66.588],["headquarters",66.732],["Engineering,",63.888],["subsequently",64.188],
["manufacturer",66.864],["locomotives,",61.8],["Photographic",66.888],["consider
able",64.308],["constructing",60.048],["engineering",60.024],["Wirksworth.",61.6
68],["conditions.",54.528],["Aggregates.",61.968],["prosperity,",53.916],["Beres
ford’s",55.548],["substantial",53.124],["established",55.812],["manufacture",62.
652],["Revolution.",55.152],["partnership",57.384],["constables,",55.476],["disc
overed.",57.504],["Bullbridge,",54.48],["Portsmouth.",58.572],["Commission.",64.
356],["architects,",50.484],["Engineering",60.42],["underground",65.604],["Wikip
edia,",56.352],["remarkable",57.564],["specialist",45.348],["depression",54.288]
,["undertaken",56.388],["Bullbridge",51.012],["announcing",59.4],["reputation",5
1.12],["demolition",52.608],["steelwork.",50.184],["production",54.156],["licens
ees.",47.532],["confronted",53.724],["Demolition",53.916],["considered",54.972],
["downturn\".",56.292],["Caledonian",58.548],["structural",45.708],["Beresford,"
,51.084],["Nottingham",59.796],["dwellings.",50.532],["vulnerable",52.356],["Mil
lennium",55.392],["businesses",52.284],["brickworks",54.228],["demolished",57.33
6],["increasing",51.948],["Napoleonic",58.86],["Normandy.",58.212],["machinery",
52.644],["scheduled",50.04],["Middleton",50.76],["extensive",45.588],["technique
",48.3],["railways,",45.768],["providing",48.408],["Alleyne’s",45.492],["necessa
ry",49.188],["ironworks",49.476],["including",46.08],["Midlands.",50.292],["lime
kilns",42.912],["hexagonal",53.424],["establish",43.284],["Pinchbeck",50.196],["
Following",48.576],["&#163;4.7",24.276],["thousands",50.616],["exploited",45.996
],["wagonways",60.408],["financial",43.248],["installed",41.34],["invention",45.
48],["reconnect",48.42],["intention",43.02],["Spinnaker",51],["factories",42.348
],["betrothed",47.784],["Telford’s",42.588],["companies",54.108],["reversing",46
.068],["interests",40.224],["Conquest,",50.316],["economic",47.928],["Brunton,",
42.156],["Godstone",48.612],["offices,",35.352],["business",41.82],["Outram’s",4
6.98],["through.",42.396],["supplied",42.336],["domestic",44.04],["property",42.
792],["Kingdom.",49.008],["wagonway",55.8],["increase",41.82],["starting",37.236
],["producer",45.54],["\"without",41.592],["designed",45.648],["country.",40.56]
,["included",42.624],["provided",44.952],["Designed",46.956],["Croydon,",47.328]
,["weapons.",48.648],["daughter",45.372],["Allowing",44.844],["engineer",44.04],
```

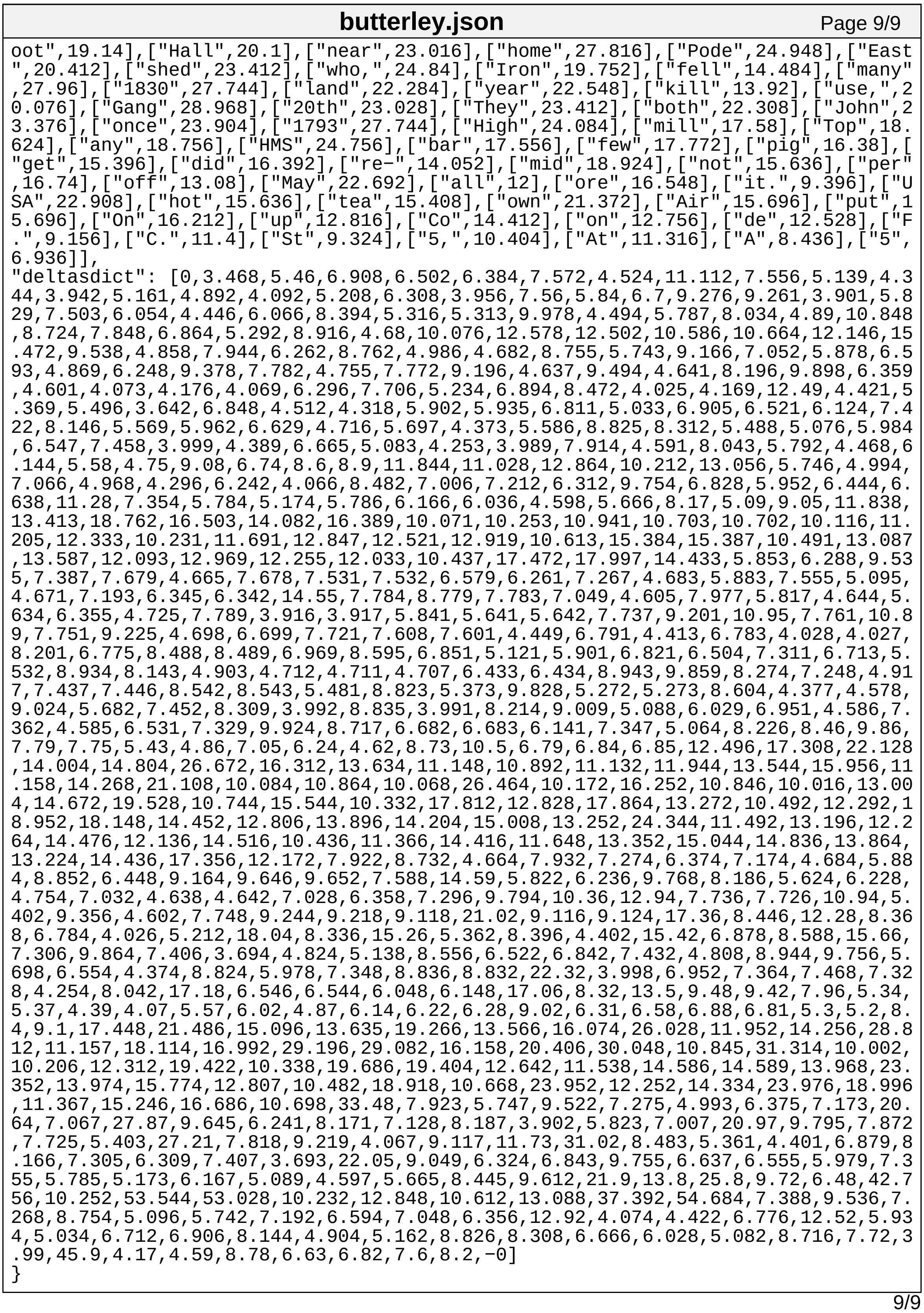
7/9



butterley.json Page 8/9

```
["million.",36.852],["Duffield",38.964],["Products",42.228],["expanded",50.544],
["company,",49.872],["declared",43.896],["thousand",46.008],["section.",37.908],
["monument",52.32],["sections",39.048],["bridges,",41.196],["exposed.",45.48],["
Counties",43.224],["Merstham",49.92],["projects",39.048],["overhead",47.364],["d
erelict",36.48],["grandson",47.856],["Falkirk,",36.84],["quarries",41.424],["pre
stige",39.792],["rollers.",34.164],["railways",42.3],["manager,",49.02],["Novemb
er",53.04],["outcrops",42.768],["Jessop’s",40.5],["dredgers",44.748],["boatlift"
,34.596],["Vauxhall",44.64],["Scotland",43.38],["Pentrich",39.78],["complete",44
.904],["William,",43.092],["licenses",38.208],["original",39.06],["employed",49.
212],["managers",50.16],["Warrior.",44.628],["Alleyne.",41.028],["arrived",36.01
2],["Railway",39.996],["allowed",39.624],["charge\"",40.236],["Hibberd",41.232],
["Thames.",41.556],["workers",39.528],["stating",33.024],["Origins",37.788],["No
table",40.644],["ransack",39.276],["Ripley,",34.032],["through",38.928],["record
s",37.332],["Francis",35.796],["winding",40.596],["surplus",35.052],["trading",3
6.42],["station",32.844],["London,",40.272],["Jessop.",36.036],["people.",37.668
],["Wright,",38.868],["digging",39.024],["method.",40.836],["Pancras",39.672],["
Jessop,",36.036],["heaters",36.36],["Ferrers",34.644],["Ichabod",40.848],["corre
ct",34.224],["working",40.836],["machine",42.624],["changed",43.704],["founded",
41.4],["second−",39.168],["connect",38.352],["Pinxton",36.768],["Erewash",42.492
],["existed",34.788],["Falkirk",33.372],["factory",34.644],["bearing",39.396],["
patents",35.844],["Aditnow",42.408],["license",33.6],["protect",34.272],["farmer
s",38.064],["Midland",42.216],["another",38.652],["wealthy",38.772],["masters",3
8.04],["person.",37.572],["event,",29.676],["rebels",29.868],["Crich,",30.228],[
"joined",31.38],["pumps.",36.768],["banker",35.232],["George",38.424],["called",
29.82],["During",34.32],["second",35.184],["number",38.364],["formed",35.724],["
locks.",28.056],["bought",35.112],["Barlow",35.064],["better",28.356],["museum",
41.16],["Canal,",33.684],["masses",35.556],["spans.",32.304],["Wheel,",36.444],[
"Middle",35.124],["Hanson",38.412],["goods,",34.368],["taking",31.08],["latter",
25.164],["United",32.76],["French",33.6],["mostly",31.644],["Norman",42.264],["h
ighly",30.732],["cranes",32.916],["helped",33.996],["agent,",31.812],["sacked",3
4.644],["Canal.",33.684],["listed",25.728],["proven",34.836],["output",31.2],["1
0,000",38.148],["funded",34.92],["invest",28.008],["Hilt’s",24.624],["decade",37
.02],["family",30.696],["works.",32.928],["placed",33.828],["having",34.272],["J
essop",32.568],["lines,",25.92],["tunnel",30.096],["person",34.104],["happen",38
.424],["Engine",34.368],["amount",37.656],["miners",33.204],["senior",30.48],["w
ithin",30.144],["vacant",33.132],["Tunnel",32.688],["nearby",35.496],["spread",3
4.692],["exists",26.868],["period",32.94],["rolled",28.548],["Quarry",36.984],["
little",19.992],["Bridge",32.868],["flight",24.828],["famous",36.408],["become",
39.36],["merely",33.6],["nearly",31.488],["roller",26.088],["ground",36.444],["o
ffice",27.276],["Thomas",38.712],["broke",28.764],["Hall,",23.568],["owner",31.4
4],["1856,",31.212],["train",23.088],["later",22.284],["Among",37.056],["Brick",
24.516],["load−",26.472],["1817,",31.212],["rails",21.204],["Ages.",29.028],["ta
ken",27.228],["after",22.92],["Park,",25.836],["faced",27.792],["Road.",29.808],
["\"This",24.672],["place",27.156],["fight",22.164],["River",24.972],["1793,",31
.212],["1980s",32.352],["least",22.68],["metal",27.276],["three",25.08],["Early"
,25.608],["Bulb,",25.368],["Frith",22.104],["1790.",31.212],["1957,",31.212],["a
llow",27.096],["Union",30.108],["4,500",31.212],["using",26.736],["where",30.816
],["1950s",32.352],["1812,",31.212],["Forth",25.536],["quick",26.7],["Park.",25.
836],["urns.",24.708],["over.",25.356],["being",28.512],["canal",27.576],["John,
",26.844],["1960s",32.352],["Tower",30.636],["gates",26.676],["2009,",31.212],["
steep",25.872],["Scoop",31.368],["wharf",29.076],["well−",23.784],["1874.",31.21
2],["enter",25.08],["Clyde",28.932],["until",21.012],["Horse",29.256],["stood",2
7.12],["site.",19.86],["iron,",23.484],["since",25.08],["After",24.684],["1965."
,31.212],["built",21.408],["years",27.156],["ships",25.212],["sons,",25.44],["sc
ene",27.888],["Wars,",31.284],["March",32.664],["1870,",31.212],["array",27.576]
,["third",23.088],["2013.",31.212],["split",19.872],["based",30.48],["Henry",30.
252],["wheel",29.268],["Steam",31.056],["mines",28.992],["month",31.116],["work.
",28.32],["risen",24],["Wars",27.816],["lock",19.98],["free",19.224],["size",19.
452],["them",24.216],["1863",27.744],["1859",27.744],["then",21.288],["1796",27.
744],["died",22.248],["With",24.528],["high",22.26],["1805",27.744],["1874",27.7
44],["Peak",24.012],["vast",19.5],["part",20.436],["have",24.144],["1814",27.744
],["Lord",21.984],["Ltd.",17.64],["beam",28.404],["RMJM",31.284],["From",25.584]
,["240−",24.792],["Hole",23.1],["back",24.18],["peak",24.744],["four",20.136],["
over",21.888],["took",21.384],["most",23.172],["sold",20.424],["1968",27.744],["
This",19.404],["area",23.412],["work",24.852],["seen",22.596],["1861",27.744],["
time",20.988],["long",22.08],["lime",20.772],["name",28.008],["yard",23.364],["f
```

8/9



butterley.json Page 9/9

```
oot",19.14],["Hall",20.1],["near",23.016],["home",27.816],["Pode",24.948],["East
",20.412],["shed",23.412],["who,",24.84],["Iron",19.752],["fell",14.484],["many"
,27.96],["1830",27.744],["land",22.284],["year",22.548],["kill",13.92],["use,",2
0.076],["Gang",28.968],["20th",23.028],["They",23.412],["both",22.308],["John",2
3.376],["once",23.904],["1793",27.744],["High",24.084],["mill",17.58],["Top",18.
624],["any",18.756],["HMS",24.756],["bar",17.556],["few",17.772],["pig",16.38],[
"get",15.396],["did",16.392],["re-",14.052],["mid",18.924],["not",15.636],["per"
,16.74],["off",13.08],["May",22.692],["all",12],["ore",16.548],["it.",9.396],["U
SA",22.908],["hot",15.636],["tea",15.408],["own",21.372],["Air",15.696],["put",1
5.696],["On",16.212],["up",12.816],["Co",14.412],["on",12.756],["de",12.528],["F
.",9.156],["C.",11.4],["St",9.324],["5,",10.404],["At",11.316],["A",8.436],["5",
6.936]],
"deltasdict": [0,3.468,5.46,6.908,6.502,6.384,7.572,4.524,11.112,7.556,5.139,4.3
44,3.942,5.161,4.892,4.092,5.208,6.308,3.956,7.56,5.84,6.7,9.276,9.261,3.901,5.8
29,7.503,6.054,4.446,6.066,8.394,5.316,5.313,9.978,4.494,5.787,8.034,4.89,10.848
,8.724,7.848,6.864,5.292,8.916,4.68,10.076,12.578,12.502,10.586,10.664,12.146,15
.472,9.538,4.858,7.944,6.262,8.762,4.986,4.682,8.755,5.743,9.166,7.052,5.878,6.5
93,4.869,6.248,9.378,7.782,4.755,7.772,9.196,4.637,9.494,4.641,8.196,9.898,6.359
,4.601,4.073,4.176,4.069,6.296,7.706,5.234,6.894,8.472,4.025,4.169,12.49,4.421,5
.369,5.496,3.642,6.848,4.512,4.318,5.902,5.935,6.811,5.033,6.905,6.521,6.124,7.4
22,8.146,5.569,5.962,6.629,4.716,5.697,4.373,5.586,8.825,8.312,5.488,5.076,5.984
,6.547,7.458,3.999,4.389,6.665,5.083,4.253,3.989,7.914,4.591,8.043,5.792,4.468,6
.144,5.58,4.75,9.08,6.74,8.6,8.9,11.844,11.028,12.864,10.212,13.056,5.746,4.994,
7.066,4.968,4.296,6.242,4.066,8.482,7.006,7.212,6.312,9.754,6.828,5.952,6.444,6.
638,11.28,7.354,5.784,5.174,5.786,6.166,6.036,4.598,5.666,8.17,5.09,9.05,11.838,
13.413,18.762,16.503,14.082,16.389,10.071,10.253,10.941,10.703,10.702,10.116,11.
205,12.333,10.231,11.691,12.847,12.521,12.919,10.613,15.384,15.387,10.491,13.087
,13.587,12.093,12.969,12.255,12.033,10.437,17.472,17.997,14.433,5.853,6.288,9.53
5,7.387,7.679,4.665,7.678,7.531,7.532,6.579,6.261,7.267,4.683,5.883,7.555,5.095,
4.671,7.193,6.345,6.342,14.55,7.784,8.779,7.783,7.049,4.605,7.977,5.817,4.644,5.
634,6.355,4.725,7.789,3.916,3.917,5.841,5.641,5.642,7.737,9.201,10.95,7.761,10.8
9,7.751,9.225,4.698,6.699,7.721,7.608,7.601,4.449,6.791,4.413,6.783,4.028,4.027,
8.201,6.775,8.488,8.489,6.969,8.595,6.851,5.121,5.901,6.821,6.504,7.311,6.713,5.
532,8.934,8.143,4.903,4.712,4.711,4.707,6.433,6.434,8.943,9.859,8.274,7.248,4.91
7,7.437,7.446,8.542,8.543,5.481,8.823,5.373,9.828,5.272,5.273,8.604,4.377,4.578,
9.024,5.682,7.452,8.309,3.992,8.835,3.991,8.214,9.009,5.088,6.029,6.951,4.586,7.
