Using Text Animated Transitions to Support Navigation in Document Histories

Fanny Chevalier
Microsoft-INRIA
chevalie@lri.fr

Pierre Dragicevic
INRIA
dragice@lri.fr

Anastasia Bezerianos
École Centrale Paris
anastasia.bezerianos@ecp.fr

Jean-Daniel Fekete
INRIA
fekete@inria.fr

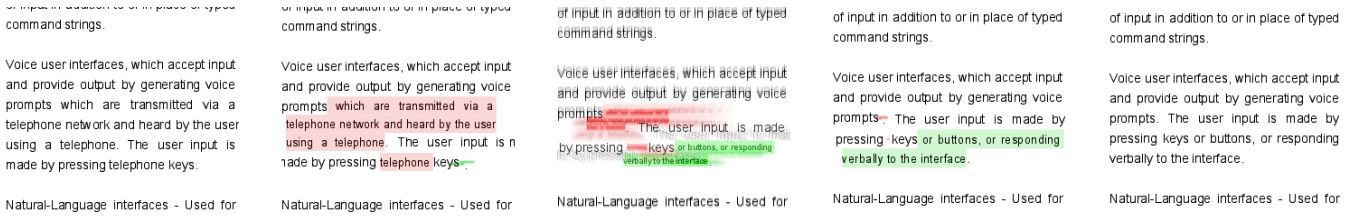

Figure 1. Detail of an animated transition between two revisions of the Wikipedia article *User interfaces*.

ABSTRACT

This article examines the benefits of using text animated transitions for navigating in the revision history of textual documents. We propose an animation technique for smoothly transitioning between different text revisions, then present the Diffamation system. Diffamation supports rapid exploration of revision histories by combining text animated transitions with simple navigation and visualization tools. We finally describe a user study showing that smooth text animation allows users to track changes in the evolution of textual documents more effectively than flipping pages.

ACM Classification Keywords

H.5.1 Multimedia Information Systems: [animations]; H.5.2 User Interfaces: [Graphical user interfaces (GUI)]

General Terms

Design, Human Factors.

INTRODUCTION

A *revision* is a snapshot of a document made persistent by its author by being saved to a file repository or database, irrespective of its quality or completeness. Keeping revisions is useful as a backup and as a means to iterate over alternative formulations without losing them. Even before electronic documents, writers kept draft paper manuscripts to monitor their progress and remember their creative process.

With the spread of collaborative writing, the need to keep revisions has increased. Apart from keeping track of their own activity, authors need to be aware of changes made by others [20]. Supporting change awareness is not only essential for writing articles, but also for programming code where changes can profoundly impact the quality of a program. Millions of people, such as programmers, researchers or Wikipedia contributors, now rely on revision control systems [22] to perform their computer-based daily activity.

While the problem of storing and retrieving revisions has been mostly solved from the technical standpoint [22], current revision management systems only support two tasks: retrieving a specified revision (by number, date, name etc.) and comparing two revisions by computing "diffs". But with the popularity of sites where anyone can edit articles, users are often interested in more complex tasks, such as understanding the edition history of an article or keeping track of specific portions of an article they are contributing to.

In this article we investigate the advantages of using smooth animation to understand changes across several revisions of a text document (Fig. 1). We describe how to smoothly animate between different text revisions and present Diffamation, an interactive tool that provides both overview and details of document changes across space and time. It allows understanding the evolution of a document, keeping track of changes to a portion of a document, or quickly seeing what happened to a document since a specific revision (e.g. what did my co-authors changed since I last revisited this article).

Animated text transitions are not meant to replace static diff visualizations, but are complementary: they are useful for rapidly browsing sequences of revisions, and once moments of interest have been identified, a static visualization can be used for detailed comparison between two specific revisions.

We first describe related work, organized according to a taxonomy of revision management tasks. We then describe our text animation technique, followed by a presentation of the Diffamation system. We finally report the results of a user study that assesses the benefits of animated text transitions.

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, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CHI 2010, April 1015, 2010, Atlanta, Georgia, USA.
Copyright 2010 ACM 978-1-60558-929-9/10/04...$10.00.

BACKGROUND

In this section we describe the basic tasks involved in text history revisitation, how changes between consecutive revisions are obtained and visualized, and finish with a brief review of previous work on animated transitions.

Tasks

Although specific domains require support for specific tasks for evolving text documents, we consider the following as the most important and ubiquitous, derived from participatory design sessions with 20 regular Wikipedia contributors:

T1: How much has changed between two revisions?

T2: What is the detailed nature of changes?

T3: What text portions have changed?

T4: Overview in time of the evolution of the entire text,

T5: Overview in time of a portion of the text,

T6: Detection of changes done and undone (reverts).

T1-T3 apply to pairs of revisions whereas T4-T6 apply to (potentially long) sequences of revisions.

Computing Changes

Although some text editors record the detailed sequence of actions performed on text documents over time, the majority of revision control systems rely on computing the differences (or *diffs*) between successive document revisions. Several algorithms have been designed for this purpose [9, 12, 15]. They mostly consist of searching for a minimal set of editing operations (insertions, deletions and sometimes moves) that transform one text file into another.

In-Text Diff Visualizations

Most revision management systems support either *side-by-side views* [1] of two revisions with visual connectors that help matching stable content, or *all-in-one views* [2] where two revisions are merged and changes are highlighted. Insertions are typically shown in green and deletions are in red or crossed out. These visualizations are useful for task T2 since they display legible text and can show details of changes in context.

Side-by-side views and all-in-one views effectively support comparison of pairs of revisions (T2) but not sequences of revisions. Several revisions can be shown side-by-side [18] but the more revisions, the less details can be shown.

Most revision control systems provide a GUI frontend for selecting revisions for comparison. But these have not been designed for efficiency: it typically takes between 10-30 sec to search and select revisions, launch the computation of diffs and display the results. It is thus difficult to track document changes over time, both for portions of text (T5) and for the entire document (T4). Detecting reverts (T6) is also slow and tedious, as it requires examining multiple diffs in sequence.

[1]For example, Kompare (http://www.caffeinated.me.uk/kompare/)

[2]For example, TortoiseMerge (http://tortoisesvn.tigris.org)

Change Overviews

In-text visualizations do not support tasks T1 and T3 very well when documents do not fit on the screen. The Eclipse editor [3] partly addresses this by showing change notification marks within the scrollbar. In Edit Wear [11] and Seesoft [6], the scrollbar shows edits aggregated over space and time using simple visualizations such as heatmaps or bar charts.

History Flow [23] shows the evolution of Wikipedia articles as horizontally-stacked bars. Each bar is an article revision, the longer the revision the higher the bar. Inserted text is shown using different colors per contributor. Putting the bars side-by-side produces the effect of a continuous editing "flow". History Flow is very effective for overview tasks T1, T4 and T6, but users have to click on vertical bars to see the actual text. This breaks the flow metaphor and makes it difficult to follow detailed changes over time (T5).

Slideshow Animations

As far as we know, the only tools that use animations to convey document edits are Wikipedia Animate and AniWiki [4]. Both show Wikipedia article revisions in sequence like a slideshow. They provide the option of highlighting the changes within the text (i.e., showing all-in-one views in sequence). A timeline widget further gives direct access to specific revisions. These tools allow to rapidly browse the document history and see changes in detail (T2). However, it is still difficult to follow specific parts of the text over time (T5) due to sudden changes in the text layout. The use of all-in-one diff views does not solve the problem. Furthermore, the absence of overviews makes it difficult to navigate in space (scrolling) or in time.

Animated Transitions

The use of animated transitions is believed to facilitate the understanding of changes and to reduce the user's cognitive load [5]. Studies have confirmed the benefits of animated transitions in zooming interfaces [19], scrolling tasks [16], panning and zooming tasks [2] and for playing back missed changes [3]. Studies consistently show that animated transitions help understanding the spatial relationship between views and help users track changes. They also indicate that animated transitions make user interfaces more pleasant to use, since users usually express a preference for them.

Although there has been work on animating text – e.g., for conveying emotions or drawing visual attention [17] – we are not aware of any work on text animated transitions. One exception is Jock et al.'s technique for showing and hiding annotations in documents [14]. Their method smoothly animates text layout to make space for supporting material such as annotations. Annotations smoothly expand or shrink between paragraphs on the user's demand. However, their method only supports insertion and deletion of entities between paragraphs – which merely involves translating paragraphs up or down and scaling text. To our knowledge, no animation technique has been proposed that can smoothly animate arbitrary changes between text documents.

[3]http://www.eclipse.org/

[4]WikiContest: http://waxy.org/2005/06/automating_wiki/

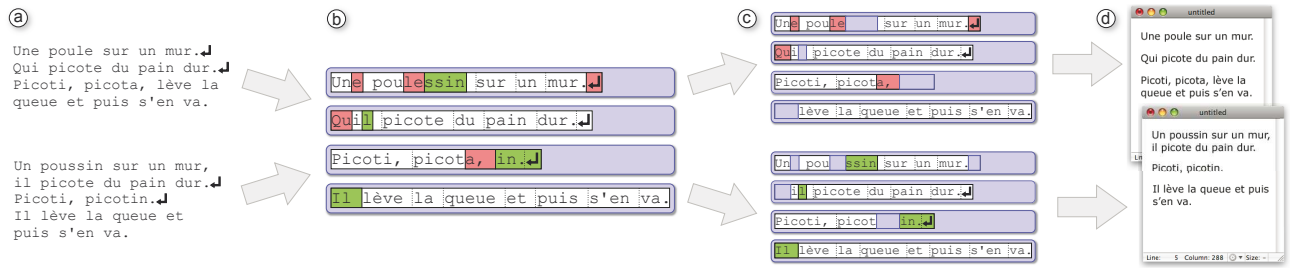

Figure 2. (a) Two texts, (b) the corresponding *superdocument* decomposed into paragraphs (blue boxes), blocks (white, red and green box) and words (separated with dashed lines), (c) the reconstructed texts, and (d) the rendered texts.

ANIMATING TEXT CHANGES

Here we address the problem of finding the best visual transition between two plain text documents. Our goal is to *make text changes understandable in the shortest possible time*, so that animations do not interfere with user interaction. From this goal we derive six design requirements.

Design requirements

R1: Non-ambiguity. The animation language must be unambiguous. For example, if a sentence flies out of the screen when it is deleted, it could be misinterpreted as having moved to a new location. In contrast, a sentence that shrinks can be more clearly interpreted as being deleted.

R2: Minimalism. The elements of the animation language must be easy to learn. For example, using different animation types for different granularities of change (character, sentence, etc.) could be confusing. In contrast, a small number of basic animations used consistently is easier to learn.

R3: Coherence. Paragraphs, sentences and words are visual landmarks that help understanding a document's organization across revisions. These visual strutures should be preserved during the animation. For example, having every word move independently would not satisfy coherence.

R4: Simple motions. Smooth animated transitions allow users to follow objects of interest. However, complex trajectories are generally more difficult to follow than simple ones, unless they serve a specific purpose [7].

R5: Smoothness. For changes to remain understandable, animations need to be smooth and sudden visual changes should be avoided. Objects should also progressively appear and disappear when added and removed [5].

R6: Adaptive speed. Animations should be as short as possible to not hinder user interaction, while being long enough to be understandable. Since the amount of changes might vary a lot across revisions, animation duration should be adaptive.

Elementary operations

Our animation language involves two elementary operations: text *insertion* and *deletion*. We did not include *move* operations for the sake of minimalism (*R2*) and coherence (*R3*). Instead, we treat them as combinations of deletions and insertions. To extract the minimal insertion and deletion operations between two revisions, we use Heckel's diff algorithm [9] at the character level. However, our animation language is not tied to any diff algorithm.