362,4.585,6.531,7.329,9.924,8.717,6.682,6.683,6.141,7.347,5.064,8.226,8.46,9.86,
7.79,7.75,5.43,4.86,7.05,6.24,4.62,8.73,10.5,6.79,6.84,6.85,12.496,17.308,22.128
,14.004,14.804,26.672,16.312,13.634,11.148,10.892,11.132,11.944,13.544,15.956,11
.158,14.268,21.108,10.084,10.864,10.068,26.464,10.172,16.252,10.846,10.016,13.00
4,14.672,19.528,10.744,15.544,10.332,17.812,12.828,17.864,13.272,10.492,12.292,1
8.952,18.148,14.452,12.806,13.896,14.204,15.008,13.252,24.344,11.492,13.196,12.2
64,14.476,12.136,14.516,10.436,11.366,14.416,11.648,13.352,15.044,14.836,13.864,
13.224,14.436,17.356,12.172,7.922,8.732,4.664,7.932,7.274,6.374,7.174,4.684,5.88
4,8.852,6.448,9.164,9.646,9.652,7.588,14.59,5.822,6.236,9.768,8.186,5.624,6.228,
4.754,7.032,4.638,4.642,7.028,6.358,7.296,9.794,10.36,12.94,7.736,7.726,10.94,5.
402,9.356,4.602,7.748,9.244,9.218,9.118,21.02,9.116,9.124,17.36,8.446,12.28,8.36
8,6.784,4.026,5.212,18.04,8.336,15.26,5.362,8.396,4.402,15.42,6.878,8.588,15.66,
7.306,9.864,7.406,3.694,4.824,5.138,8.556,6.522,6.842,7.432,4.808,8.944,9.756,5.
698,6.554,4.374,8.824,5.978,7.348,8.836,8.832,22.32,3.998,6.952,7.364,7.468,7.32
8,4.254,8.042,17.18,6.546,6.544,6.048,6.148,17.06,8.32,13.5,9.48,9.42,7.96,5.34,
5.37,4.39,4.07,5.57,6.02,4.87,6.14,6.22,6.28,9.02,6.31,6.58,6.88,6.81,5.3,5.2,8.
4,9.1,17.448,21.486,15.096,13.635,19.266,13.566,16.074,26.028,11.952,14.256,28.8
12,11.157,18.114,16.992,29.196,29.082,16.158,20.406,30.048,10.845,31.314,10.002,
10.206,12.312,19.422,10.338,19.686,19.404,12.642,11.538,14.586,14.589,13.968,23.
352,13.974,15.774,12.807,10.482,18.918,10.668,23.952,12.252,14.334,23.976,18.996
,11.367,15.246,16.686,10.698,33.48,7.923,5.747,9.522,7.275,4.993,6.375,7.173,20.
64,7.067,27.87,9.645,6.241,8.171,7.128,8.187,3.902,5.823,7.007,20.97,9.795,7.872
,7.725,5.403,27.21,7.818,9.219,4.067,9.117,11.73,31.02,8.483,5.361,4.401,6.879,8
.166,7.305,6.309,7.407,3.693,22.05,9.049,6.324,6.843,9.755,6.637,6.555,5.979,7.3
55,5.785,5.173,6.167,5.089,4.597,5.665,8.445,9.612,21.9,13.8,25.8,9.72,6.48,42.7
56,10.252,53.544,53.028,10.232,12.848,10.612,13.088,37.392,54.684,7.388,9.536,7.
268,8.754,5.096,5.742,7.192,6.594,7.048,6.356,12.92,4.074,4.422,6.776,12.52,5.93
4,5.034,6.712,6.906,8.144,4.904,5.162,8.826,8.308,6.666,6.028,5.082,8.716,7.72,3
.99,45.9,4.17,4.59,8.78,6.63,6.82,7.6,8.2,-0]
}
```

9/9

B

SAMPLE LAYOUTS

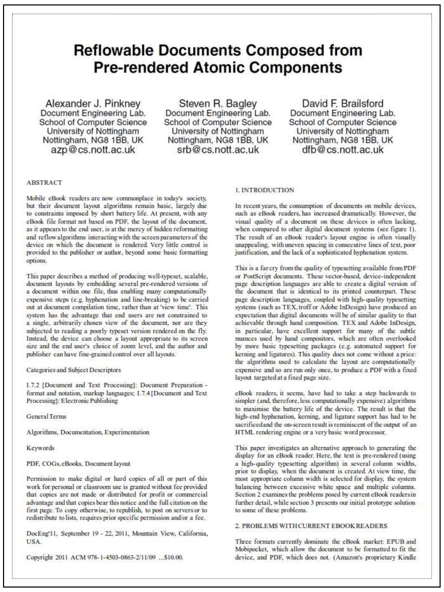
All the layouts in this section are of the author's 2011 paper, *Reflowable Documents Composed from Prerendered Atomic Components* [PBB11].

All the Malleable Document System layouts use the same selection of galley renderings as were used in Rendering D of the user study — see Table 2 on page 79.

B.1 LAYOUT BY THE MALLEABLE DOCUMENT SYSTEM

B.1.1 *Rendered by Mozilla Firefox on a PC*

[PBB11] laid out by the malleable document system, running in Mozilla Firefox on a PC. The page size has been selected to resemble that of A4 paper in a portrait orientation.



Reflowable Documents Composed from Pre-rendered Atomic Components

Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

David F. Brailsford
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk

ABSTRACT

Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to the publisher or author, beyond some basic formatting options.

This paper describes a method of producing well-typeset, scalable, document layouts by embedding several pre-rendered versions of a document within one file, thus enabling many computationally expensive steps (e.g. hyphenation and line-breaking) to be carried out at document compilation time, rather than at 'view time'. This system has the advantage that end users are not constrained to a single, arbitrarily chosen view of the document, nor are they subjected to reading a poorly typeset version rendered on the fly. Instead, the device can choose a layout appropriate to its screen size and the end user's choice of zoom level, and the author and publisher can have fine-grained control over all layouts.

Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation - format and notation, markup languages; I.7.4 [Document and Text Processing]: Electronic Publishing

General Terms

Algorithms, Documentation, Experimentation

Keywords

PDF, COGs, eBooks, Document layout

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DocEng'11, September 19 - 22, 2011, Mountain View, California, USA.

Copyright 2011 ACM 978-1-4503-0863-2/11/09 ...$10.00.

1. INTRODUCTION

In recent years, the consumption of documents on mobile devices, such as eBook readers, has increased dramatically. However, the visual quality of a document on these devices is often lacking, when compared to other digital document systems (see figure 1). The result of an eBook reader's layout engine is often visually unappealing, with uneven spacing in consecutive lines of text, poor justification, and the lack of a sophisticated hyphenation system.

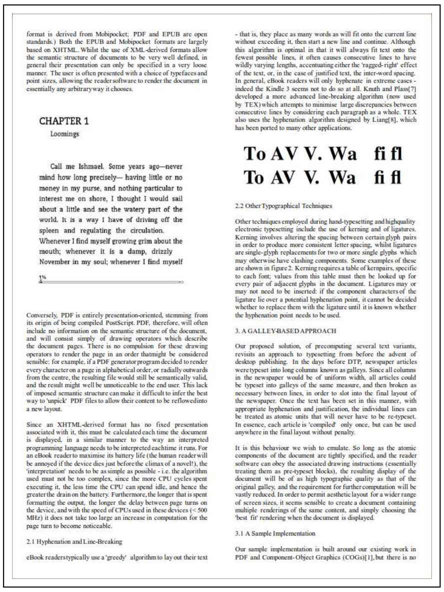
This is a far cry from the quality of typesetting available from PDF or PostScript documents. These vector-based, device-independent page description languages are able to create a digital version of the document that is identical to its printed counterpart. These page description languages, coupled with high-quality typesetting systems (such as TEX, troff or Adobe InDesign) have produced an expectation that digital documents will be of similar quality to that achievable through hand composition. TEX and Adobe InDesign, in particular, have excellent support for many of the subtle nuances used by hand compositors, which are often overlooked by more basic typesetting packages (e.g. automated support for kerning and ligatures). This quality does not come without a price: the algorithms used to calculate the layout are computationally expensive and so are run only once, to produce a PDF with a fixed layout targeted at a fixed page size.

eBook readers, it seems, have had to take a step backwards to simpler (and, therefore, less computationally expensive) algorithms to maximise the battery life of the device. The result is that the high-end hyphenation, kerning, and ligature support has had to be sacrificedand the on-screen result is reminiscent of the output of an HTML rendering engine or a very basic word processor.

This paper investigates an alternative approach to generating the display for an eBook reader. Here, the text is pre-rendered (using a high-quality typesetting algorithm) in several column widths, prior to display, when the document is created. At view time, the most appropriate column width is selected for display, the system balancing between excessive white space and multiple columns. Section 2 examines the problems posed by current eBook readers in further detail, while section 3 presents our initial prototype solution to some of these problems.

2. PROBLEMS WITH CURRENT EBOOK READERS

Three formats currently dominate the eBook market: EPUB and Mobipocket, which allow the document to be formatted to fit the device, and PDF, which does not. (Amazon's proprietary Kindle



format is derived from Mobipocket; PDF and EPUB are open standards.) Both the EPUB and Mobipocket formats are largely based on XHTML. Whilst the use of XML-derived formats allow the semantic structure of documents to be very well defined, in general their presentation can only be specified in a very loose manner. The user is often presented with a choice of typefaces and point sizes, allowing the reader software to render the document in essentially any arbitrary way it chooses.

CHAPTER 1

Loomings

Call me Ishmael. Some years ago—never mind how long precisely— having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself

1%

Conversely, PDF is entirely presentation-oriented, stemming from its origin of being compiled PostScript. PDF, therefore, will often include no information on the semantic structure of the document, and will consist simply of drawing operators which describe the document pages. There is no compulsion for these drawing operators to render the page in an order thatmight be considered sensible: for example, if a PDF generator program decided to render every character on a page in alphabetical order, or radially outwards from the centre, the resulting file would still be semantically valid, and the result might well be unnoticeable to the end user. This lack of imposed semantic structure can make it difficult to infer the best way to 'unpick' PDF files to allow their content to be reflowedinto a new layout.

Since an XHTML-derived format has no fixed presentation associated with it, this must be calculated each time the document is displayed, in a similar manner to the way an interpreted programming language needs to be interpreted eachtime it runs. For an eBook reader to maximise its battery life (the human reader will be annoyed if the device dies just before the climax of a novel!), the 'interpretation' needs to be as simple as possible - i.e. the algorithm used must not be too complex, since the more CPU cycles spent executing it, the less time the CPU can spend idle, and hence the greater the drain on the battery. Furthermore, the longer that is spent formatting the output, the longer the delay between page turns on the device, and with the speed of CPUs used in these devices (< 500 MHz) it does not take too large an increase in computation for the page turn to become noticeable.

2.1 Hyphenation and Line-Breaking

eBook readerstypically use a 'greedy' algorithm to lay out their text - that is, they place as many words as will fit onto the current line without exceeding it, then start a new line and continue. Although this algorithm is optimal in that it will always fit text onto the fewest possible lines, it often causes consecutive lines to have wildly varying lengths, accentuating either the 'ragged-right' effect of the text, or, in the case of justified text, the inter-word spacing. In general, eBook readers will only hyphenate in extreme cases - indeed the Kindle 3 seems not to do so at all. Knuth and Plass[7] developed a more advanced line-breaking algorithm (now used by TEX) which attempts to minimise large discrepancies between consecutive lines by considering each paragraph as a whole. TEX also uses the hyphenation algorithm designed by Liang[8], which has been ported to many other applications.

To AV V. Wa fi fl
To AV V. Wa fi fl

2.2 Other Typographical Techniques

Other techniques employed during hand-typesetting and highquality electronic typesetting include the use of kerning and of ligatures. Kerning involves altering the spacing between certain glyph pairs in order to produce more consistent letter spacing, whilst ligatures are single-glyph replacements for two or more single glyphs which may otherwise have clashing components. Some examples of these are shown in figure 2. Kerning requires a table of kernpairs, specific to each font; values from this table must then be looked up for every pair of adjacent glyphs in the document. Ligatures may or may not need to be inserted: if the component characters of the ligature lie over a potential hyphenation point, it cannot be decided whether to replace them with the ligature until it is known whether the hyphenation point needs to be used.

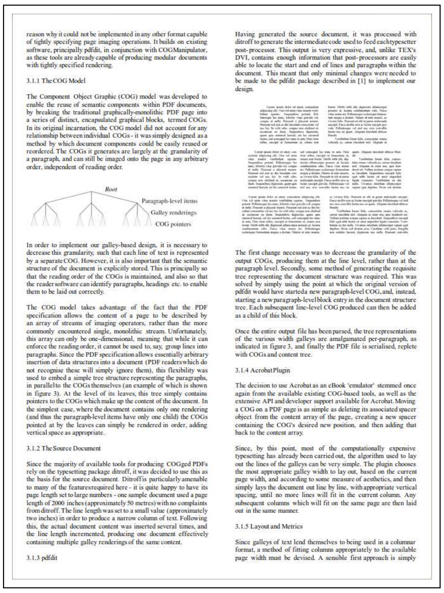
3. A GALLEY-BASED APPROACH

Our proposed solution, of precomputing several text variants, revisits an approach to typesetting from before the advent of desktop publishing. In the days before DTP, newspaper articles were typeset into long columns known as galleys. Since all columns in the newspaper would be of uniform width, all articles could be typeset into galleys of the same measure, and then broken as necessary between lines, in order to slot into the final layout of the newspaper. Once the text has been set in this manner, with appropriate hyphenation and justification, the individual lines can be treated as atomic units that will never have to be re-typeset. In essence, each article is 'compiled' only once, but can be used anywhere in the final layout without penalty.

It is this behaviour we wish to emulate. So long as the atomic components of the document are tightly specified, and the reader software can obey the associated drawing instructions (essentially treating them as pre-typeset blocks), the resulting display of the document will be of as high typographic quality as that of the original galley, and the requirement for further computation will be vastly reduced. In order to permit aesthetic layout for a wider range of screen sizes, it seems sensible to create a document containing multiple renderings of the same content, and simply choosing the 'best fit' rendering when the document is displayed.

3.1 A Sample Implementation

Our sample implementation is built around our existing work in PDF and Component-Object Graphics (COGs)[1], but there is no



reason why it could not be implemented in any other format capable of tightly specifying page imaging operations. It builds on existing software, principally pdfdit, in conjunction with COGManipulator, as these tools are already capable of producing modular documents with tightly specified rendering.

3.1.1 The COG Model

The Component Object Graphic (COG) model was developed to enable the reuse of semantic components within PDF documents, by breaking the traditional graphically-monolithic PDF page into a series of distinct, encapsulated graphical blocks, termed COGs. In its original incarnation, the COG model did not account for any relationship between individual COGs - it was simply designed as a method by which document components could be easily reused or reordered. The COGs it generates are largely at the granularity of a paragraph, and can still be imaged onto the page in any arbitrary order, independent of reading order.

In order to implement our galley-based design, it is necessary to decrease this granularity, such that each line of text is represented by a separate COG. However, it is also important that the semantic structure of the document is explicitly stored. This is principally so that the reading order of the COGs is maintained, and also so that the reader software can identify paragraphs, headings etc. to enable them to be laid out correctly.

The COG model takes advantage of the fact that the PDF specification allows the content of a page to be described by an array of streams of imaging operators, rather than the more commonly encountered single, monolithic stream. Unfortunately, this array can only be one-dimensional, meaning that while it can enforce the reading order, it cannot be used to, say, group lines into paragraphs. Since the PDF specification allows essentially arbitrary insertion of data structures into a document (PDF readers which do not recognise these will simply ignore them), this flexibility was used to embed a simple tree structure representing the paragraphs, in parallel to the COGs themselves (an example of which is shown in figure 3). At the level of its leaves, this tree simply contains pointers to the COGs which make up the content of the document. In the simplest case, where the document contains only one rendering (and thus the paragraph-level items have only one child) the COGs pointed at by the leaves can simply be rendered in order, adding vertical space as appropriate.

3.1.2 The Source Document

Since the majority of available tools for producing COGged PDFs rely on the typesetting package ditroff, it was decided to use this as the basis for the source document. Ditroff is particularly amenable to many of the featuresrequired here - it is quite happy to have its page length set to large numbers - one sample document used a page length of 2000 inches (approximately 50 metres) with no complaints from ditroff. The line length was set to a small value (approximately two inches) in order to produce a narrow column of text. Following this, the actual document content was inserted several times, and the line length incremented, producing one document effectively containing multiple galley renderings of the same content.

3.1.3 pdfdit

Having generated the source document, it was processed with ditroff to generate the intermediate code used to feed eachtypesetter post-processor. This output is very expressive, and, unlike TEX's DVI, contains enough information that post-processors are easily able to locate the start and end of lines and paragraphs within the document. This meant that only minimal changes were needed to be made to the pdfdit package described in [1] to implement our design.

The first change necessary was to decrease the granularity of the output COGs, producing them at the line level, rather than at the paragraph level. Secondly, some method of generating the requisite tree representing the document structure was required. This was solved by simply using the point at which the original version of pdfdit would have starteda new paragraph-level COG, and, instead, starting a new paragraph-level block entry in the document structure tree. Each subsequent line-level COG produced can then be added as a child of this block.

Once the entire output file has been parsed, the tree representations of the various width galleys are amalgamated per-paragraph, as indicated in figure 3, and finally the PDF file is serialised, replete with COGs and content tree.