Superdocuments

We store the two document revisions within a single data structure we call *superdocument*. This structure contains a merged version obtained by sorting the diff operations (Fig. 2a,b). Superdocuments are used both for rendering the original revisions (Fig. 2c,d) and for animating the transitions.

A superdocument has a hierachical structure (see Figure 2b): *Paragraphs* are text portions separated by carriage returns and are split into *blocks* that are either stable, deleted or inserted. Blocks are built by taking the diff operations and cutting them at every carriage return. Blocks contain *words*, but note these do not always match actual words. For instance, in Figure 2 *poule* is replaced by *poussin*, which yields the "words" *pou* (stable), *le* (deleted) and *ssin* (inserted).

Rendering the original document revisions involves computing their geometry – i.e., where and how text is displayed – based on global rendering attributes such as font, page width and line spacing. Every entity (paragraph, block or word) stores its geometry as a pair of points (p, q) referring to the *top-left corner of the first word* and the *bottom-right corner of the last word*. p and q are relative to the parent's coordinates. From now on, entity's *position* will refer to p.

For rendering, the superdocument takes a parameter t and computes the geometry of the older text revision if $t = 0$ or of the newer text revision if $t = 1$. This involves discarding all insert or delete blocks (Fig. 2c) and performing a standard flow layout. In the next sections we focus on how geometry is interpolated for values of $t \in]0, 1[$. Since geometry is a function of t we will use the notation (p_t, q_t).

Geometry Interpolation

The geometries for $t = 0$ and $t = 1$ are already known. They can be quite dissimilar since even small edits may cause words to jump one line. To address this we introduce a *modulo* interpolation scheme where words progressively leave offscreen to reappar on a new line (see Fig.4c). This method relies on the notion of *equivalence of positions*, defined next.

Equivalence of Positions

Let w be the page width and h the line spacing on a document. We define a *relation of equivalence* R for positions in the document as follows: $p(x, y)\ R\ p'(x', y') \Leftrightarrow \exists\ k \in \mathbb{Z}\ /\ (x, y) = (x' + kw, y' - kh)$. The *class of equivalence* of p will be noted $\overline{p}^R$. R can be similarly defined on rectangles.

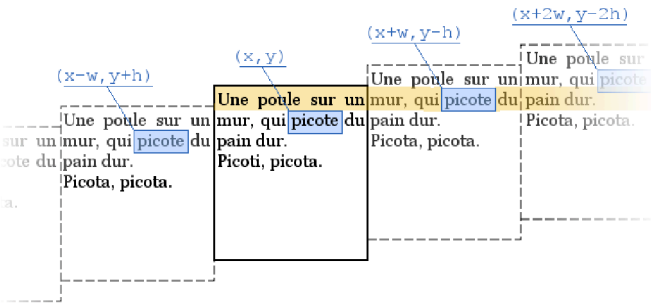

Figure 3. Illustration of the relation of equivalence R. The class of equivalence of the bounding box of the word *picote* is highligted in blue.

R is illustrated in Figure 3. The large bold rectangle depicts a document with text rendered inside. Outside the rectangle are words whose position (and bounding boxes) are equivalent: the position (x, y) of any entity within the document (such as the word *picote* in blue) is equivalent to all positions $(x - kw, y + kh)$ outside the document.

Interpolation of Positions

We interpolate the position of all entities (paragraph, block, word) in a top-down hierachical order. An entity whose initial and final positions are $p_0(x_0, y_0)$ and $p_1(x_1, y_1)$ is interpolated using either *direct* or *modulo* interpolation:

• *Direct interpolation:* p_t is obtained by linearly interpolating between p_0 and p_1, both of which are within the viewport. This results in a simple translation (Fig. 4b).

• *Modulo interpolation:* p_t is obtained by linearly interpolating between p_0 and the position $p_1^*(x_1^*, y_1^*) \in \overline{p_1}^R / y_1^* = y_0$. This results in the text moving horizontally and appearing on a different line (Fig. 4a,c).

More specifically:

• *Paragraph position* is always directly interpolated from p_0 to p_1, as paragraphs verify $x_t = 0$ and only shift vertically.

• *Block position* is directly interpolated if p_0 is closer to p_1 than to p_1^*. Otherwise it is modulo-interpolated. This strategy effectively minimizes travel distance *(R4)*.

• *Word position* is always modulo-interpolated in order to prevent words from shifting vertically within stable regions of text *(R3)*. Word-level interpolation effectively interpolates word spacings, which can be different between two revisions if for example text is justified.

Shrink and Grow

We convey deletion and insertion by having blocks shrink or grow. We describe the process for insertions – for the deletion case, indices 1 and 0 must be swapped.

We grow all inserted blocks by setting $q_0 = p_0$ and linearly interpolating q in a way that minimizes relative travel distance. Let $p_1 = (x_1, y_1)$. We define q_1^* as the element of $\overline{q_1}^R$ whose y-coordinate is $y_1 + h$, with h being line height. We then use a direct or modulo interpolation:

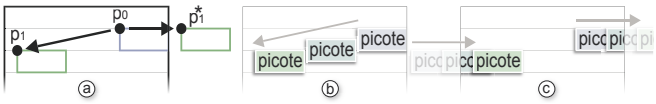

Figure 4. (a) The two possible interpolation paths for a block. Here a direct interpolation (b) yields more motion than a modulo one (c).

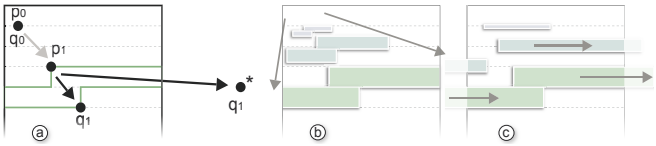

Figure 5. (a) The two possible growing behaviors for a block. Here direct interpolation (b) is shorter than modulo interpolation (c).

• *Direct interpolation*: if p_1 is closer to q_1 than to q_1^*, q is linearly interpolated between q_0 and q_1. The block grows and its layout is preserved (Fig. 5b).

• *Modulo interpolation*: if q_1^* is closer, q is linearly interpolated between q_0 and q_1^*. The block grows to the right as if it was a single long line (Fig. 5a).

We additionally set font size, line width and line height to zero when $q_0 = p_0$ and interpolate them linearly.

Since the geometry and rendering attributes are known for every t the document can be rendered for any animation step. Text is rendered normally except words are rendered multiple times when they intersect the document boundaries: a word of bounding box (p, q) is rendered inside all bounding boxes belonging to $\overline{(p, q)}^R$ that intersect the viewport.

Highlights

To reinforce the changes, we highlight text blocks that are inserted and deleted using the standard diff coloring scheme (see Fig. 1). To remain consistent with the rendering of the original revisions, highlights are shown only for $t \in]0, 1[$. Color opacity is animated in order to preserve the smoothness of the transition when text appears and disappears.

Vertical Stabilization

Changes in a document can affect the subsequent document layout and cause confusion: if part of a document is shown in a viewport, text may shift upwards or downwards even if it has not changed within the viewport. To address this issue, we perform *vertical stabilization* as follows:

1. Build the list of all visible words. A word is visible if it is within the document's viewport in the two revisions,
2. Compute the vertical motion ΔY of each of these words and average their motion into ΔY_{avg},
3. During the animation, translate the document vertically by linearly interpolating its original position Doc_Y with the final position $Doc_Y - \Delta Y_{avg}$.

As both word trajectories and stabilization are linear, a document that does not change within the viewport will not move at all. This approach also minimizes the sum of all word motions, which is consistent with requirement *R4*.

Timing

There are two aspects of animation timing: *relative timing*, the timing between individual animated components, and *overall timing*, the timing of the entire animation.

Relative Timing

Edit operations can be animated in parallel or sequentially. Although one study recommends sequential animations for complex visual changes [10], we found parallel animations to be the best choice for several reasons: 1) sequential animations take time and we want text animations to be brief; 2) the sequence of editions between two revisions is unknown thus any particular animation sequence can be misleading; 3) parallel animation is more consistent with requirement R4. For example, when a text portion is replaced (i.e., a delete + an insert), the text afterward appears more stable if the two operations are played in parallel rather than in sequence.

We therefore show all operations within a transition in parallel. However, we sequentially play transitions across several revisions. This combination of parallel and sequential animations has the advantage of clearly showing the granularity of document revisions within the document history.

When deciding about the timing of highlights vs. geometry, we observed that it was preferable to color-highlight text deletions slightly in advance, so that the user could anticipate where the text would disappear. Text insertions should also remain highlighted for some time after the text animation to allow for further examination. Note that these two effects, *deletion anticipation* and *insertion persistence*, are reversed when an animation is played backwards.

Overall Timing

Overall timing involves both the dynamics of the animation and its duration. With respect to the dynamics, we used a slow-out effect: the whole animated transition starts fast and ends up slow. Slow-out has been said to contribute to the realism of animations [5]. In our case, it also brings practical advantages. First, it makes edit highlights remain much longer on the screen without delaying the animation – users must wait for the "anticipation" highlights but not the "persistence" highlights. Moreover, when several revisions are shown in sequence, slow-out produces series of accelerations and decelerations that help distinguishing between revisions and reinforces the sequential nature of edit histories.

The second aspect of overall timing is animation duration, which should depend on the amount of changes (R6). Ideally, animations should be slow enough for changes to be clearly understandable. Our observations further suggested that showing all animations with the same duration tended to yield visual discomfort due to their varying subjective speed and length, even when browsing histories *with no intention of understanding changes*. Since our goal was only to provide a basic support for adaptive animation duration, this led us focus on the perceptual level (i.e., keeping subjective speed constant) rather than on the cognitive level (i.e., ensuring all changes are understood). The cognitive level is likely much more complex, and it is reasonable to assume that perceptual normalization will help cognition.

Although subjective animation speed and duration have been occasionally studied in specific cases [13, 8], we are not aware of any general perceptual model of animation speed. We hence devised a simple method based on the *largest relative motion*:

1. Build a list of visible words. Here we define a visible word as a word that is visible at least $k\%$ of the time (we used $k = 30\%$).
2. Compute the largest relative motion ΔD_{max} by taking the largest difference between all motions of visible words.

Due to animation's linearity, the largest relative speed ΔV_{max} that can be observed on the screen is $\Delta V_{max} = \Delta D_{max}/T$, with T being the duration of the animation. So by keeping ΔV_{max} constant across transitions we normalize the perceptual speed of the animation and derive the appropriate T.

More specifically, we used the following more general formula: $T = a * \Delta D_{max} + b$. Optimal values for a and b were approximated by having one of the authors adjust the speed of different animations so they all seem to have a similar speed that is "not too fast nor too slow". Then we performed a linear regression to compute the constants, deriving $a = 0.37$ ms $\cdot$ pixels^{-1} and $b = 100$ ms.

As animations need to be short in order to not interfere with user interaction, we decided to cap animation length to a predefined threshold of 500 ms. So there will be a complexity limit above which animations will be difficult to understand. We believe having such a limit is acceptable, as there will always exist extreme cases that are inherently difficult to comprehend even with very long animations (e.g. changes involving 90% of the document).

Like most GUI designs, our adaptive timing method is the product of a heuristic approach. We do not claim to provide a scientifically valid model of subjective animation speed, which is outside the scope of this article. However, we did find that simple normalization methods like the one proposed can produce very acceptable results and dramatically reduce the visual discomfort previously mentioned. Although no user study has been conducted to confirm this particular point, the mere fact that no subject mentioned animation speed as being a problem in our experiment is promising.

THE DIFFAMATION SYSTEM

As a proof of concept we have developed Diffamation, a complete document history navigation system. Diffamation stands out from current tools due to its elegant combination of minimalistic yet very effective navigation techniques. We show how such a design has been made possible by the use of animated transitions and how it covers tasks T1-T6.