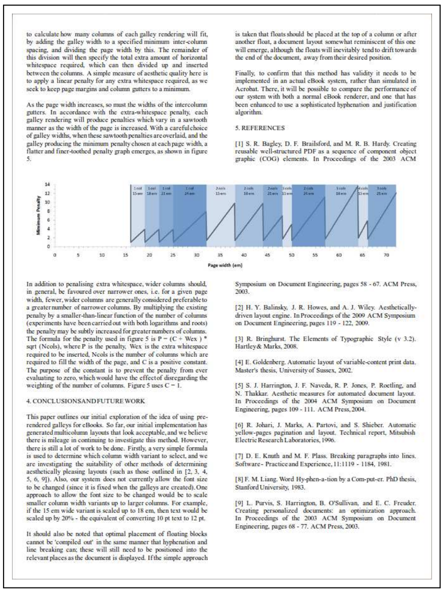
3.1.4 Acrobat Plugin

The decision to use Acrobat as an eBook 'emulator' stemmed once again from the available existing COG-based tools, as well as the extensive API and developer support available for Acrobat. Moving a COG on a PDF page is as simple as deleting its associated spacer object from the content array of the page, creating a new spacer containing the COG's desired new position, and then adding that back to the content array.

Since, by this point, most of the computationally expensive typesetting has already been carried out, the algorithm used to lay out the lines of the galleys can be very simple. The plugin chooses the most appropriate galley width to lay out, based on the current page width, and according to some measure of aesthetics, and then simply lays the document out line by line, with appropriate vertical spacing, until no more lines will fit in the current column. Any subsequent columns which will fit on the same page are then laid out in the same manner.

3.1.5 Layout and Metrics

Since galleys of text lend themselves to being used in a columnar format, a method of fitting columns appropriately to the available page width must be devised. A sensible first approach is simply



to calculate how many columns of each galley rendering will fit, by adding the galley width to a specified minimum inter-column spacing, and dividing the page width by this. The remainder of this division will then specify the total extra amount of horizontal whitespace required, which can then divided up and inserted between the columns. A simple measure of aesthetic quality here is to apply a linear penalty for any extra whitespace required, as we seek to keep page margins and column gutters to a minimum.

As the page width increases, so must the widths of the intercolumn gutters. In accordance with the extra-whitespace penalty, each galley rendering will produce penalties which vary in a sawtooth manner as the width of the page is increased. With a careful choice of galley widths, when these sawtooth penalties are overlaid, and the galley producing the minimum penalty chosen at each page width, a flatter and finer-toothed penalty graph emerges, as shown in figure 5.

In addition to penalising extra whitespace, wider columns should, in general, be favoured over narrower ones, i.e. for a given page width, fewer, wider columns are generally considered preferable to a greater number of narrower columns. By multiplying the existing penalty by a smaller-than-linear function of the number of columns (experiments have been carried out with both logarithms and roots) the penalty may be subtly increased for greater numbers of columns. The formula for the penalty used in figure 5 is P = (C + Wex) * sqrt (Ncols), where P is the penalty, Wex is the extra whitespace required to be inserted, Ncols is the number of columns which are required to fill the width of the page, and C is a positive constant. The purpose of the constant is to prevent the penalty from ever evaluating to zero, which would have the effect of disregarding the weighting of the number of columns. Figure 5 uses C = 1.

4. CONCLUSIONS AND FUTURE WORK

This paper outlines our initial exploration of the idea of using pre-rendered galleys for eBooks. So far, our initial implementation has generated multicolumn layouts that look acceptable, and we believe there is mileage in continuing to investigate this method. However, there is still a lot of work to be done. Firstly, a very simple formula is used to determine which column width variant to select, and we are investigating the suitability of other methods of determining aesthetically pleasing layouts (such as those outlined in [2, 3, 4, 5, 6, 9]). Also, our system does not currently allow the font size to be changed (since it is fixed when the galleys are created). One approach to allow the font size to be changed would be to scale smaller column width variants up to larger columns. For example, if the 15 em wide variant is scaled up to 18 em, then text would be scaled up by 20% - the equivalent of converting 10 pt text to 12 pt.

It should also be noted that optimal placement of floating blocks cannot be 'compiled out' in the same manner that hyphenation and line breaking can; these will still need to be positioned into the relevant places as the document is displayed. If the simple approach is taken that floats should be placed at the top of a column or after another float, a document layout somewhat reminiscent of this one will emerge, although the floats will inevitably tend to drift towards the end of the document, away from their desired position.

Finally, to confirm that this method has validity it needs to be implemented in an actual eBook system, rather than simulated in Acrobat. There, it will be possible to compare the performance of our system with both a normal eBook renderer, and one that has been enhanced to use a sophisticated hyphenation and justification algorithm.

5. REFERENCES

[1] S. R. Bagley, D. F. Brailsford, and M. R. B. Hardy. Creating reusable well-structured PDF as a sequence of component object graphic (COG) elements. In Proceedings of the 2003 ACM Symposium on Document Engineering, pages 58 - 67. ACM Press, 2003.

[2] H. Y. Balinsky, J. R. Howes, and A. J. Wiley. Aesthetically-driven layout engine. In Proceedings of the 2009 ACM Symposium on Document Engineering, pages 119 - 122, 2009.

[3] R. Bringhurst. The Elements of Typographic Style (v 3.2). Hartley & Marks, 2008.

[4] E. Goldenberg. Automatic layout of variable-content print data. Master's thesis, University of Sussex, 2002.

[5] S. J. Harrington, J. F. Naveda, R. P. Jones, P. Roetling, and N. Thakkar. Aesthetic measures for automated document layout. In Proceedings of the 2004 ACM Symposium on Document Engineering, pages 109 - 111. ACM Press, 2004.

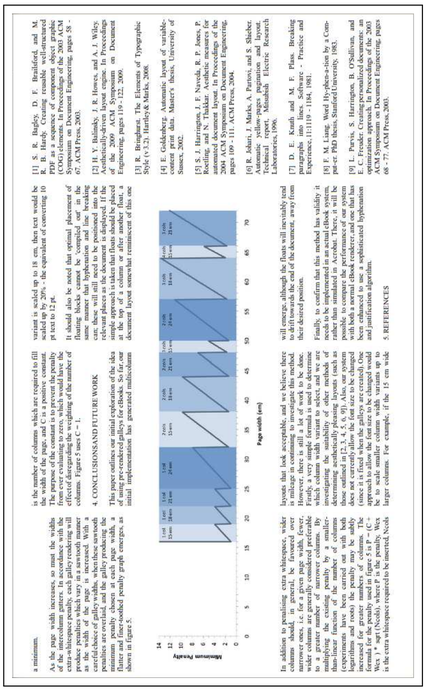
[6] R. Johari, J. Marks, A. Partovi, and S. Shieber. Automatic yellow-pages pagination and layout. Technical report, Mitsubish Electric Research Laboratories, 1996.

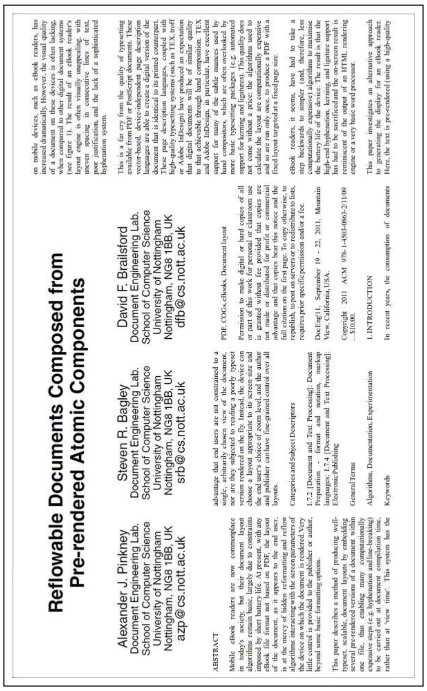
[7] D. E. Knuth and M. F. Plass. Breaking paragraphs into lines. Software- Practice and Experience, 11:1119 - 1184, 1981.

[8] F. M. Liang. Word Hy-phen-a-tion by a Com-put-er. PhD thesis, Stanford University, 1983.

[9] L. Purvis, S. Harrington, B. O'Sullivan, and E. C. Freuder. Creating personalized documents: an optimization approach. In Proceedings of the 2003 ACM Symposium on Document Engineering, pages 68 - 77. ACM Press, 2003.

[PBB11] laid out by the malleable document system, running in Mozilla Firefox on a PC. The page size has been selected to resemble that of A4 paper in a landscape orientation.



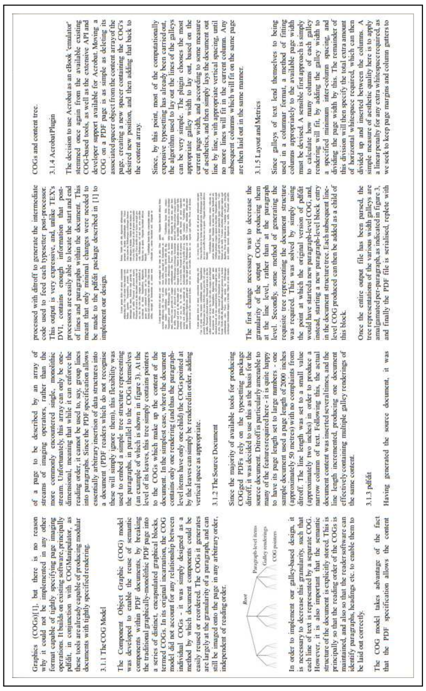


Reflowable Documents Composed from Pre-rendered Atomic Components

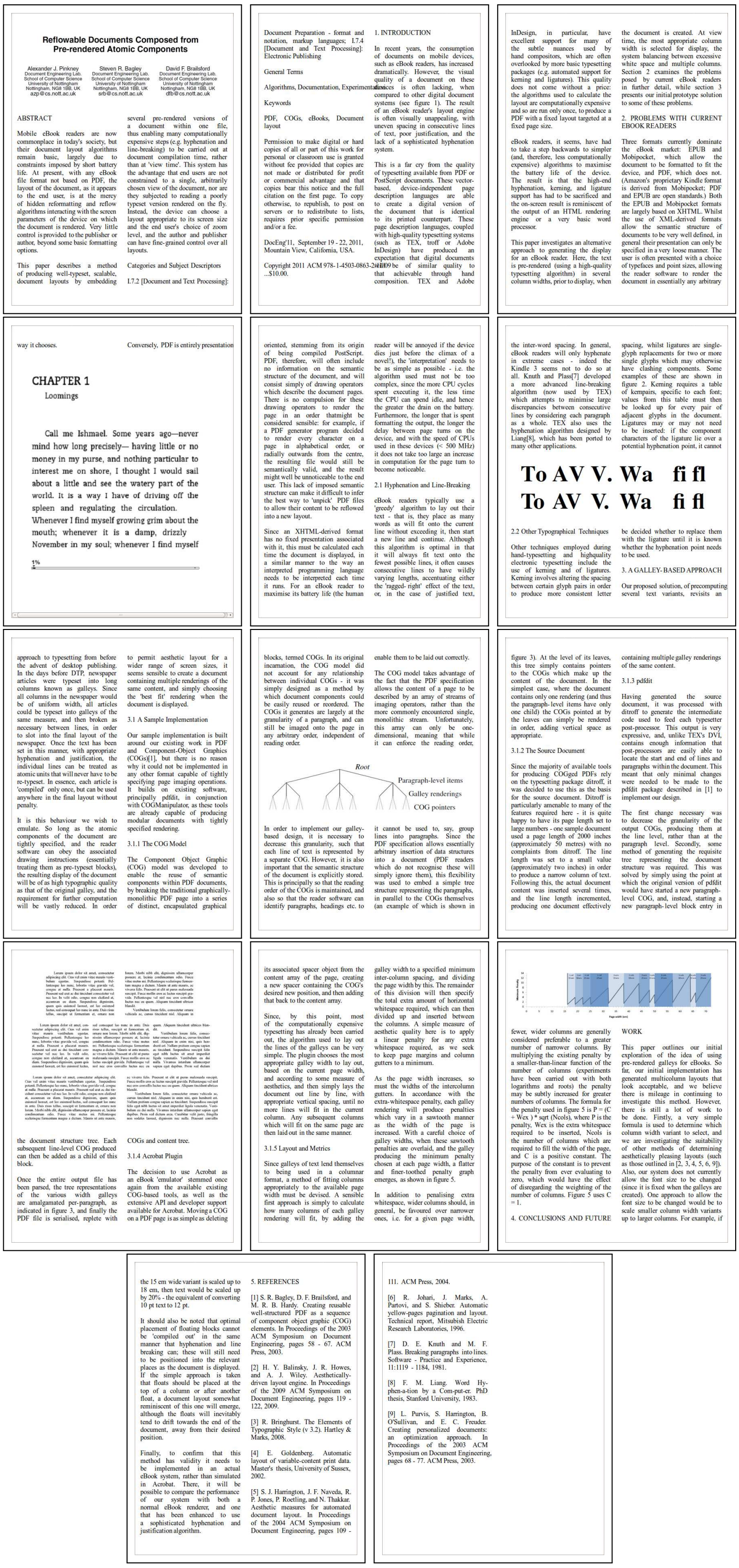
Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

David F. Brailsford
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk



[PBB11] laid out by the malleable document system, running in Mozilla Firefox on a PC. The page size has been selected to resemble that of an ebook reader in a portrait orientation.



Reflowable Documents Composed from Pre-rendered Atomic Components

Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

David F. Brailsford
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk

ABSTRACT

Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to the publisher or author, beyond some basic formatting options.

This paper describes a method of producing well-typeset, scalable, document layouts by embedding several pre-rendered versions of a document within one file, thus enabling many computationally expensive steps (e.g. hyphenation and line-breaking) to be carried out at document compilation time, rather than at 'view time'. This system has the advantage that end users are not constrained to a single, arbitrarily chosen view of the document, nor are they subjected to reading a poorly typeset version rendered on the fly. Instead, the device can choose a layout appropriate to its screen size and the end user's choice of zoom level, and the author and publisher can have fine-grained control over all layouts.

Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation - format and notation, markup languages; I.7.4 [Document and Text Processing]: Electronic Publishing

General Terms

Algorithms, Documentation, Experimentation

Keywords

PDF, COGs, eBooks, Document layout

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DocEng'11, September 19 - 22, 2011, Mountain View, California, USA.

Copyright 2011 ACM 978-1-4503-0863-2/11/09 ...$10.00.

1. INTRODUCTION

In recent years, the consumption of documents on mobile devices, such as eBook readers, has increased dramatically. However, the visual quality of a document on these devices is often lacking, when compared to other digital document systems (see figure 1). The result of an eBook reader's layout engine is often visually unappealing, with uneven spacing in consecutive lines of text, poor justification, and the lack of a sophisticated hyphenation system.

This is a far cry from the quality of typesetting available from PDF or PostScript documents. These vector-based, device-independent page description languages are able to create a digital version of the document that is identical to its printed counterpart. These page description languages, coupled with high-quality typesetting systems (such as TEX, troff or Adobe InDesign) have produced an expectation that digital documents will be of similar quality to that achievable through hand composition. TEX and Adobe InDesign, in particular, have excellent support for many of the subtle nuances used by hand compositors, which are often overlooked by more basic typesetting packages (e.g. automated support for kerning and ligatures). This quality does not come without a price: the algorithms used to calculate the layout are computationally expensive and so are run only once, to produce a PDF with a fixed layout targeted at a fixed page size.

eBook readers, it seems, have had to take a step backwards to simpler (and, therefore, less computationally expensive) algorithms to maximise the battery life of the device. The result is that the high-end hyphenation, kerning, and ligature support has had to be sacrificed and the on-screen result is reminiscent of the output of an HTML rendering engine or a very basic word processor.

This paper investigates an alternative approach to generating the display for an eBook reader. Here, the text is pre-rendered (using a high-quality typesetting algorithm) in several column widths, prior to display, when the document is created. At view time, the most appropriate column width is selected for display, the system balancing between excessive white space and multiple columns. Section 2 examines the problems posed by current eBook readers in further detail, while section 3 presents our initial prototype solution to some of these problems.

2. PROBLEMS WITH CURRENT EBOOK READERS

Three formats currently dominate the eBook market: EPUB and Mobipocket, which allow the document to be formatted to fit the device, and PDF, which does not. (Amazon's proprietary Kindle format is derived from Mobipocket; PDF and EPUB are open standards.) Both the EPUB and Mobipocket formats are largely based on XHTML. Whilst the use of XML-derived formats allow the semantic structure of documents to be very well defined, in general their presentation can only be specified in a very loose manner. The user is often presented with a choice of typefaces and point sizes, allowing the reader software to render the document in essentially any arbitrary way it chooses.

CHAPTER 1
Loomings

Call me Ishmael. Some years ago—never mind how long precisely— having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself

1%

Conversely, PDF is entirely presentation oriented, stemming from its origin of being compiled PostScript. PDF, therefore, will often include no information on the semantic structure of the document, and will consist simply of drawing operators which describe the document pages. There is no compulsion for these drawing operators to render the page in an order thatmight be considered sensible: for example, if a PDF generator program decided to render every character on a page in alphabetical order, or radially outwards from the centre, the resulting file would still be semantically valid, and the result might well be unnoticeable to the end user. This lack of imposed semantic structure can make it difficult to infer the best way to 'unpick' PDF files to allow their content to be reflowed into a new layout.

Since an XHTML-derived format has no fixed presentation associated with it, this must be calculated each time the document is displayed, in a similar manner to the way an interpreted programming language needs to be interpreted each time it runs. For an eBook reader to maximise its battery life (the human reader will be annoyed if the device dies just before the climax of a novel!), the 'interpretation' needs to be as simple as possible - i.e. the algorithm used must not be too complex, since the more CPU cycles spent executing it, the less time the CPU can spend idle, and hence the greater the drain on the battery. Furthermore, the longer that is spent formatting the output, the longer the delay between page turns on the device, and with the speed of CPUs used in these devices (< 500 MHz) it does not take too large an increase in computation for the page turn to become noticeable.