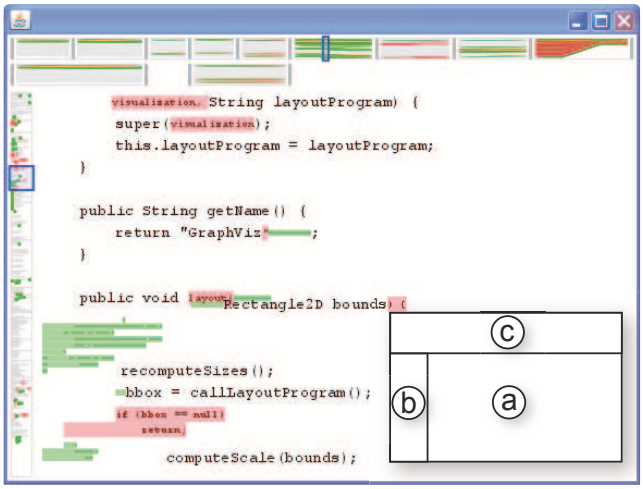

Figure 6. Screenshot the Diffamation system during a transition: (a) the document view, (b) the overview scrollbar and (c) the timeline.

Diffamation can load any type of text document history (articles, code, etc) in the form of text files suffixed with revision number. It then allows to rapidly navigate between revisions using the mouse or the keyboard. The GUI only involves three components (Fig. 6): the document view (a) and two widgets for navigating in space and time: the overview scrollbar (b) and the timeline (c).

The Document View

The document view displays the currently selected revision (Fig. 6a). When a new revision is selected this view is animated, allowing the user to observe changes at the document's level of detail (supporting T2). Pan and Zoom mouse interactions further allow to focus on different parts of the document and change its level of detail (T4,T5).

The Overview Scrollbar

A scrollbar is displayed on the side of the document (Fig. 6b). We use a fixed scale factor for the scrollbar, computed from the height of the longest document revision in the history. This is because we found that users were distracted by changes in scrollbar's sensitivity while browsing the history.

The Diffamation's scrollbar shows an overview of the document similarly to [11]. Text is shown in gray to provide an overview of the document's layout. The overview shows the animated transitions across the entire document when they occur on the viewport (T3,T4). The user can see the document shrink or grow. All insertions and deletions are shown with miniature highlights and the user can drag the scrollbar's thumb to any highlight to get more details.

The scrollbar's thumb moves during transitions as a result of the stabilization (T5), and the document's overview underneath moves along with it and appears to "drag" it. However, our stabilization mechanism is asymmetric: one can go from revision A to B and find that the viewport has changed when returning to A. To address this, each time a new revision is displayed we store its viewport and animate back to the same viewport when the revision is displayed again.

The Timeline

The timeline shows an overview of the revision history (Fig. 6c). Revisions are shown as vertical marks and a blue cursor depicts the currently viewed revision. In order to provide landmarks and give a rough idea of where large changes occur (T1), the spacing between revision marks varies according to the edit distance: $d = k * log(1 + E)$, with E being the total number of characters edited.

A summary of edits is also shown between each pair of successive revisions (T3). Summaries are built by rendering animations using a slit-tear approach [21], except alpha compositing is used instead of single pixel extraction. Overall the visualization produced is akin to a History Flow [23] and allows the user to see the evolution of the document at a glance (T4). It also helps navigation: for example, the user can solely focus on large deletions or edits occurring at the end of the document.

Clicking on a revision mark changes the revision in focus and triggers an animated transition. Users can thus easily compare any pair of revisions (T2). The position of the revision cursor reflects the progress of the animation. Conversely, it can be dragged like a seeker bar to control the playback of the last animation.

The revision cursor can also be rapidly moved one revision forward or backward with the arrow keys. We prevent visual discontinuities by queuing animations and accelerating all of them except the last one. So when a new revision is selected, current animations will quickly finish before the new animation is started. The whole document history can be fast-forwarded by holding the arrow key pressed (T4,T5).

Finally, the timeline also supports backtracks in the editing process, such as code reverts and edit wars (T6). Although analytic methods have been proposed to detect controversies [4], we adopt a simpler approach. We compute a subset of the revision history based on the shortest edit path between the first and last revisions, and use it to display "edit shortcuts" underneath the history (Fig. 7a). When the user clicks on the shortcut view, the timeline fills the gaps in the shortest path and shows deviations from this path on top (Figure 7b). These two modes allow the user to focus either on the entire edit process or on the persistent contributions.

Our prototype currently supports histories of about 100 revisions. Supporting more revisions will require adding pan-and-zoom capabilities to the timeline.

Space-Time Navigation

Diffamation supports space-time navigation by using mouse movements orthogonal to the scrollbar [1]. Dragging the thumb left or right beyond a certain threshold transfers control to the timeline's revision cursor. This allows for rapidly switching between space and time navigation. For example, one can watch an animated overview of changes in the scrollbar, then move the scrollbar to a location where a change occurred, then play the animation again, then scroll further, and so forth, without releasing the mouse button.

Figure 7. Edit history of the Wikipedia article *Copernius (Disambiguation)*, as shown by the timeline. (a) Focus on the complete edit history with the occasional shortcuts displayed underneath, (b) Focus on the shortest edit path with the occasional deviations displayed above.

EVALUATION

Although text animation can be used to examine the differences between two specific revisions, it is meant to support the browsing of multiple revisions. Therefore, we did not focus on comparing text animation with static diff methods, but focused on assessing the benefits of animated transitions (vs. no animation) when switching between document revisions. Our experimental task involves tracking changes in a portion of text across multiple revisions. Such a task is representative of real-world browsing tasks, e.g., finding what happened to one's contributions in a Wikipedia article and when. We also added a control condition of static (all-in-one) diff visualization in order to better isolate the effects of animated transitions. Because we are solely interested in the benefits of animated transitions, we deactivated scrolling and stabilization, and removed the scrollbar and the timeline.

Task Description

The task, inspired by tasks T2 & T5, required users to follow the evolution of a particular sentence in a document (*sentence of interest*) and identify the presence of changes across 5 successive revisions of the document.

More specifically, for each document participants compared the sentence in two consecutive revisions at a time (revisions **A** and **B**). They were first shown revision **A** and the sentence of interest was highlighted for 2 seconds. The participant then had to press the right arrow key to display revision **B** and the experiment timing started. If she was confident there was no change in the sentence, she proceeded to the next comparison by hitting the right arrow while in revision **B**. If the participant spotted a change in the sentence, she indicated it by pressing the spacebar.

By either pressing the right arrow or spacebar at version **B**, the participant indicated that they completed the trial and could proceed to the next revision comparison. The old revision **B** became the new revision **A** and the sentence of interest was highlighted again.

Participants went through 5 revisions per document (i.e., conducted 4 trials of comparisons/transitions between revisions 1→2, 2→3, 3→4, 4→5). They were instructed to not miss any changes, and to be as fast as possible without compromising accuracy.

After a trial started, participants were permitted to backtrack (go back-and-forth between **A** and **B**) using the left and right arrows. They could also explicitly re-highlight the sentence of interest using the *H* key, but only while in revision **A**, to ensure that users would search and find the sentence in **B**. They were told to use re-highlighting only when they had completely forgotten either the sentence or its location in **A**.

Auditory feedback was used to prevent mode errors (at the beginning of a trial, when hitting spacebar or hitting a wrong key). A widget was displayed to the right of the document indicating the revision currently being viewed (**A** or **B**), and its position within the 5-revision history of the document.

Dataset

The characteristics of actual text documents and the nature of their edit history are extremely variable and are difficult to formalize. It hence seems hard to design an experiment on text history nagivation that is both fully controlled and ecologically valid. We adopted a middle-ground approach by randomly sampling a popular dataset (Wikipedia), but filtering out data to dismiss extreme or uninteresting cases and get a degree of control over the difficulty of the tasks.

We selected a pool of Wikipedia articles using the Website's *random article* feature. Only articles between 1500 and 5000 characters and with history of 5 or more revisions were selected. This represents about 50% of Wikipedia articles – the remainder mostly consists of very short and relatively unimportant articles. We downloaded the last 50 revisions (less for shorter histories) of each of the selected articles and converted them into plain text.

Based on this corpus, we built a dataset of about 400 *scenarios*. A scenario is defined by 1) a series of 5 consecutive article revisions, i.e., 4 transitions/comparisons, and 2) a sentence of interest. Scenarios were selected randomly from the corpus according to the following criteria:

1. A scenario involves at least 100 edited (inserted or deleted) characters that are visible on the screen [5],
2. The sentence of interest is between 50 and 150 characters and always visible,
3. During a transition between revisions, the characters of the sentence of interest are either not edited, or edited by at least 5% to ensure clear changes.
4. The sentence of interest cannot be split, merged with another sentence, or fully deleted during transitions.

These criteria allowed us to prune scenarios with little or no edit activity that are arguably less interesting for comparing

[5]Some of the articles fit on the screen once rendered and some of them do not. To avoid scrolling tasks that complicate the experiment setup, we ignored the lower invisible part of the articles.

diff visualization techniques (criterion 1), sentences of interest whose changes are too easy or hard to spot (criteria 2, 3), and ambiguous changes that can confuse users (criteria 3, 4).

We expect that the difficulty of tracking the sentence of interest will greatly depend on how far the sentence moves between revisions. We thus decided to balance data per technique in terms of the motion of the sentence of interest. So based on the generated dataset of 400 scenarios, we randomly formed *scenario groups* to be used with each technique in our evaluation. A scenario group:

1. Contains 18 scenarios of 4 transitions between revisions, i.e., $18 * 4 = 72$ transitions. Among these:
   (a) 24 involve *No Motion* of the sentence of interest,
   (b) 24 involve a *Small Motion* from 1 to 52 pixels [6] due to edits occurring before the sentence,
   (c) 24 involve a *Large Motion* of 104 pixels or more,
2. The sentence of interest is edited in exactly 8 (11%) of the transitions (value derived from statistics on the dataset),
3. To avoid learning, scenarios never share the same article.

The goal of these additional criteria was 1) to get more balanced scenario sets across techniques, and 2) to be able to statistically assess the effects of the visual motion of the sentence of interest. The choices of visual motion range were derived by statistics on the dataset pool: most sentences do not move, and from those that move approximately half do so by 1-52 pixels, while the rest by over 104 pixels.

Techniques

To understand the effect of animated transitions (A), we evaluated our technique for animated transitions between revisions against a simple slideshow (S) of consecutive revisions without any transitions. Since we use color highlighting to indicate changes, we also tested the effect of highlighting (H). Thus the techniques evaluated were:

AH: our proposed technique of animated transitions with color highlighting during the transition,

A: animated transitions without the color highlighting,

SH: a slideshow technique derived from the traditional all-in-one view that represents changes between consecutive revisions highlighted in color,

S: and finally, a base case slideshow technique with no change highlighting and no animated transition between revisions.

Transitions lasted from 50 ms to 500 ms for *AH* and *A*, and were instantaneous for *SH* and *S*.

Participants and Apparatus

16 participants (3 females) aged 24-35 took part in our study. The majority (15/16) had experience in using text versioning systems and collaborative writing tools.

[6]The values are the distance travelled by the first character of the sentence between the two revisions and include both horizontal and vertical motion. Text size is given in the *Apparatus* section.

The experiment ran on an Intel Xeon 5160 with a 30" monitor of 2560×1600 pixels resolution. The document view was 1200×1600 pixels large. Text was displayed in black on a white background using an anti-aliased proportional font of 16 pixels, left justification, a line height of 26 pixels including spacing, and an effective line width of 1078 pixels. Animated transitions were played back at 60 fps.

Design

A repeated measures within-participant full factorial design was used. The independent variables were selection *Technique*: *AH*, *A*, *SH* and *S*; and *Motion* of the sentence of interest: NoMotion, SmallMotion, LargeMotion.