2.1 Hyphenation and Line-Breaking

eBook readers typically use a 'greedy' algorithm to lay out their text - that is, they place as many words as will fit onto the current line without exceeding it, then start a new line and continue. Although this algorithm is optimal in that it will always fit text onto the fewest possible lines, it often causes consecutive lines to have wildly varying lengths, accentuating either the 'ragged- right' effect of the text, or, in the case of justified text, the inter-word spacing. In general, eBook readers will only hyphenate in extreme cases - indeed the Kindle 3 seems not to do so at all. Knuth and Plass[7] developed a more advanced line-breaking algorithm (now used by TEX) which attempts to minimise large discrepancies between consecutive lines by considering each paragraph as a whole. TEX also uses the hyphenation algorithm designed by Liang[8], which has been ported to many other applications.

To AV V. Wa fi fl
To AV V. Wa fi fl

2.2 Other Typographical Techniques

Other techniques employed during hand-typesetting and highquality electronic typesetting include the use of kerning and of ligatures. Kerning involves altering the spacing between certain glyph pairs in order to produce more consistent letter spacing, whilst ligatures are single-glyph replacements for two or more single glyphs which may otherwise have clashing components. Some examples of these are shown in figure 2. Kerning requires a table of kernpairs, specific to each font; values from this table must then be looked up for every pair of adjacent glyphs in the document. Ligatures may or may not need to be inserted: if the component characters of the ligature lie over a potential hyphenation point, it cannot be decided whether to replace them with the ligature until it is known whether the hyphenation point needs to be used.

3. A GALLEY- BASED APPROACH

Our proposed solution, of precomputing several text variants, revisits an approach to typesetting from before the advent of desktop publishing. In the days before DTP, newspaper articles were typeset into long columns known as galleys. Since all columns in the newspaper would be of uniform width, all articles could be typeset into galleys of the same measure, and then broken as necessary between lines, in order to slot into the final layout of the newspaper. Once the text has been set in this manner, with appropriate hyphenation and justification, the individual lines can be treated as atomic units that will never have to be re-typeset. In essence, each article is 'compiled' only once, but can be used anywhere in the final layout without penalty.

It is this behaviour we wish to emulate. So long as the atomic components of the document are tightly specified, and the reader software can obey the associated drawing instructions (essentially treating them as pre-typeset blocks), the resulting display of the document will be of as high typographic quality as that of the original galley, and the requirement for further computation will be vastly reduced. In order to permit aesthetic layout for a wider range of screen sizes, it seems sensible to create a document containing multiple renderings of the same content, and simply choosing the 'best fit' rendering when the document is displayed.

3.1 A Sample Implementation

Our sample implementation is built around our existing work in PDF and Component-Object Graphics (COGs)[1], but there is no reason why it could not be implemented in any other format capable of tightly specifying page imaging operations. It builds on existing software, principally pdfdit, in conjunction with COGManipulator, as these tools are already capable of producing modular documents with tightly specified rendering.

3.1.1 The COG Model

The Component Object Graphic (COG) model was developed to enable the reuse of semantic components within PDF documents, by breaking the traditional graphically-monolithic PDF page into a series of distinct, encapsulated graphical blocks, termed COGs. In its original incarnation, the COG model did not account for any relationship between individual COGs - it was simply designed as a method by which document components could be easily reused or reordered. The COGs it generates are largely at the granularity of a paragraph, and can still be imaged onto the page in any arbitrary order, independent of reading order.

In order to implement our galley-based design, it is necessary to decrease this granularity, such that each line of text is represented by a separate COG. However, it is also important that the semantic structure of the document is explicitly stored. This is principally so that the reading order of the COGs is maintained, and also so that the reader software can identify paragraphs, headings etc. to enable them to be laid out correctly.

The COG model takes advantage of the fact that the PDF specification allows the content of a page to be described by an array of streams of imaging operators, rather than the more commonly encountered single, monolithic stream. Unfortunately, this array can only be one-dimensional, meaning that while it can enforce the reading order, it cannot be used to, say, group lines into paragraphs. Since the PDF specification allows essentially arbitrary insertion of data structures into a document (PDF readers which do not recognise these will simply ignore them), this flexibility was used to embed a simple tree structure representing the paragraphs, in parallel to the COGs themselves (an example of which is shown in figure 3). At the level of its leaves, this tree simply contains pointers to the COGs which make up the content of the document. In the simplest case, where the document contains only one rendering (and thus the paragraph- level items have only one child) the COGs pointed at by the leaves can simply be rendered in order, adding vertical space as appropriate.

3.1.2 The Source Document

Since the majority of available tools for producing COGged PDFs rely on the typesetting package ditroff, it was decided to use this as the basis for the source document. Ditroff is particularly amenable to many of the features required here - it is quite happy to have its page length set to large numbers - one sample document used a page length of 2000 inches (approximately 50 metres) with no complaints from ditroff. The line length was set to a small value (approximately two inches) in order to produce a narrow column of text. Following this, the actual document content was inserted several times, and the line length incremented, producing one document effectively containing multiple galley renderings of the same content.

3.1.3 pdfdit

Having generated the source document, it was processed with ditroff to generate the intermediate code used to feed each typesetter post-processor. This output is very expressive, and, unlike TEX's DVI, contains enough information that post-processors are easily able to locate the start and end of lines and paragraphs within the document. This meant that only minimal changes were needed to be made to the pdfdit package described in [1] to implement our design.

The first change necessary was to decrease the granularity of the output COGs, producing them at the line level, rather than at the paragraph level. Secondly, some method of generating the requisite tree representing the document structure was required. This was solved by simply using the point at which the original version of pdfdit would have started a new paragraph-level COG, and, instead, starting a new paragraph- level block entry in the document structure tree. Each subsequent line-level COG produced can then be added as a child of this block.

Once the entire output file has been parsed, the tree representations of the various width galleys are amalgamated per-paragraph, as indicated in figure 3, and finally the PDF file is serialised, replete with COGs and content tree.

3.1.4 Acrobat Plugin

The decision to use Acrobat as an eBook 'emulator' stemmed once again from the available existing COG-based tools, as well as the extensive API and developer support available for Acrobat. Moving a COG on a PDF page is as simple as deleting its associated spacer object from the content array of the page, creating a new spacer containing the COG's desired new position, and then adding that back to the content array.

Since, by this point, most of the computationally expensive typesetting has already been carried out, the algorithm used to lay out the lines of the galleys can be very simple. The plugin chooses the most appropriate galley width to lay out, based on the current page width, and according to some measure of aesthetics, and then simply lays the document out line by line, with appropriate vertical spacing, until no more lines will fit in the current column. Any subsequent columns which will fit on the same page are then laid out in the same manner.

3.1.5 Layout and Metrics

Since galleys of text lend themselves to being used in a columnar format, a method of fitting columns appropriately to the available page width must be devised. A sensible first approach is simply to calculate how many columns of each galley rendering will fit, by adding the galley width to a specified minimum inter-column spacing, and dividing the page width by this. The remainder of this division will then specify the total extra amount of horizontal whitespace required, which can then divided up and inserted between the columns. A simple measure of aesthetic quality here is to apply a linear penalty for any extra whitespace required, as we seek to keep page margins and column gutters to a minimum.

As the page width increases, so must the widths of the intercolumn gutters. In accordance with the extra-whitespace penalty, each galley rendering will produce penalties which vary in a sawtooth manner as the width of the page is increased. With a careful choice of galley widths, when these sawtooth penalties are overlaid, and the galley producing the minimum penalty chosen at each page width, a flatter and finer-toothed penalty graph emerges, as shown in figure 5.

In addition to penalising extra whitespace, wider columns should, in general, be favoured over narrower ones, i.e. for a given page width, fewer, wider columns are generally considered preferable to a greater number of narrower columns. By multiplying the existing penalty by a smaller-than-linear function of the number of columns (experiments have been carried out with both logarithms and roots) the penalty may be subtly increased for greater numbers of columns. The formula for the penalty used in figure 5 is P = (C + Wex) * sqrt (Ncols), where P is the penalty, Wex is the extra whitespace required to be inserted, Ncols is the number of columns which are required to fill the width of the page, and C is a positive constant. The purpose of the constant is to prevent the penalty from ever evaluating to zero, which would have the effect of disregarding the weighting of the number of columns. Figure 5 uses C = 1.

4. CONCLUSIONS AND FUTURE WORK

This paper outlines our initial exploration of the idea of using pre-rendered galleys for eBooks. So far, our initial implementation has generated multicolumn layouts that look acceptable, and we believe there is mileage in continuing to investigate this method. However, there is still a lot of work to be done. Firstly, a very simple formula is used to determine which column width variant to select, and we are investigating the suitability of other methods of determining aesthetically pleasing layouts (such as those outlined in [2, 3, 4, 5, 6, 9]). Also, our system does not currently allow the font size to be changed (since it is fixed when the galleys are created). One approach to allow the font size to be changed would be to scale smaller column width variants up to larger columns. For example, if the 15 em wide variant is scaled up to 18 em, then text would be scaled up by 20% - the equivalent of converting 10 pt text to 12 pt.

It should also be noted that optimal placement of floating blocks cannot be 'compiled out' in the same manner that hyphenation and line breaking can; these will still need to be positioned into the relevant places as the document is displayed. If the simple approach is taken that floats should be placed at the top of a column or after another float, a document layout somewhat reminiscent of this one will emerge, although the floats will inevitably tend to drift towards the end of the document, away from their desired position.

Finally, to confirm that this method has validity it needs to be implemented in an actual eBook system, rather than simulated in Acrobat. There, it will be possible to compare the performance of our system with both a normal eBook renderer, and one that has been enhanced to use a sophisticated hyphenation and justification algorithm.

5. REFERENCES

[1] S. R. Bagley, D. F. Brailsford, and M. R. B. Hardy. Creating reusable well-structured PDF as a sequence of component object graphic (COG) elements. In Proceedings of the 2003 ACM Symposium on Document Engineering, pages 58 - 67. ACM Press, 2003.

[2] H. Y. Balinsky, J. R. Howes, and A. J. Wiley. Aesthetically-driven layout engine. In Proceedings of the 2009 ACM Symposium on Document Engineering, pages 119 - 122, 2009.

[3] R. Bringhurst. The Elements of Typographic Style (v 3.2). Hartley & Marks, 2008.

[4] E. Goldenberg. Automatic layout of variable-content print data. Master's thesis, University of Sussex, 2002.

[5] S. J. Harrington, J. F. Naveda, R. P. Jones, P. Roetling, and N. Thakkar. Aesthetic measures for automated document layout. In Proceedings of the 2004 ACM Symposium on Document Engineering, pages 109 - 111. ACM Press, 2004.

[6] R. Johari, J. Marks, A. Partovi, and S. Shieber. Automatic yellow-pages pagination and layout. Technical report, Mitsubish Electric Research Laboratories, 1996.

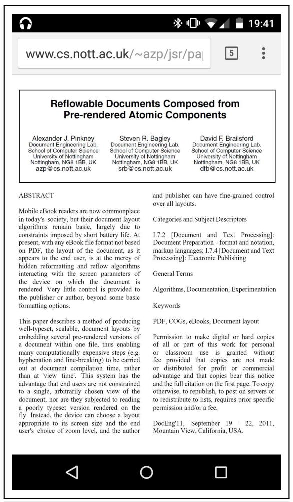
[7] D. E. Knuth and M. F. Plass. Breaking paragraphs into lines. Software - Practice and Experience, 11:1119 - 1184, 1981.

[8] F. M. Liang. Word Hy-phen-a-tion by a Com-put-er. PhD thesis, Stanford University, 1983.

[9] L. Purvis, S. Harrington, B. O'Sullivan, and E. C. Freuder. Creating personalized documents: an optimization approach. In Proceedings of the 2003 ACM Symposium on Document Engineering, pages 68 - 77. ACM Press, 2003.

B.1.2 *Chrome on an Android Phone*

[PBB11] laid out by the malleable document system, running in Chrome on an Android phone. This example shows a point size that is a little on the small side, but is still readable.



Reflowable Documents Composed from Pre-rendered Atomic Components

Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

David F. Brailsford
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk

ABSTRACT

Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to the publisher or author, beyond some basic formatting options.

This paper describes a method of producing well-typeset, scalable, document layouts by embedding several pre-rendered versions of a document within one file, thus enabling many computationally expensive steps (e.g. hyphenation and line-breaking) to be carried out at document compilation time, rather than at 'view time'. This system has the advantage that end users are not constrained to a single, arbitrarily chosen view of the document, nor are they subjected to reading a poorly typeset version rendered on the fly. Instead, the device can choose a layout appropriate to its screen size and the end user's choice of zoom level, and the author and publisher can have fine-grained control over all layouts.

Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation - format and notation, markup languages; I.7.4 [Document and Text Processing]: Electronic Publishing

General Terms

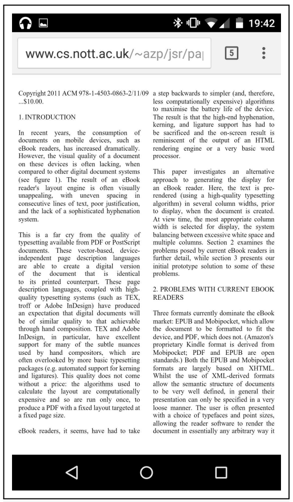
Algorithms, Documentation, Experimentation

Keywords

PDF, COGs, eBooks, Document layout

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DocEng'11, September 19 - 22, 2011, Mountain View, California, USA.



Copyright 2011 ACM 978-1-4503-0863-2/11/09 ...$10.00.

1. INTRODUCTION

In recent years, the consumption of documents on mobile devices, such as eBook readers, has increased dramatically. However, the visual quality of a document on these devices is often lacking, when compared to other digital document systems (see figure 1). The result of an eBook reader's layout engine is often visually unappealing, with uneven spacing in consecutive lines of text, poor justification, and the lack of a sophisticated hyphenation system.

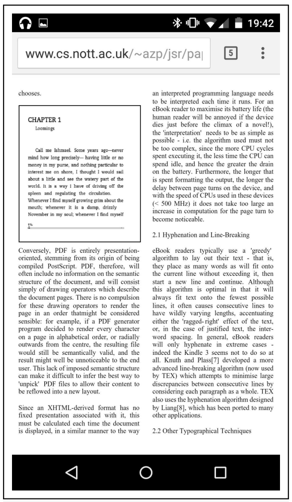
This is a far cry from the quality of typesetting available from PDF or PostScript documents. These vector-based, device-independent page description languages are able to create a digital version of the document that is identical to its printed counterpart. These page description languages, coupled with high-quality typesetting systems (such as TEX, troff or Adobe InDesign) have produced an expectation that digital documents will be of similar quality to that achievable through hand composition. TEX and Adobe InDesign, in particular, have excellent support for many of the subtle nuances used by hand compositors, which are often overlooked by more basic typesetting packages (e.g. automated support for kerning and ligatures). This quality does not come without a price: the algorithms used to calculate the layout are computationally expensive and so are run only once, to produce a PDF with a fixed layout targeted at a fixed page size.

eBook readers, it seems, have had to take a step backwards to simpler (and, therefore, less computationally expensive) algorithms to maximise the battery life of the device. The result is that the high-end hyphenation, kerning, and ligature support has had to be sacrificed and the on-screen result is reminiscent of the output of an HTML rendering engine or a very basic word processor.

This paper investigates an alternative approach to generating the display for an eBook reader. Here, the text is pre-rendered (using a high-quality typesetting algorithm) in several column widths, prior to display, when the document is created. At view time, the most appropriate column width is selected for display, the system balancing between excessive white space and multiple columns. Section 2 examines the problems posed by current eBook readers in further detail, while section 3 presents our initial prototype solution to some of these problems.

2. PROBLEMS WITH CURRENT EBOOK READERS

Three formats currently dominate the eBook market: EPUB and Mobipocket, which allow the document to be formatted to fit the device, and PDF, which does not. (Amazon's proprietary Kindle format is derived from Mobipocket; PDF and EPUB are open standards.) Both the EPUB and Mobipocket formats are largely based on XHTML. Whilst the use of XML-derived formats allow the semantic structure of documents to be very well defined, in general their presentation can only be specified in a very loose manner. The user is often presented with a choice of typefaces and point sizes, allowing the reader software to render the document in essentially any arbitrary way it



chooses.

CHAPTER 1
Loomings

Call me Ishmael. Some years ago—never mind how long precisely— having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself

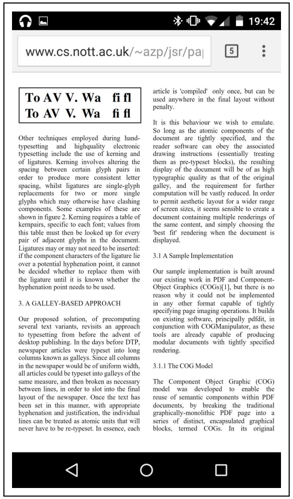
Conversely, PDF is entirely presentation-oriented, stemming from its origin of being compiled PostScript. PDF, therefore, will often include no information on the semantic structure of the document, and will consist simply of drawing operators which describe the document pages. There is no compulsion for these drawing operators to render the page in an order thatmight be considered sensible: for example, if a PDF generator program decided to render every character on a page in alphabetical order, or radially outwards from the centre, the resulting file would still be semantically valid, and the result might well be unnoticeable to the end user. This lack of imposed semantic structure can make it difficult to infer the best way to 'unpick' PDF files to allow their content to be reflowed into a new layout.