Participants were randomly assigned to one of 4 groups. Each group used all 4 techniques, in an ordering balanced using a Latin square. Each technique was assigned a *scenario group* as defined in the section *Dataset*, and the scenario group used per technique was mixed between participants. The experiment consisted of:

16 participants × 4 tech × 3 motions × 24 transitions
= 4608 trials

Prior to each technique users were given instructions and performed short warm-up sessions (22 trials) to familiarize themselves with the technique. It took on average 45 minutes to complete the experiment and afterwards participants filled out a brief questionnaire eliciting their preferences.

Performance Measures

We compared the techniques using several measures. *Time* is the time elapsed from the moment the subject first saw revision **B** until giving their answer. This includes backtracking, as well as the animation time for *AH* and *A*. *ErrorRate* is the percentage of trials where the user answered incorrectly as to whether a change occurred in the sentence of interest during a transition. To further understand how hard it was for users to follow changes between revisions, we counted another two metrics: number of *Backtracks*, the times the user went back-and-forth between two revisions before answering; and the number of *Re-highlights*, the times the user asked the system to re-highlight the position of the sentence of interest after the trial had started.

Results

Trials were marked as outliers when *Time* was beyond 3 standard deviations from the mean for a given subject and condition and the remaining trials were aggregated for analysis.

Time

Analysis of variance showed a significant difference between techniques for *Time* ($F_{3,36} = 69.6$, $p < .0001$). Post-hoc pair-wise means comparison (all adjustments Bonferroni) showed animated highlighting (*AH*) being significantly faster than slideshow with highlighting (*SH*), and all techniques were significantly faster than the simple slideshow technique (*S*) (all $p < .05$). No other significant pairs were found. Mean times for *AH* were fastest 1.597sec, followed by *SH*: 2.164sec, *A*: 2.166sec and *S*: 4.281sec.

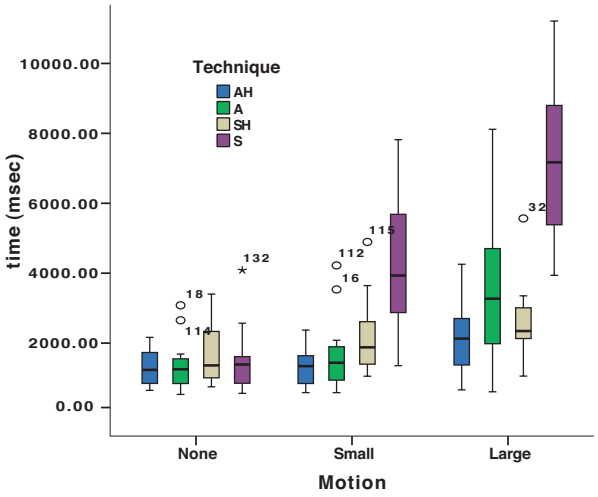

Figure 8. Boxplots (median, range) of time by technique and motion.

The *Motion* of the sentence of interest had a significant effect ($F_{2,24} = 95$, $p < .0001$) on *Time*, with NoMotion resulting in the fastest times, followed by SmallMotion and LargeMotion. Mean times for the 3 Motions were significantly different (all $p < .0001$), indicating that our definition of visual motion corresponds to task difficulty as indicated by the time needed to perform tasks per motion type. A significant *Technique* x *Motion* interaction was present ($F_{6,72} = 29.7$, $p < .0001$). Pair-wise means comparison (all $p < .05$) showed that for SmallMotions *AH* was significantly faster than *SH*, but there was also a trend ($p = .06$) for *A* to be significantly faster than *SH*. These 3 techniques were significantly faster than the base case slideshow only condition (*S*). It thus seems that the animated transition conditions benefit the task more in the SmallMotions than highlighting in the slideshow+highlighting technique. In the LargeMotions *AH* was significantly faster than *A*, but not *SH*. Again all techniques were significantly faster than *S* (all $p < .05$). This indicates that for LargeMotions the benefit of the animated transition drops and the color highlighting may play a more important role. For the NoMotion, all techniques were similar (even *S*), indicating that keeping track of the state of a sentence that has not moved is as easy for all techniques.

Error Rate

As Error Rate data does not follow the normal distribution, we conducted non-parametric tests. Similar tests are used for *Backtracks* and *Re-highlights*. The Friedman's test showed a significant effect of Technique ($p < .05$) on *ErrorRate*. Overall, 1.5% of trials in *AH* were errors, 1.6% in *SH*, 2.8% in *S* and 3.2% in *A*. Pair-wise means comparisons using the Wilcoxon test (all $p < .05$) showed that *AH* is less error prone than *A* and *S* with no other significant pairs. Examining *ErrorRate* per *Motion*, the techniques are similarly error prone for the NoMotion and SmallMotion, but the mentioned effects appear in LargeMotions. This indicates that color highlights help user accuracy in harder tasks.

Backtracks and Re-highlights

There was a significant effect of number of *Backtracks* per *Technique* ($p < .05$), with *AH* having the lowest mean of backtracks per transition (0.14), followed by *SH* (.22), *A* (.37) and *S* (.82). Pair-wise comparisons showed *AH, A, SH* to require less backtracks than the simple *S* condition. Moreover *AH* required significantly less backtracks than *A*, with no other significant pairs.

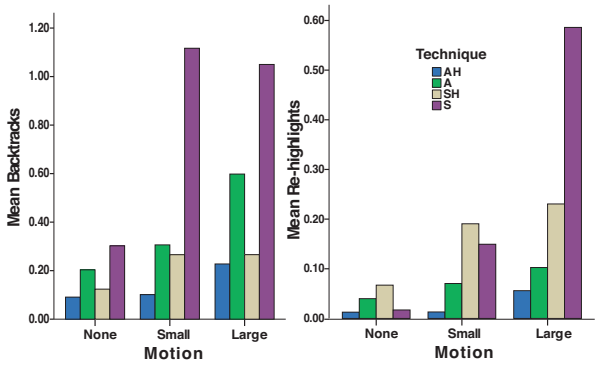

Figure 9. Means for Backtracks (left) and Re-Highlights (right) by technique and motion.