Since an XHTML-derived format has no fixed presentation associated with it, this must be calculated each time the document is displayed, in a similar manner to the way an interpreted programming language needs to be interpreted each time it runs. For an eBook reader to maximise its battery life (the human reader will be annoyed if the device dies just before the climax of a novel!), the 'interpretation' needs to be as simple as possible - i.e. the algorithm used must not be too complex, since the more CPU cycles spent executing it, the less time the CPU can spend idle, and hence the greater the drain on the battery. Furthermore, the longer that is spent formatting the output, the longer the delay between page turns on the device, and with the speed of CPUs used in these devices (< 500 MHz) it does not take too large an increase in computation for the page turn to become noticeable.

2.1 Hyphenation and Line-Breaking

eBook readers typically use a 'greedy' algorithm to lay out their text - that is, they place as many words as will fit onto the current line without exceeding it, then start a new line and continue. Although this algorithm is optimal in that it will always fit text onto the fewest possible lines, it often causes consecutive lines to have wildly varying lengths, accentuating either the 'ragged-right' effect of the text, or, in the case of justified text, the inter-word spacing. In general, eBook readers will only hyphenate in extreme cases - indeed the Kindle 3 seems not to do so at all. Knuth and Plass[7] developed a more advanced line-breaking algorithm (now used by TEX) which attempts to minimise large discrepancies between consecutive lines by considering each paragraph as a whole. TEX also uses the hyphenation algorithm designed by Liang[8], which has been ported to many other applications.

2.2 Other Typographical Techniques



Other techniques employed during hand-typesetting and highquality electronic typesetting include the use of kerning and of ligatures. Kerning involves altering the spacing between certain glyph pairs in order to produce more consistent letter spacing, whilst ligatures are single-glyph replacements for two or more single glyphs which may otherwise have clashing components. Some examples of these are shown in figure 2. Kerning requires a table of kernpairs, specific to each font; values from this table must then be looked up for every pair of adjacent glyphs in the document. Ligatures may or may not need to be inserted: if the component characters of the ligature lie over a potential hyphenation point, it cannot be decided whether to replace them with the ligature until it is known whether the hyphenation point needs to be used.

3. A GALLEY-BASED APPROACH

Our proposed solution, of precomputing several text variants, revisits an approach to typesetting from before the advent of desktop publishing. In the days before DTP, newspaper articles were typeset into long columns known as galleys. Since all columns in the newspaper would be of uniform width, all articles could be typeset into galleys of the same measure, and then broken as necessary between lines, in order to slot into the final layout of the newspaper. Once the text has been set in this manner, with appropriate hyphenation and justification, the individual lines can be treated as atomic units that will never have to be re-typeset. In essence, each article is 'compiled' only once, but can be used anywhere in the final layout without penalty.

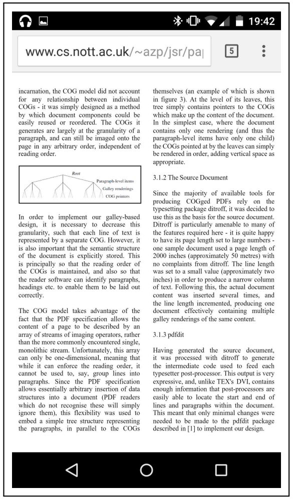
It is this behaviour we wish to emulate. So long as the atomic components of the document are tightly specified, and the reader software can obey the associated drawing instructions (essentially treating them as pre-typeset blocks), the resulting display of the document will be of as high typographic quality as that of the original galley, and the requirement for further computation will be vastly reduced. In order to permit aesthetic layout for a wider range of screen sizes, it seems sensible to create a document containing multiple renderings of the same content, and simply choosing the 'best fit' rendering when the document is displayed.

3.1 A Sample Implementation

Our sample implementation is built around our existing work in PDF and Component-Object Graphics (COGs)[1], but there is no reason why it could not be implemented in any other format capable of tightly specifying page imaging operations. It builds on existing software, principally pdfdit, in conjunction with COGManipulator, as these tools are already capable of producing modular documents with tightly specified rendering.

3.1.1 The COG Model

The Component Object Graphic (COG) model was developed to enable the reuse of semantic components within PDF documents, by breaking the traditional graphically-monolithic PDF page into a series of distinct, encapsulated graphical blocks, termed COGs. In its original



incarnation, the COG model did not account for any relationship between individual COGs - it was simply designed as a method by which document components could be easily reused or reordered. The COGs it generates are largely at the granularity of a paragraph, and can still be imaged onto the page in any arbitrary order, independent of reading order.

In order to implement our galley-based design, it is necessary to decrease this granularity, such that each line of text is represented by a separate COG. However, it is also important that the semantic structure of the document is explicitly stored. This is principally so that the reading order of the COGs is maintained, and also so that the reader software can identify paragraphs, headings etc. to enable them to be laid out correctly.

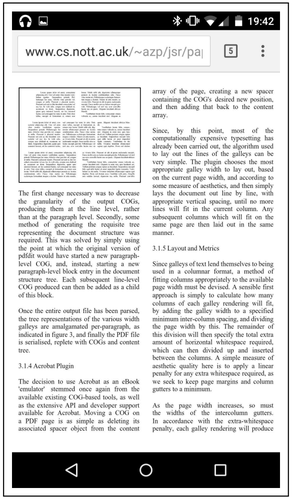
The COG model takes advantage of the fact that the PDF specification allows the content of a page to be described by an array of streams of imaging operators, rather than the more commonly encountered single, monolithic stream. Unfortunately, this array can only be one-dimensional, meaning that while it can enforce the reading order, it cannot be used to, say, group lines into paragraphs. Since the PDF specification allows essentially arbitrary insertion of data structures into a document (PDF readers which do not recognise these will simply ignore them), this flexibility was used to embed a simple tree structure representing the paragraphs, in parallel to the COGs themselves (an example of which is shown in figure 3). At the level of its leaves, this tree simply contains pointers to the COGs which make up the content of the document. In the simplest case, where the document contains only one rendering (and thus the paragraph-level items have only one child) the COGs pointed at by the leaves can simply be rendered in order, adding vertical space as appropriate.

3.1.2 The Source Document

Since the majority of available tools for producing COGged PDFs rely on the typesetting package ditroff, it was decided to use this as the basis for the source document. Ditroff is particularly amenable to many of the features required here - it is quite happy to have its page length set to large numbers - one sample document used a page length of 2000 inches (approximately 50 metres) with no complaints from ditroff. The line length was set to a small value (approximately two inches) in order to produce a narrow column of text. Following this, the actual document content was inserted several times, and the line length incremented, producing one document effectively containing multiple galley renderings of the same content.

3.1.3 pdfdit

Having generated the source document, it was processed with ditroff to generate the intermediate code used to feed each typesetter post-processor. This output is very expressive, and, unlike TEX's DVI, contains enough information that post-processors are easily able to locate the start and end of lines and paragraphs within the document. This meant that only minimal changes were needed to be made to the pdfdit package described in [1] to implement our design.



The first change necessary was to decrease the granularity of the output COGs, producing them at the line level, rather than at the paragraph level. Secondly, some method of generating the requisite tree representing the document structure was required. This was solved by simply using the point at which the original version of pdfdit would have started a new paragraph-level COG, and, instead, starting a new paragraph-level block entry in the document structure tree. Each subsequent line-level COG produced can then be added as a child of this block.

Once the entire output file has been parsed, the tree representations of the various width galleys are amalgamated per-paragraph, as indicated in figure 3, and finally the PDF file is serialised, replete with COGs and content tree.

3.1.4 Acrobat Plugin

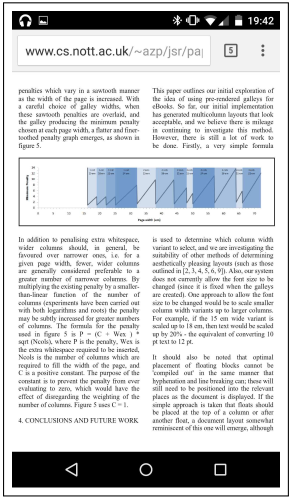
The decision to use Acrobat as an eBook 'emulator' stemmed once again from the available existing COG-based tools, as well as the extensive API and developer support available for Acrobat. Moving a COG on a PDF page is as simple as deleting its associated spacer object from the content array of the page, creating a new spacer containing the COG's desired new position, and then adding that back to the content array.

Since, by this point, most of the computationally expensive typesetting has already been carried out, the algorithm used to lay out the lines of the galleys can be very simple. The plugin chooses the most appropriate galley width to lay out, based on the current page width, and according to some measure of aesthetics, and then simply lays the document out line by line, with appropriate vertical spacing, until no more lines will fit in the current column. Any subsequent columns which will fit on the same page are then laid out in the same manner.

3.1.5 Layout and Metrics

Since galleys of text lend themselves to being used in a columnar format, a method of fitting columns appropriately to the available page width must be devised. A sensible first approach is simply to calculate how many columns of each galley rendering will fit, by adding the galley width to a specified minimum inter-column spacing, and dividing the page width by this. The remainder of this division will then specify the total extra amount of horizontal whitespace required, which can then divided up and inserted between the columns. A simple measure of aesthetic quality here is to apply a linear penalty for any extra whitespace required, as we seek to keep page margins and column gutters to a minimum.

As the page width increases, so must the widths of the intercolumn gutters. In accordance with the extra-whitespace penalty, each galley rendering will produce



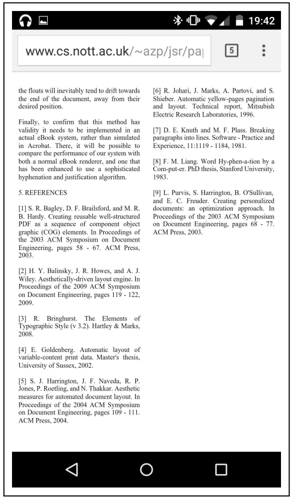
penalties which vary in a sawtooth manner as the width of the page is increased. With a careful choice of galley widths, when these sawtooth penalties are overlaid, and the galley producing the minimum penalty chosen at each page width, a flatter and finer-toothed penalty graph emerges, as shown in figure 5.

In addition to penalising extra whitespace, wider columns should, in general, be favoured over narrower ones, i.e. for a given page width, fewer, wider columns are generally considered preferable to a greater number of narrower columns. By multiplying the existing penalty by a smaller-than-linear function of the number of columns (experiments have been carried out with both logarithms and roots) the penalty may be subtly increased for greater numbers of columns. The formula for the penalty used in figure 5 is P = (C + Wex) * sqrt (Ncols), where P is the penalty, Wex is the extra whitespace required to be inserted, Ncols is the number of columns which are required to fill the width of the page, and C is a positive constant. The purpose of the constant is to prevent the penalty from ever evaluating to zero, which would have the effect of disregarding the weighting of the number of columns. Figure 5 uses C = 1.

4. CONCLUSIONS AND FUTURE WORK

This paper outlines our initial exploration of the idea of using pre-rendered galleys for eBooks. So far, our initial implementation has generated multicolumn layouts that look acceptable, and we believe there is mileage in continuing to investigate this method. However, there is still a lot of work to be done. Firstly, a very simple formula is used to determine which column width variant to select, and we are investigating the suitability of other methods of determining aesthetically pleasing layouts (such as those outlined in [2, 3, 4, 5, 6, 9]). Also, our system does not currently allow the font size to be changed (since it is fixed when the galleys are created). One approach to allow the font size to be changed would be to scale smaller column width variants up to larger columns. For example, if the 15 em wide variant is scaled up to 18 em, then text would be scaled up by 20% - the equivalent of converting 10 pt text to 12 pt.

It should also be noted that optimal placement of floating blocks cannot be 'compiled out' in the same manner that hyphenation and line breaking can; these will still need to be positioned into the relevant places as the document is displayed. If the simple approach is taken that floats should be placed at the top of a column or after another float, a document layout somewhat reminiscent of this one will emerge, although



the floats will inevitably tend to drift towards the end of the document, away from their desired position.

Finally, to confirm that this method has validity it needs to be implemented in an actual eBook system, rather than simulated in Acrobat. There, it will be possible to compare the performance of our system with both a normal eBook renderer, and one that has been enhanced to use a sophisticated hyphenation and justification algorithm.

5. REFERENCES

[1] S. R. Bagley, D. F. Brailsford, and M. R. B. Hardy. Creating reusable well-structured PDF as a sequence of component object graphic (COG) elements. In Proceedings of the 2003 ACM Symposium on Document Engineering, pages 58 - 67. ACM Press, 2003.

[2] H. Y. Balinsky, J. R. Howes, and A. J. Wiley. Aesthetically-driven layout engine. In Proceedings of the 2009 ACM Symposium on Document Engineering, pages 119 - 122, 2009.

[3] R. Bringhurst. The Elements of Typographic Style (v 3.2). Hartley & Marks, 2008.

[4] E. Goldenberg. Automatic layout of variable-content print data. Master's thesis, University of Sussex, 2002.

[5] S. J. Harrington, J. F. Naveda, R. P. Jones, P. Roetling, and N. Thakkar. Aesthetic measures for automated document layout. In Proceedings of the 2004 ACM Symposium on Document Engineering, pages 109 - 111. ACM Press, 2004.

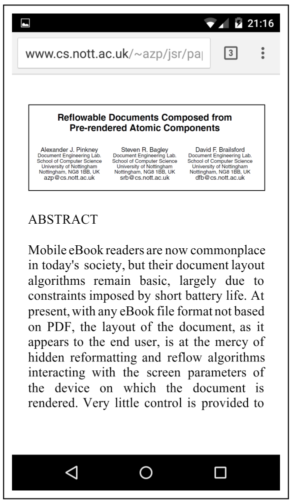
[6] R. Johari, J. Marks, A. Partovi, and S. Shieber. Automatic yellow-pages pagination and layout. Technical report, Mitsubish Electric Research Laboratories, 1996.

[7] D. E. Knuth and M. F. Plass. Breaking paragraphs into lines. Software - Practice and Experience, 11:1119 - 1184, 1981.

[8] F. M. Liang. Word Hy-phen-a-tion by a Com-put-er. PhD thesis, Stanford University, 1983.

[9] L. Purvis, S. Harrington, B. O'Sullivan, and E. C. Freuder. Creating personalized documents: an optimization approach. In Proceedings of the 2003 ACM Symposium on Document Engineering, pages 68 - 77. ACM Press, 2003.

[PBB11] laid out by the malleable document system, running in Chrome on an Android phone, in portrait and in landscape. This example uses a point size that is likely to be readable by a greater range of people. For brevity's sake, only the first three pages of each rendering are shown.


21:16
www.cs.nott.ac.uk/~azp/jsr/pa|

Reflowable Documents Composed from Pre-rendered Atomic Components

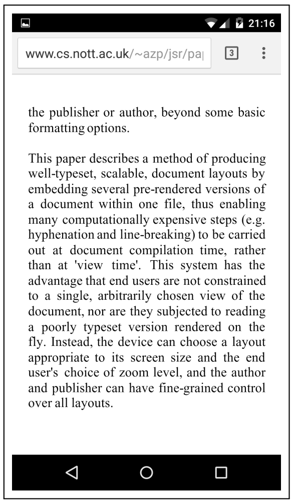
Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

David F. Brailsford
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk

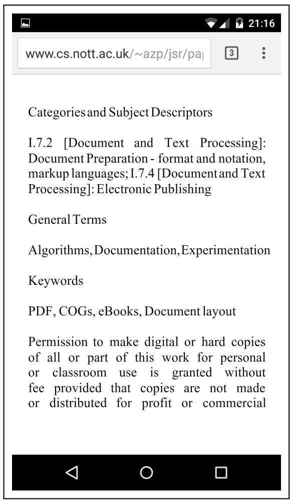
ABSTRACT

Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to


21:16
www.cs.nott.ac.uk/~azp/jsr/pa|

the publisher or author, beyond some basic formatting options.

This paper describes a method of producing well-typeset, scalable, document layouts by embedding several pre-rendered versions of a document within one file, thus enabling many computationally expensive steps (e.g. hyphenation and line-breaking) to be carried out at document compilation time, rather than at 'view time'. This system has the advantage that end users are not constrained to a single, arbitrarily chosen view of the document, nor are they subjected to reading a poorly typeset version rendered on the fly. Instead, the device can choose a layout appropriate to its screen size and the end user's choice of zoom level, and the author and publisher can have fine-grained control over all layouts.


21:16
www.cs.nott.ac.uk/~azp/jsr/pa|

Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation - format and notation, markup languages; I.7.4 [Document and Text Processing]: Electronic Publishing

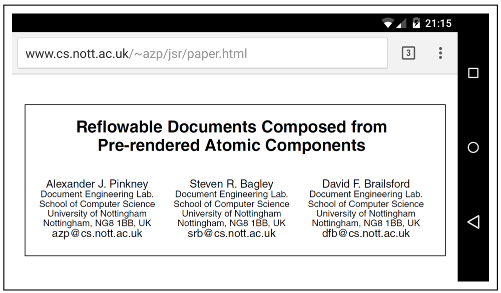
General Terms

Algorithms, Documentation, Experimentation

Keywords

PDF, COGs, eBooks, Document layout

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial

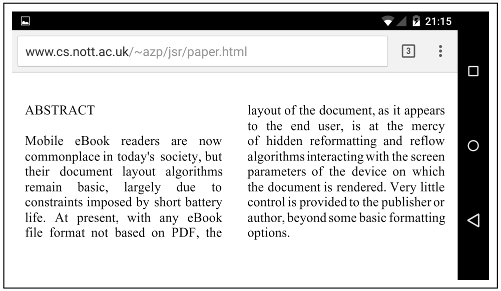

21:15
www.cs.nott.ac.uk/~azp/jsr/paper.html

Reflowable Documents Composed from Pre-rendered Atomic Components

Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

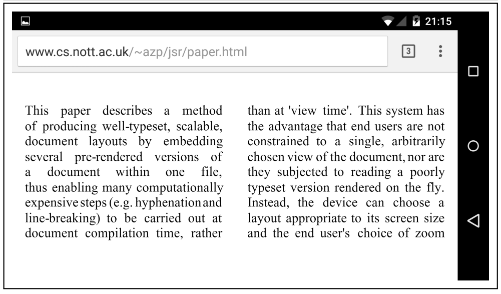
Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

David F. Brailsford
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk


21:15
www.cs.nott.ac.uk/~azp/jsr/paper.html

ABSTRACT

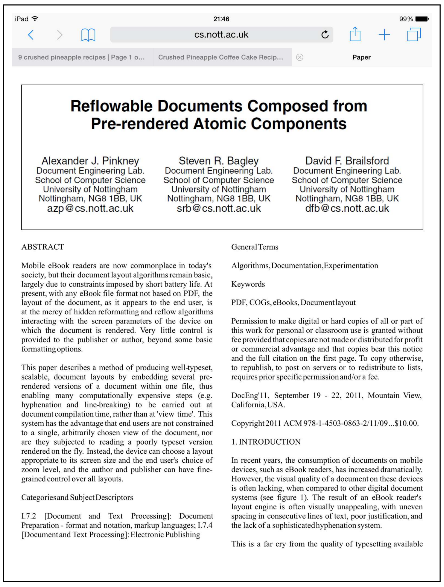
Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to the publisher or author, beyond some basic formatting options.


21:15
www.cs.nott.ac.uk/~azp/jsr/paper.html

This paper describes a method of producing well-typeset, scalable, document layouts by embedding several pre-rendered versions of a document within one file, thus enabling many computationally expensive steps (e.g. hyphenation and line-breaking) to be carried out at document compilation time, rather than at 'view time'. This system has the advantage that end users are not constrained to a single, arbitrarily chosen view of the document, nor are they subjected to reading a poorly typeset version rendered on the fly. Instead, the device can choose a layout appropriate to its screen size and the end user's choice of zoom

B.1.3 *Safari on an iPad*

[PBB11] rendered in Safari on a iPad in portrait orientation:


iPad
21:46
99%
cs.nott.ac.uk
9 crushed pineapple recipes | Page 1 o...
Crushed Pineapple Coffee Cake Recip...
Paper

Reflowable Documents Composed from Pre-rendered Atomic Components

Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

David F. Brailsford
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk

ABSTRACT

Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to the publisher or author, beyond some basic formatting options.

This paper describes a method of producing well-typeset, scalable, document layouts by embedding several pre-rendered versions of a document within one file, thus enabling many computationally expensive steps (e.g. hyphenation and line-breaking) to be carried out at document compilation time, rather than at 'view time'. This system has the advantage that end users are not constrained to a single, arbitrarily chosen view of the document, nor are they subjected to reading a poorly typeset version rendered on the fly. Instead, the device can choose a layout appropriate to its screen size and the end user's choice of zoom level, and the author and publisher can have fine-grained control over all layouts.

Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation - format and notation, markup languages; I.7.4 [Document and Text Processing]: Electronic Publishing

General Terms

Algorithms, Documentation, Experimentation

Keywords

PDF, COGs, eBooks, Document layout

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

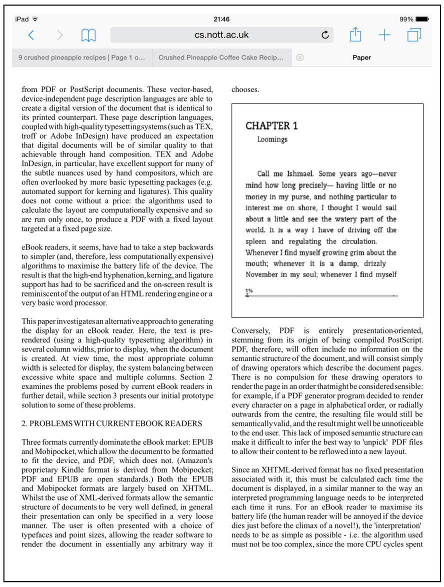
DocEng'11, September 19 - 22, 2011, Mountain View, California, USA.

Copyright 2011 ACM 978-1-4503-0863-2/11/09...$10.00.

1. INTRODUCTION

In recent years, the consumption of documents on mobile devices, such as eBook readers, has increased dramatically. However, the visual quality of a document on these devices is often lacking, when compared to other digital document systems (see figure 1). The result of an eBook reader's layout engine is often visually unappealing, with uneven spacing in consecutive lines of text, poor justification, and the lack of a sophisticated hyphenation system.

This is a far cry from the quality of typesetting available


iPad
21:46
99%
cs.nott.ac.uk
9 crushed pineapple recipes | Page 1 o...
Crushed Pineapple Coffee Cake Recip...
Paper

from PDF or PostScript documents. These vector-based, device-independent page description languages are able to create a digital version of the document that is identical to its printed counterpart. These page description languages, coupled with high-quality typesetting systems (such as TEX, troff or Adobe InDesign) have produced an expectation that digital documents will be of similar quality to that achievable through hand composition. TEX and Adobe InDesign, in particular, have excellent support for many of the subtle nuances used by hand compositors, which are often overlooked by more basic typesetting packages (e.g. automated support for kerning and ligatures). This quality does not come without a price: the algorithms used to calculate the layout are computationally expensive and so are run only once, to produce a PDF with a fixed layout targeted at a fixed page size.

eBook readers, it seems, have had to take a step backwards to simpler (and, therefore, less computationally expensive) algorithms to maximise the battery life of the device. The result is that the high-end hyphenation, kerning, and ligature support has had to be sacrificed and the on-screen result is reminiscent of the output of an HTML rendering engine or a very basic word processor.

This paper investigates an alternative approach to generating the display for an eBook reader. Here, the text is pre-rendered (using a high-quality typesetting algorithm) in several column widths, prior to display, when the document is created. At view time, the most appropriate column width is selected for display, the system balancing between excessive white space and multiple columns. Section 2 examines the problems posed by current eBook readers in further detail, while section 3 presents our initial prototype solution to some of these problems.

2. PROBLEMS WITH CURRENT EBOOK READERS

Three formats currently dominate the eBook market: EPUB and Mobipocket, which allow the document to be formatted to fit the device, and PDF, which does not. (Amazon's proprietary Kindle format is derived from Mobipocket; PDF and EPUB are open standards.) Both the EPUB and Mobipocket formats are largely based on XHTML. Whilst the use of XML-derived formats allow the semantic structure of documents to be very well defined, in general their presentation can only be specified in a very loose manner. The user is often presented with a choice of typefaces and point sizes, allowing the reader software to render the document in essentially any arbitrary way it chooses.

CHAPTER 1

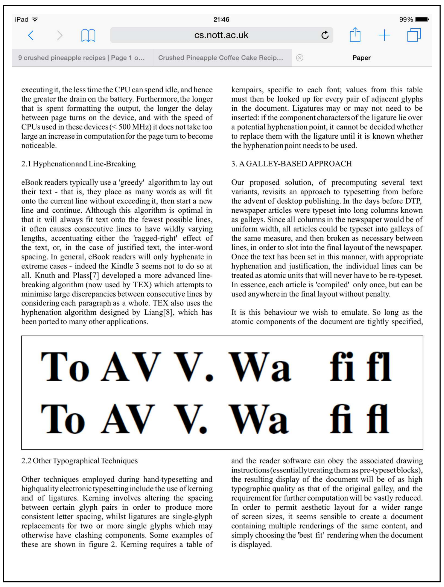
Loomings

Call me Ishmael. Some years ago—never mind how long precisely— having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself

1%

Conversely, PDF is entirely presentation-oriented, stemming from its origin of being compiled PostScript. PDF, therefore, will often include no information on the semantic structure of the document, and will consist simply of drawing operators which describe the document pages. There is no compulsion for these drawing operators to render the page in an order that might be considered sensible: for example, if a PDF generator program decided to render every character on a page in alphabetical order, or radially outwards from the centre, the resulting file would still be semantically valid, and the result might well be unnoticeable to the end user. This lack of imposed semantic structure can make it difficult to infer the best way to 'unpick' PDF files to allow their content to be reflowed into a new layout.

Since an XHTML-derived format has no fixed presentation associated with it, this must be calculated each time the document is displayed, in a similar manner to the way an interpreted programming language needs to be interpreted each time it runs. For an eBook reader to maximise its battery life (the human reader will be annoyed if the device dies just before the climax of a novel!), the 'interpretation' needs to be as simple as possible - i.e. the algorithm used must not be too complex, since the more CPU cycles spent


iPad
21:46
99%
cs.nott.ac.uk
9 crushed pineapple recipes | Page 1 o...
Crushed Pineapple Coffee Cake Recip...
Paper
To AV V. Wa fi fl
To AV V. Wa fi fl

executing it, the less time the CPU can spend idle, and hence the greater the drain on the battery. Furthermore, the longer that is spent formatting the output, the longer the delay between page turns on the device, and with the speed of CPUs used in these devices (< 500 MHz) it does not take too large an increase in computation for the page turn to become noticeable.

2.1 Hyphenation and Line-Breaking

eBook readers typically use a 'greedy' algorithm to lay out their text - that is, they place as many words as will fit onto the current line without exceeding it, then start a new line and continue. Although this algorithm is optimal in that it will always fit text onto the fewest possible lines, it often causes consecutive lines to have wildly varying lengths, accentuating either the 'ragged-right' effect of the text, or, in the case of justified text, the inter-word spacing. In general, eBook readers will only hyphenate in extreme cases - indeed the Kindle 3 seems not to do so at all. Knuth and Plass[7] developed a more advanced line-breaking algorithm (now used by TEX) which attempts to minimise large discrepancies between consecutive lines by considering each paragraph as a whole. TEX also uses the hyphenation algorithm designed by Liang[8], which has been ported to many other applications.

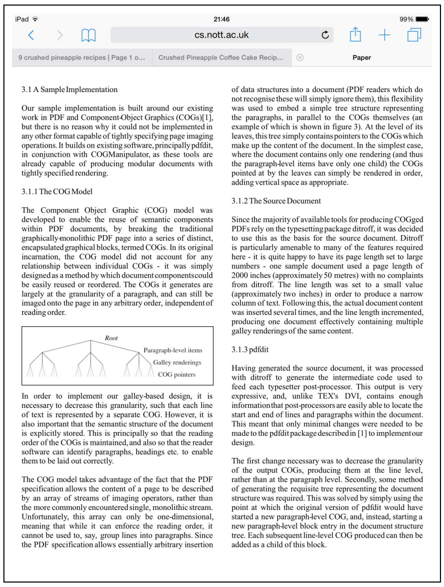
2.2 Other Typographical Techniques

Other techniques employed during hand-typesetting and highquality electronic typesetting include the use of kerning and of ligatures. Kerning involves altering the spacing between certain glyph pairs in order to produce more consistent letter spacing, whilst ligatures are single-glyph replacements for two or more single glyphs which may otherwise have clashing components. Some examples of these are shown in figure 2. Kerning requires a table of kernpairs, specific to each font; values from this table must then be looked up for every pair of adjacent glyphs in the document. Ligatures may or may not need to be inserted: if the component characters of the ligature lie over a potential hyphenation point, it cannot be decided whether to replace them with the ligature until it is known whether the hyphenation point needs to be used.

3. A GALLEY-BASED APPROACH

Our proposed solution, of precomputing several text variants, revisits an approach to typesetting from before the advent of desktop publishing. In the days before DTP, newspaper articles were typeset into long columns known as galleys. Since all columns in the newspaper would be of uniform width, all articles could be typeset into galleys of the same measure, and then broken as necessary between lines, in order to slot into the final layout of the newspaper. Once the text has been set in this manner, with appropriate hyphenation and justification, the individual lines can be treated as atomic units that will never have to be re-typeset. In essence, each article is 'compiled' only once, but can be used anywhere in the final layout without penalty.

It is this behaviour we wish to emulate. So long as the atomic components of the document are tightly specified, and the reader software can obey the associated drawing instructions (essentially treating them as pre-typeset blocks), the resulting display of the document will be of as high typographic quality as that of the original galley, and the requirement for further computation will be vastly reduced. In order to permit aesthetic layout for a wider range of screen sizes, it seems sensible to create a document containing multiple renderings of the same content, and simply choosing the 'best fit' rendering when the document is displayed.



3.1 A Sample Implementation

Our sample implementation is built around our existing work in PDF and Component-Object Graphics (COGs)[1], but there is no reason why it could not be implemented in any other format capable of tightly specifying page imaging operations. It builds on existing software, principally pdfdit, in conjunction with COGManipulator, as these tools are already capable of producing modular documents with tightly specified rendering.

3.1.1 The COG Model

The Component Object Graphic (COG) model was developed to enable the reuse of semantic components within PDF documents, by breaking the traditional graphically-monolithic PDF page into a series of distinct, encapsulated graphical blocks, termed COGs. In its original incarnation, the COG model did not account for any relationship between individual COGs - it was simply designed as a method by which document components could be easily reused or reordered. The COGs it generates are largely at the granularity of a paragraph, and can still be imaged onto the page in any arbitrary order, independent of reading order.

In order to implement our galley-based design, it is necessary to decrease this granularity, such that each line of text is represented by a separate COG. However, it is also important that the semantic structure of the document is explicitly stored. This is principally so that the reading order of the COGs is maintained, and also so that the reader software can identify paragraphs, headings etc. to enable them to be laid out correctly.

The COG model takes advantage of the fact that the PDF specification allows the content of a page to be described by an array of streams of imaging operators, rather than the more commonly encountered single, monolithic stream. Unfortunately, this array can only be one-dimensional, meaning that while it can enforce the reading order, it cannot be used to, say, group lines into paragraphs. Since the PDF specification allows essentially arbitrary insertion of data structures into a document (PDF readers which do not recognise these will simply ignore them), this flexibility was used to embed a simple tree structure representing the paragraphs, in parallel to the COGs themselves (an example of which is shown in figure 3). At the level of its leaves, this tree simply contains pointers to the COGs which make up the content of the document. In the simplest case, where the document contains only one rendering (and thus the paragraph-level items have only one child) the COGs pointed at by the leaves can simply be rendered in order, adding vertical space as appropriate.

3.1.2 The Source Document

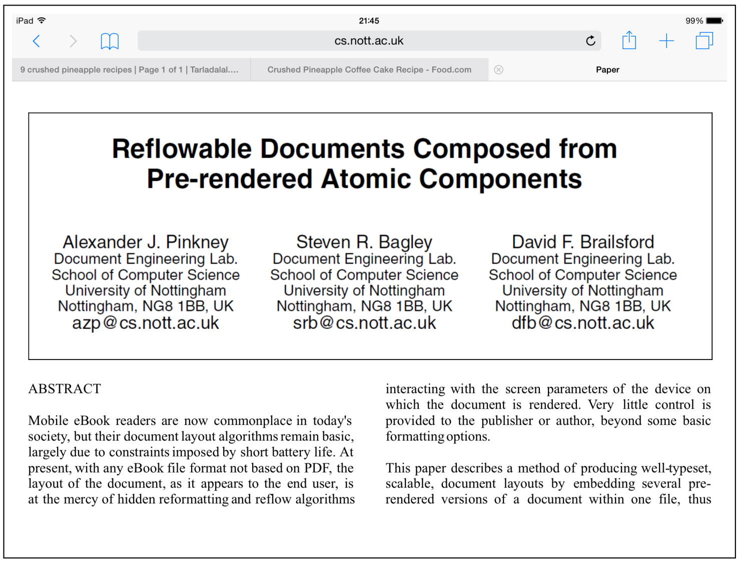
Since the majority of available tools for producing COGged PDFs rely on the typesetting package ditroff, it was decided to use this as the basis for the source document. Ditroff is particularly amenable to many of the features required here - it is quite happy to have its page length set to large numbers - one sample document used a page length of 2000 inches (approximately 50 metres) with no complaints from ditroff. The line length was set to a small value (approximately two inches) in order to produce a narrow column of text. Following this, the actual document content was inserted several times, and the line length incremented, producing one document effectively containing multiple galley renderings of the same content.

3.1.3 pdfdit

Having generated the source document, it was processed with ditroff to generate the intermediate code used to feed each typesetter post-processor. This output is very expressive, and, unlike TEX's DVI, contains enough information that post-processors are easily able to locate the start and end of lines and paragraphs within the document. This meant that only minimal changes were needed to be made to the pdfdit package described in [1] to implement our design.

The first change necessary was to decrease the granularity of the output COGs, producing them at the line level, rather than at the paragraph level. Secondly, some method of generating the requisite tree representing the document structure was required. This was solved by simply using the point at which the original version of pdfdit would have started a new paragraph-level COG, and, instead, starting a new paragraph-level block entry in the document structure tree. Each subsequent line-level COG produced can then be added as a child of this block.

[PBB11] rendered in Safari on a iPad in landscape orientation:


iPad
21:45
99%
cs.nott.ac.uk
9 crushed pineapple recipes | Page 1 of 1 | Tarladalal....
Crushed Pineapple Coffee Cake Recipe - Food.com
Paper

Reflowable Documents Composed from Pre-rendered Atomic Components

Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

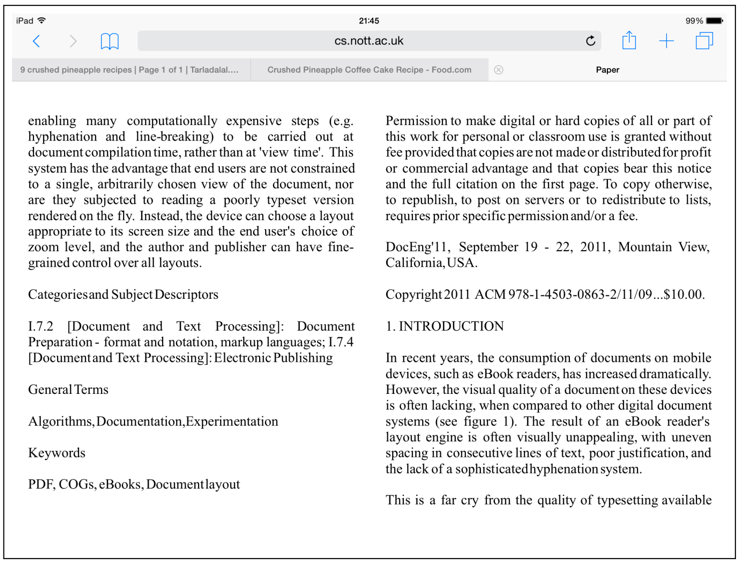
Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

David F. Brailsford
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk

ABSTRACT

Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to the publisher or author, beyond some basic formatting options.

This paper describes a method of producing well-typeset, scalable, document layouts by embedding several pre-rendered versions of a document within one file, thus


iPad
21:45
99%
cs.nott.ac.uk
9 crushed pineapple recipes | Page 1 of 1 | Tarladalal....
Crushed Pineapple Coffee Cake Recipe - Food.com
Paper

enabling many computationally expensive steps (e.g. hyphenation and line-breaking) to be carried out at document compilation time, rather than at 'view time'. This system has the advantage that end users are not constrained to a single, arbitrarily chosen view of the document, nor are they subjected to reading a poorly typeset version rendered on the fly. Instead, the device can choose a layout appropriate to its screen size and the end user's choice of zoom level, and the author and publisher can have fine-grained control over all layouts.

Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation - format and notation, markup languages; I.7.4 [Document and Text Processing]: Electronic Publishing

General Terms

Algorithms, Documentation, Experimentation

Keywords

PDF, COGs, eBooks, Document layout

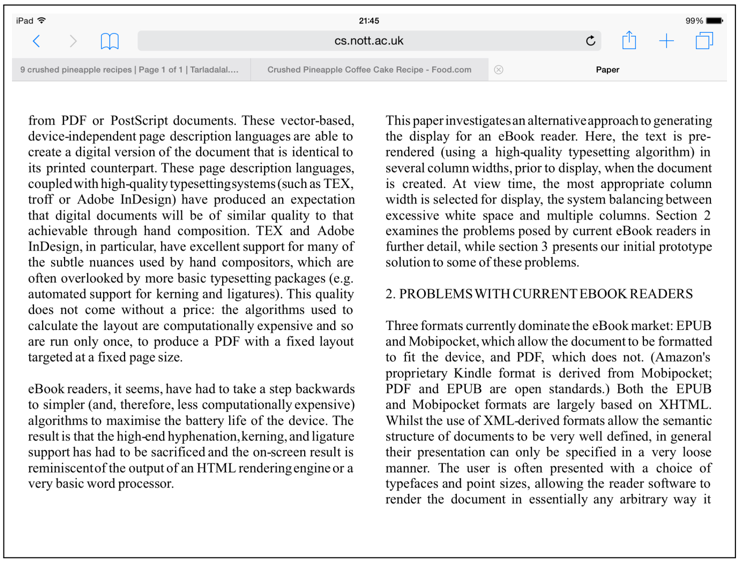
DocEng'11, September 19 - 22, 2011, Mountain View, California, USA.

Copyright 2011 ACM 978-1-4503-0863-2/11/09...$10.00.

1. INTRODUCTION

In recent years, the consumption of documents on mobile devices, such as eBook readers, has increased dramatically. However, the visual quality of a document on these devices is often lacking, when compared to other digital document systems (see figure 1). The result of an eBook reader's layout engine is often visually unappealing, with uneven spacing in consecutive lines of text, poor justification, and the lack of a sophisticated hyphenation system.

This is a far cry from the quality of typesetting available


iPad
21:45
99%
cs.nott.ac.uk
9 crushed pineapple recipes | Page 1 of 1 | Tarladalal....
Crushed Pineapple Coffee Cake Recipe - Food.com
Paper

from PDF or PostScript documents. These vector-based, device-independent page description languages are able to create a digital version of the document that is identical to its printed counterpart. These page description languages, coupled with high-quality typesetting systems (such as TEX, troff or Adobe InDesign) have produced an expectation that digital documents will be of similar quality to that achievable through hand composition. TEX and Adobe InDesign, in particular, have excellent support for many of the subtle nuances used by hand compositors, which are often overlooked by more basic typesetting packages (e.g. automated support for kerning and ligatures). This quality does not come without a price: the algorithms used to calculate the layout are computationally expensive and so are run only once, to produce a PDF with a fixed layout targeted at a fixed page size.

eBook readers, it seems, have had to take a step backwards to simpler (and, therefore, less computationally expensive) algorithms to maximise the battery life of the device. The result is that the high-end hyphenation, kerning, and ligature support has had to be sacrificed and the on-screen result is reminiscent of the output of an HTML rendering engine or a very basic word processor.

This paper investigates an alternative approach to generating the display for an eBook reader. Here, the text is pre-rendered (using a high-quality typesetting algorithm) in several column widths, prior to display, when the document is created. At view time, the most appropriate column width is selected for display, the system balancing between excessive white space and multiple columns. Section 2 examines the problems posed by current eBook readers in further detail, while section 3 presents our initial prototype solution to some of these problems.

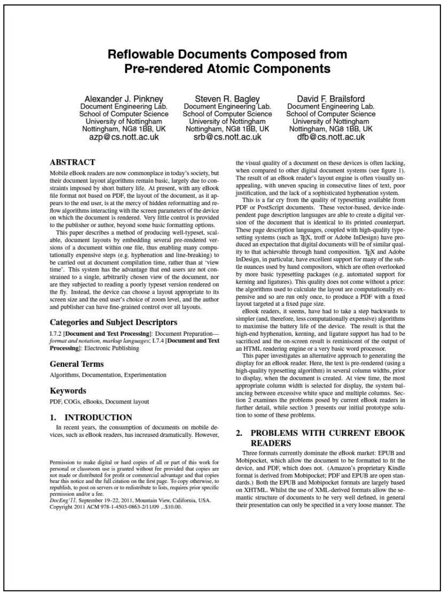
2. PROBLEMS WITH CURRENT EBOOK READERS

Three formats currently dominate the eBook market: EPUB and Mobipocket, which allow the document to be formatted to fit the device, and PDF, which does not. (Amazon's proprietary Kindle format is derived from Mobipocket; PDF and EPUB are open standards.) Both the EPUB and Mobipocket formats are largely based on XHTML. Whilst the use of XML-derived formats allow the semantic structure of documents to be very well defined, in general their presentation can only be specified in a very loose manner. The user is often presented with a choice of typefaces and point sizes, allowing the reader software to render the document in essentially any arbitrary way it

B.2 OTHER SYSTEMS

B.2.1 LaTeX

[PBB11] as rendered by LaTeX. The layout is very similar to that produced by the malleable document system (see B.1.1) though LaTeX favours placing floats at the top of columns, rather than directly at their logical position.



Reflowable Documents Composed from Pre-rendered Atomic Components

Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

David F. Brailsford
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk

ABSTRACT

Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to the publisher or author, beyond some basic formatting options.

This paper describes a method of producing well-typeset, scalable, document layouts by embedding several pre-rendered versions of a document within one file, thus enabling many computationally expensive steps (e.g. hyphenation and line-breaking) to be carried out at document compilation time, rather than at 'view time'. This system has the advantage that end users are not constrained to a single, arbitrarily chosen view of the document, nor are they subjected to reading a poorly typeset version rendered on the fly. Instead, the device can choose a layout appropriate to its screen size and the end user's choice of zoom level, and the author and publisher can have fine-grained control over all layouts.

Categories and Subject Descriptors

I.7.2 [**Document and Text Processing**]: Document Preparation—*format and notation, markup languages*; I.7.4 [**Document and Text Processing**]: Electronic Publishing

General Terms

Algorithms, Documentation, Experimentation

Keywords

PDF, COGs, eBooks, Document layout

1. INTRODUCTION

In recent years, the consumption of documents on mobile devices, such as eBook readers, has increased dramatically. However, the visual quality of a document on these devices is often lacking, when compared to other digital document systems (see figure 1). The result of an eBook reader's layout engine is often visually unappealing, with uneven spacing in consecutive lines of text, poor justification, and the lack of a sophisticated hyphenation system.

This is a far cry from the quality of typesetting available from PDF or PostScript documents. These vector-based, device-independent page description languages are able to create a digital version of the document that is identical to its printed counterpart. These page description languages, coupled with high-quality typesetting systems (such as TeX, troff or Adobe InDesign) have produced an expectation that digital documents will be of similar quality to that achievable through hand composition. TeX and Adobe InDesign, in particular, have excellent support for many of the subtle nuances used by hand compositors, which are often overlooked by more basic typesetting packages (e.g. automated support for kerning and ligatures). This quality does not come without a price: the algorithms used to calculate the layout are computationally expensive and so are run only once, to produce a PDF with a fixed layout targeted at a fixed page size.

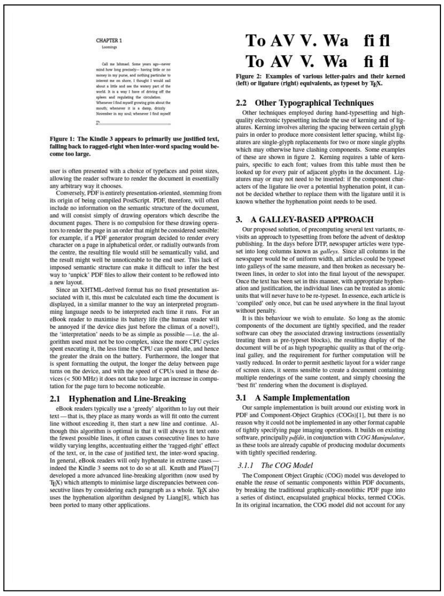
eBook readers, it seems, have had to take a step backwards to simpler (and, therefore, less computationally expensive) algorithms to maximise the battery life of the device. The result is that the high-end hyphenation, kerning, and ligature support has had to be sacrificed and the on-screen result is reminiscent of the output of an HTML rendering engine or a very basic word processor.

This paper investigates an alternative approach to generating the display for an eBook reader. Here, the text is pre-rendered (using a high-quality typesetting algorithm) in several column widths, prior to display, when the document is created. At view time, the most appropriate column width is selected for display, the system balancing between excessive white space and multiple columns. Section 2 examines the problems posed by current eBook readers in further detail, while section 3 presents our initial prototype solution to some of these problems.

2. PROBLEMS WITH CURRENT EBOOK READERS

Three formats currently dominate the eBook market: EPUB and Mobipocket, which allow the document to be formatted to fit the device, and PDF, which does not. (Amazon's proprietary Kindle format is derived from Mobipocket; PDF and EPUB are open standards.) Both the EPUB and Mobipocket formats are largely based on XHTML. Whilst the use of XML-derived formats allow the semantic structure of documents to be very well defined, in general their presentation can only be specified in a very loose manner. The

DocEng'11, September 19–22, 2011, Mountain View, California, USA.
Copyright 2011 ACM 978-1-4503-0863-2/11/09 ...$10.00.



CHAPTER 1
Loomings

Call me Ishmael. Some years ago—never mind how long precisely— having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself

Figure 1: The Kindle 3 appears to primarily use justified text, falling back to ragged-right when inter-word spacing would become too large.

user is often presented with a choice of typefaces and point sizes, allowing the reader software to render the document in essentially any arbitrary way it chooses.

Conversely, PDF is entirely presentation-oriented, stemming from its origin of being compiled PostScript. PDF, therefore, will often include no information on the semantic structure of the document, and will consist simply of drawing operators which describe the document pages. There is no compulsion for these drawing operators to render the page in an order that might be considered sensible: for example, if a PDF generator program decided to render every character on a page in alphabetical order, or radially outwards from the centre, the resulting file would still be semantically valid, and the result might well be unnoticeable to the end user. This lack of imposed semantic structure can make it difficult to infer the best way to 'unpick' PDF files to allow their content to be reflowed into a new layout.

Since an XHTML-derived format has no fixed presentation associated with it, this must be calculated each time the document is displayed, in a similar manner to the way an interpreted programming language needs to be interpreted each time it runs. For an eBook reader to maximise its battery life (the human reader will be annoyed if the device dies just before the climax of a novel!), the 'interpretation' needs to be as simple as possible — i.e. the algorithm used must not be too complex, since the more CPU cycles spent executing it, the less time the CPU can spend idle, and hence the greater the drain on the battery. Furthermore, the longer that is spent formatting the output, the longer the delay between page turns on the device, and with the speed of CPUs used in these devices (< 500 MHz) it does not take too large an increase in computation for the page turn to become noticeable.

2.1 Hyphenation and Line-Breaking

eBook readers typically use a 'greedy' algorithm to lay out their text — that is, they place as many words as will fit onto the current line without exceeding it, then start a new line and continue. Although this algorithm is optimal in that it will always fit text onto the fewest possible lines, it often causes consecutive lines to have wildly varying lengths, accentuating either the 'ragged-right' effect of the text, or, in the case of justified text, the inter-word spacing. In general, eBook readers will only hyphenate in extreme cases — indeed the Kindle 3 seems not to do so at all. Knuth and Plass[7] developed a more advanced line-breaking algorithm (now used by TeX) which attempts to minimise large discrepancies between consecutive lines by considering each paragraph as a whole. TeX also uses the hyphenation algorithm designed by Liang[8], which has been ported to many other applications.

To AV V. Wa fi fl
To AV V. Wa fi fl

Figure 2: Examples of various letter-pairs and their kerned (left) or ligature (right) equivalents, as typeset by TeX.

2.2 Other Typographical Techniques

Other techniques employed during hand-typesetting and high-quality electronic typesetting include the use of kerning and of ligatures. Kerning involves altering the spacing between certain glyph pairs in order to produce more consistent letter spacing, whilst ligatures are single-glyph replacements for two or more single glyphs which may otherwise have clashing components. Some examples of these are shown in figure 2. Kerning requires a table of kern-pairs, specific to each font; values from this table must then be looked up for every pair of adjacent glyphs in the document. Ligatures may or may not need to be inserted: if the component characters of the ligature lie over a potential hyphenation point, it cannot be decided whether to replace them with the ligature until it is known whether the hyphenation point needs to be used.

3. A GALLEY-BASED APPROACH

Our proposed solution, of precomputing several text variants, revisits an approach to typesetting from before the advent of desktop publishing. In the days before DTP, newspaper articles were typeset into long columns known as *galleys*. Since all columns in the newspaper would be of uniform width, all articles could be typeset into galleys of the same measure, and then broken as necessary between lines, in order to slot into the final layout of the newspaper. Once the text has been set in this manner, with appropriate hyphenation and justification, the individual lines can be treated as atomic units that will never have to be re-typeset. In essence, each article is 'compiled' only once, but can be used anywhere in the final layout without penalty.

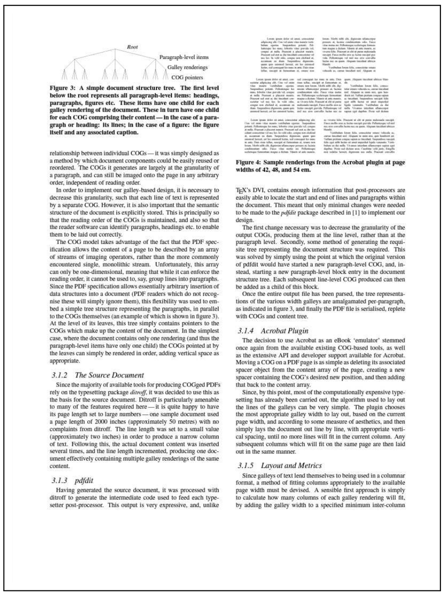
It is this behaviour we wish to emulate. So long as the atomic components of the document are tightly specified, and the reader software can obey the associated drawing instructions (essentially treating them as pre-typeset blocks), the resulting display of the document will be of as high typographic quality as that of the original galley, and the requirement for further computation will be vastly reduced. In order to permit aesthetic layout for a wider range of screen sizes, it seems sensible to create a document containing multiple renderings of the same content, and simply choosing the 'best fit' rendering when the document is displayed.

3.1 A Sample Implementation

Our sample implementation is built around our existing work in PDF and Component-Object Graphics (COGs)[1], but there is no reason why it could not be implemented in any other format capable of tightly specifying page imaging operations. It builds on existing software, principally *pdfdit*, in conjunction with *COG Manipulator*, as these tools are already capable of producing modular documents with tightly specified rendering.

3.1.1 The COG Model

The Component Object Graphic (COG) model was developed to enable the reuse of semantic components within PDF documents, by breaking the traditional graphically-monolithic PDF page into a series of distinct, encapsulated graphical blocks, termed COGs. In its original incarnation, the COG model did not account for any



Root
Paragraph-level items
Galley renderings
COG pointers

Figure 3: A simple document structure tree. The first level below the root represents all paragraph-level items: headings, paragraphs, figures etc. These items have one child for each galley rendering of the document. These in turn have one child for each COG comprising their content — in the case of a paragraph or heading: its lines; in the case of a figure: the figure itself and any associated caption.

relationship between individual COGs — it was simply designed as a method by which document components could be easily reused or reordered. The COGs it generates are largely at the granularity of a paragraph, and can still be imaged onto the page in any arbitrary order, independent of reading order.

In order to implement our galley-based design, it is necessary to decrease this granularity, such that each line of text is represented by a separate COG. However, it is also important that the semantic structure of the document is explicitly stored. This is principally so that the reading order of the COGs is maintained, and also so that the reader software can identify paragraphs, headings etc. to enable them to be laid out correctly.

The COG model takes advantage of the fact that the PDF specification allows the content of a page to be described by an array of streams of imaging operators, rather than the more commonly encountered single, monolithic stream. Unfortunately, this array can only be one-dimensional, meaning that while it can enforce the reading order, it cannot be used to, say, group lines into paragraphs. Since the PDF specification allows essentially arbitrary insertion of data structures into a document (PDF readers which do not recognise these will simply ignore them), this flexibility was used to embed a simple tree structure representing the paragraphs, in parallel to the COGs themselves (an example of which is shown in figure 3). At the level of its leaves, this tree simply contains pointers to the COGs which make up the content of the document. In the simplest case, where the document contains only one rendering (and thus the paragraph-level items have only one child) the COGs pointed at by the leaves can simply be rendered in order, adding vertical space as appropriate.

3.1.2 The Source Document

Since the majority of available tools for producing COGged PDFs rely on the typesetting package *ditroff*, it was decided to use this as the basis for the source document. Ditroff is particularly amenable to many of the features required here — it is quite happy to have its page length set to large numbers — one sample document used a page length of 2000 inches (approximately 50 metres) with no complaints from ditroff. The line length was set to a small value (approximately two inches) in order to produce a narrow column of text. Following this, the actual document content was inserted several times, and the line length incremented, producing one document effectively containing multiple galley renderings of the same content.

3.1.3 pdfdit

Having generated the source document, it was processed with ditroff to generate the intermediate code used to feed each typesetter post-processor. This output is very expressive, and, unlike TeX's DVI, contains enough information that post-processors are easily able to locate the start and end of lines and paragraphs within the document. This meant that only minimal changes were needed to be made to the *pdfdit* package described in [1] to implement our design.

Figure 4: Sample renderings from the Acrobat plugin at page widths of 42, 48, and 54 em.

The first change necessary was to decrease the granularity of the output COGs, producing them at the line level, rather than at the paragraph level. Secondly, some method of generating the requisite tree representing the document structure was required. This was solved by simply using the point at which the original version of pdfdit would have started a new paragraph-level COG, and, instead, starting a new paragraph-level block entry in the document structure tree. Each subsequent line-level COG produced can then be added as a child of this block.

Once the entire output file has been parsed, the tree representations of the various width galleys are amalgamated per-paragraph, as indicated in figure 3, and finally the PDF file is serialised, replete with COGs and content tree.

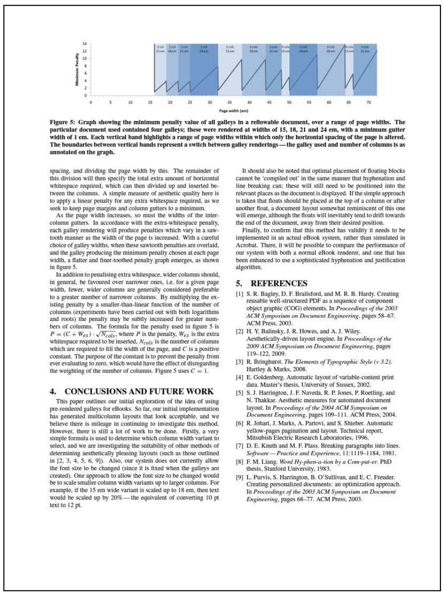
3.1.4 Acrobat Plugin

The decision to use Acrobat as an eBook 'emulator' stemmed once again from the available existing COG-based tools, as well as the extensive API and developer support available for Acrobat. Moving a COG on a PDF page is as simple as deleting its associated spacer object from the content array of the page, creating a new spacer containing the COG's desired new position, and then adding that back to the content array.

Since, by this point, most of the computationally expensive typesetting has already been carried out, the algorithm used to lay out the lines of the galleys can be very simple. The plugin chooses the most appropriate galley width to lay out, based on the current page width, and according to some measure of aesthetics, and then simply lays the document out line by line, with appropriate vertical spacing, until no more lines will fit in the current column. Any subsequent columns which will fit on the same page are then laid out in the same manner.

3.1.5 Layout and Metrics

Since galleys of text lend themselves to being used in a columnar format, a method of fitting columns appropriately to the available page width must be devised. A sensible first approach is simply to calculate how many columns of each galley rendering will fit, by adding the galley width to a specified minimum inter-column



Minimum Penalty
Page width (em)

Figure 5: Graph showing the minimum penalty value of all galleys in a reflowable document, over a range of page widths. The particular document used contained four galleys; these were rendered at widths of 15, 18, 21 and 24 em, with a minimum gutter width of 1 em. Each vertical band highlights a range of page widths within which only the horizontal spacing of the page is altered. The boundaries between vertical bands represent a switch between galley renderings — the galley used and number of columns is as annotated on the graph.

spacing, and dividing the page width by this. The remainder of this division will then specify the total extra amount of horizontal whitespace required, which can then divided up and inserted between the columns. A simple measure of aesthetic quality here is to apply a linear penalty for any extra whitespace required, as we seek to keep page margins and column gutters to a minimum.

As the page width increases, so must the widths of the inter-column gutters. In accordance with the extra-whitespace penalty, each galley rendering will produce penalties which vary in a saw-tooth manner as the width of the page is increased. With a careful choice of galley widths, when these sawtooth penalties are overlaid, and the galley producing the minimum penalty chosen at each page width, a flatter and finer-toothed penalty graph emerges, as shown in figure 5.

In addition to penalising extra whitespace, wider columns should, in general, be favoured over narrower ones, i.e. for a given page width, fewer, wider columns are generally considered preferable to a greater number of narrower columns. By multiplying the existing penalty by a smaller-than-linear function of the number of columns (experiments have been carried out with both logarithms and roots) the penalty may be subtly increased for greater numbers of columns. The formula for the penalty used in figure 5 is $P = (C + W_{ex}) \cdot \sqrt{N_{cols}}$, where P is the penalty, W_{ex} is the extra whitespace required to be inserted, N_{cols} is the number of columns which are required to fill the width of the page, and C is a positive constant. The purpose of the constant is to prevent the penalty from ever evaluating to zero, which would have the effect of disregarding the weighting of the number of columns. Figure 5 uses $C = 1$.

4. CONCLUSIONS AND FUTURE WORK

This paper outlines our initial exploration of the idea of using pre-rendered galleys for eBooks. So far, our initial implementation has generated multicolumn layouts that look acceptable, and we believe there is mileage in continuing to investigate this method. However, there is still a lot of work to be done. Firstly, a very simple formula is used to determine which column width variant to select, and we are investigating the suitability of other methods of determining aesthetically pleasing layouts (such as those outlined in [2, 3, 4, 5, 6, 9]). Also, our system does not currently allow the font size to be changed (since it is fixed when the galleys are created). One approach to allow the font size to be changed would be to scale smaller column width variants up to larger columns. For example, if the 15 em wide variant is scaled up to 18 em, then text would be scaled up by 20% — the equivalent of converting 10 pt text to 12 pt.

It should also be noted that optimal placement of floating blocks cannot be 'compiled out' in the same manner that hyphenation and line breaking can; these will still need to be positioned into the relevant places as the document is displayed. If the simple approach is taken that floats should be placed at the top of a column or after another float, a document layout somewhat reminiscent of this one will emerge, although the floats will inevitably tend to drift towards the end of the document, away from their desired position.

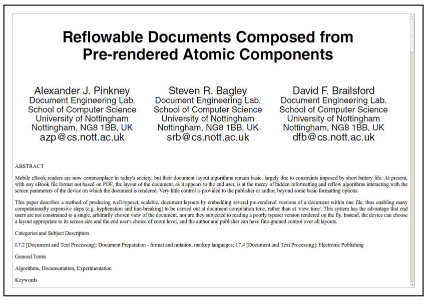
Finally, to confirm that this method has validity it needs to be implemented in an actual eBook system, rather than simulated in Acrobat. There, it will be possible to compare the performance of our system with both a normal eBook renderer, and one that has been enhanced to use a sophisticated hyphenation and justification algorithm.

5. REFERENCES

[1] S. R. Bagley, D. F. Brailsford, and M. R. B. Hardy. Creating reusable well-structured PDF as a sequence of component object graphic (COG) elements. In *Proceedings of the 2003 ACM Symposium on Document Engineering*, pages 58–67. ACM Press, 2003.
[2] H. Y. Balinsky, J. R. Howes, and A. J. Wiley. Aesthetically-driven layout engine. In *Proceedings of the 2009 ACM Symposium on Document Engineering*, pages 119–122, 2009.
[3] R. Bringhurst. *The Elements of Typographic Style (v 3.2)*. Hartley & Marks, 2008.
[4] E. Goldenberg. Automatic layout of variable-content print data. Master's thesis, University of Sussex, 2002.
[5] S. J. Harrington, J. F. Naveda, R. P. Jones, P. Roetling, and N. Thakkar. Aesthetic measures for automated document layout. In *Proceedings of the 2004 ACM Symposium on Document Engineering*, pages 109–111. ACM Press, 2004.
[6] R. Johari, J. Marks, A. Partovi, and S. Shieber. Automatic yellow-pages pagination and layout. Technical report, Mitsubish Electric Research Laboratories, 1996.
[7] D. E. Knuth and M. F. Plass. Breaking paragraphs into lines. *Software — Practice and Experience*, 11:1119–1184, 1981.
[8] F. M. Liang. *Word Hy-phen-a-tion by a Com-put-er*. PhD thesis, Stanford University, 1983.
[9] L. Purvis, S. Harrington, B. O'Sullivan, and E. C. Freuder. Creating personalized documents: an optimization approach. In *Proceedings of the 2003 ACM Symposium on Document Engineering*, pages 68–77. ACM Press, 2003.

B.2.2 HTML

HTML version of [PBB11], as rendered by Mozilla Firefox, scaled to fit multiple screen sizes. Though HTML can stretch to any screen size, it tends to produce typographically inferior results, for example lines that are too long, or line breaking that results in extremely uneven spacing between adjacent lines.



Reflowable Documents Composed from Pre-rendered Atomic Components

Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

David F. Brailsford
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk

ABSTRACT

Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to the publisher or author, beyond some basic formatting options.

This paper describes a method of producing well-typeset, scalable, document layouts by embedding several pre-rendered versions of a document within one file, thus enabling many computationally expensive steps (e.g. hyphenation and line-breaking) to be carried out at document compilation time, rather than at 'view time'. This system has the advantage that end users are not constrained to a single, arbitrarily chosen view of the document, nor are they subjected to reading a poorly typeset version rendered on the fly. Instead, the device can choose a layout appropriate to its screen size and the end user's choice of zoom level, and the author and publisher can have fine-grained control over all layouts.

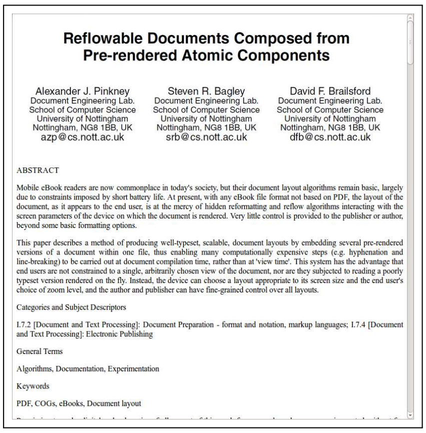
Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation - format and notation, markup languages; I.7.4 [Document and Text Processing]: Electronic Publishing

General Terms

Algorithms, Documentation, Experimentation

Keywords



Reflowable Documents Composed from Pre-rendered Atomic Components

Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

David F. Brailsford
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk

ABSTRACT

Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to the publisher or author, beyond some basic formatting options.

This paper describes a method of producing well-typeset, scalable, document layouts by embedding several pre-rendered versions of a document within one file, thus enabling many computationally expensive steps (e.g. hyphenation and line-breaking) to be carried out at document compilation time, rather than at 'view time'. This system has the advantage that end users are not constrained to a single, arbitrarily chosen view of the document, nor are they subjected to reading a poorly typeset version rendered on the fly. Instead, the device can choose a layout appropriate to its screen size and the end user's choice of zoom level, and the author and publisher can have fine-grained control over all layouts.

Categories and Subject Descriptors

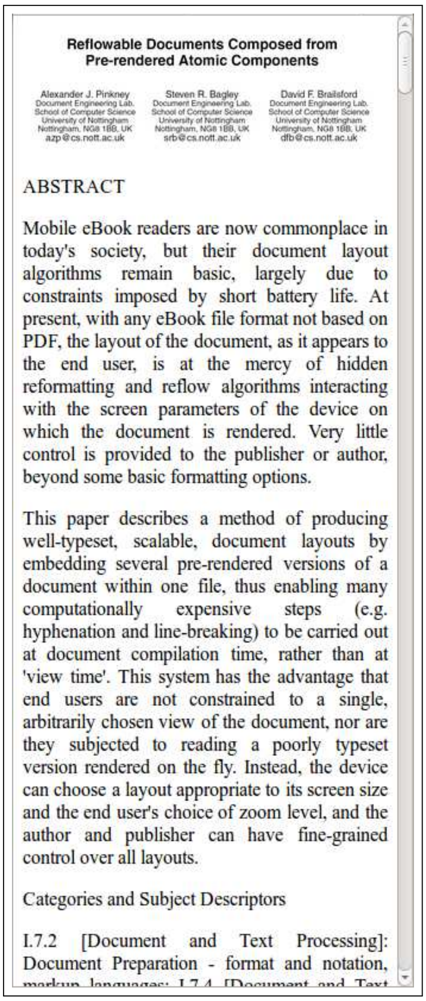
I.7.2 [Document and Text Processing]: Document Preparation - format and notation, markup languages; I.7.4 [Document and Text Processing]: Electronic Publishing

General Terms

Algorithms, Documentation, Experimentation

Keywords

PDF, COGs, eBooks, Document layout



Reflowable Documents Composed from Pre-rendered Atomic Components

Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

David F. Brailsford
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk

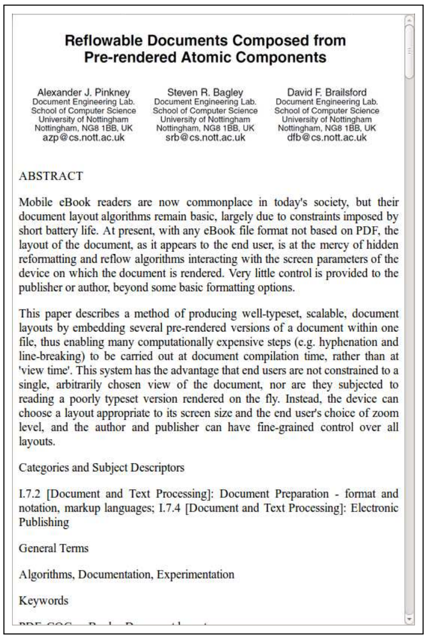
ABSTRACT

Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to the publisher or author, beyond some basic formatting options.

This paper describes a method of producing well-typeset, scalable, document layouts by embedding several pre-rendered versions of a document within one file, thus enabling many computationally expensive steps (e.g. hyphenation and line-breaking) to be carried out at document compilation time, rather than at 'view time'. This system has the advantage that end users are not constrained to a single, arbitrarily chosen view of the document, nor are they subjected to reading a poorly typeset version rendered on the fly. Instead, the device can choose a layout appropriate to its screen size and the end user's choice of zoom level, and the author and publisher can have fine-grained control over all layouts.

Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation - format and notation, markup languages; I.7.4 [Document and Text



Reflowable Documents Composed from Pre-rendered Atomic Components

Alexander J. Pinkney
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
azp@cs.nott.ac.uk

Steven R. Bagley
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
srb@cs.nott.ac.uk

David F. Brailsford
Document Engineering Lab.
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
dfb@cs.nott.ac.uk

ABSTRACT

Mobile eBook readers are now commonplace in today's society, but their document layout algorithms remain basic, largely due to constraints imposed by short battery life. At present, with any eBook file format not based on PDF, the layout of the document, as it appears to the end user, is at the mercy of hidden reformatting and reflow algorithms interacting with the screen parameters of the device on which the document is rendered. Very little control is provided to the publisher or author, beyond some basic formatting options.

This paper describes a method of producing well-typeset, scalable, document layouts by embedding several pre-rendered versions of a document within one file, thus enabling many computationally expensive steps (e.g. hyphenation and line-breaking) to be carried out at document compilation time, rather than at 'view time'. This system has the advantage that end users are not constrained to a single, arbitrarily chosen view of the document, nor are they subjected to reading a poorly typeset version rendered on the fly. Instead, the device can choose a layout appropriate to its screen size and the end user's choice of zoom level, and the author and publisher can have fine-grained control over all layouts.

Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation - format and notation, markup languages; I.7.4 [Document and Text Processing]: Electronic Publishing

General Terms

Algorithms, Documentation, Experimentation

Keywords

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". [Bri08] `classicthesis` is available for both LaTeX and LYX:

`http://code.google.com/p/classicthesis/`

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

`http://postcards.miede.de/`

Final Version as of 15th May 2015 (`azp-thesis` 1.0).