There was a significant effect of number of *Re-highlights* per *Technique* ($p < .05$), with *AH* having the lowest re-highlighting per transition (0.02), followed by *A* (.07), *SH* (.16) and *S* (.25). Pair-wise mean comparisons showed significantly less re-highlights between the animated (*AH, A*) and the non-animated conditions *SH, S*.

Subjective Comments from Participants

For the specific tasks, the majority of participants (14/16) ranked *AH* as their preferred technique, while 2/16 ranked *SH* higher. Participants mentioned that the animation helped them understand the new location of the sentence of interest thus increasing their efficiency, while the color highlighting helped them identify changes, thus increasing their accuracy. They all stated they would like to use *AH* in a real system as a complement to their current diff tools.

Discussion

In our evaluation we set out to investigate if animated transitions between document revisions are helpful. For tasks related to T2 & T5, we found that the proposed animated transition technique with highlighting (*AH*) was the best performing and best received technique by users.

In terms of efficiency, our results show that the animation time overhead not only did not slow users down, but in the case of *AH* the benefit of animation was such that the overall times for this technique were faster than slideshow (*SH*, *S*). We found that the benefit of animation was especially important with small movements of the sentence of interest, where the user can easily keep track of the animation. Although *AH* is still faster, this benefit drops for large movements, as the user needs to follow larger text movements and the animation lasts longer.

In terms of accuracy, *AH* was the least error prone technique followed closely by *SH*, especially when users attempted to understand changes with large movements of the sentence of interest. This indicates that color highlighting rather than animation positively affects accuracy.

The lower number of backtracks in the *AH* condition shows that our technique gives the most confidence to users when spotting changes and requires less verification. The lower number of backtracks in both *AH* and *SH* suggest that color highlighting plays a major role, and can explain the lower error rates and task completion times for these techniques, as backtracks take up time.

Re-highlights were used by participants to redisplay the position of the sentence of interest in the first revision. The lower number of re-highlights with the two animated techniques (*AH*, *A*) indicates that users were more aware of the position of the sentence between revisions. Thus animated transitions seem to help users keep track of changes across multiple revisions in the long-run.

In summary our proposed animated & highlighting technique (*AH*) for transitioning between document revisions outperformed all other techniques. Animated transitions help users track the position of text across revisions, while color highlighting aids in spotting changes in the text. These findings were backed-up by subjective comments from participants.

CONCLUSION AND FUTURE WORK

We have presented a technique for smoothly animating changes between text revisions, and have shown this technique to be effective for tasks involving tracking changes in portions of text over time. We have also described the Diffamation system that supports navigation in text document histories through animated transitions. Diffamation can be used for example to get a quick overview of the entire history of a Wikipedia article or to see what happened to one's contributions. Our approach complements classical diff visualizations: once moments of interest have been identified, they can come in useful to compare two given revisions in detail.

Apart from Diffamation, text animated transitions can be used for other novel interfaces. For example, when opening a shared document with a system such as Google Documents, having a smooth animation of the changes would allow users to quickly review what happened since they last saw the document. Smooth animations could also be useful during the editing of text document, e.g., when deleting or pasting large portions of text.

In future work, we plan to combine our animation technique with static diff visualizations. One example would be to smoothly transition between side-by-side diff views of different pairs of revisions. We also plan to support rich text.

REFERENCES

1. C. Appert and J.-D. Fekete. Orthozoom scroller: 1d multi-scale navigation. In *Proc. CHI'06*, 21–30, 2006.
2. B. B. Bederson and A. Boltman. Does animation help users build mental maps of spatial information? In *Proc. INFOVIS'99*, Washington, DC, USA, 1999.
3. A. Bezerianos, P. Dragicevic, and R. Balakrishnan. Mnemonic rendering: an image-based approach for exposing hidden changes in dynamic displays. In *Proc. UIST'06*, 159–168, 2006.
4. U. Brandes and J. Lerner. Visual analysis of controversy in user-generated encyclopedias. *Information Visualization*, 7(1):34–48, 2008.
5. B.-W. Chang and D. Ungar. Animation: from cartoons to the user interface. In *Proc. UIST'93*, 45–55, 1993.
6. S. Eick, J. Steffen, and E. S. Jr. Seesoft - a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering*, 18(11):957–968, 1992.
7. C. Friedrich and P. Eades. Graph drawing in motion. *J. Graph Algorithms Appl.*, 6(3):353–370, 2002.
8. C. Harrison, B. Amento, S. Kuznetsov, and R. Bell. Rethinking the progress bar. In *Proc. UIST'07*, 115–118, 2007.
9. P. Heckel. A technique for isolating differences between files. *Commun. ACM*, 21(4):264–268, 1978.
10. J. Heer and G. Robertson. Animated transitions in statistical data graphics. *IEEE Trans. on Vis. and Comp. Graphics*, 13(6):1240–1247, 2007.
11. W. C. Hill, J. D. Hollan, D. Wroblewski, and T. McCandless. Edit wear and read wear. In *Proc. CHI '92*, 3–9, 1992.
12. J. W. Hunt and M. D. McIlroy. An algorithm for differential file comparison. Technical report, Bell Laboratories, 1976.
13. T. Igarashi and K. Hinckley. Speed-dependent automatic zooming for browsing large documents. In *Proc. UIST'00*, 139–148, 2000.
14. B.-W. C. Jock, B. wei Chang, J. D. Mackinlay, P. T. Zellweger, and T. Igarashi. A negotiation architecture for fluid documents. In *Proc. UIST'98*, 123–132, 1998.
15. D. E. M. John W. Ratcliff. Pattern matching: The gestalt approach. *Dr. Dobbs Journal*, 46–51, July 1988.
16. C. Klein and B. B. Bederson. Benefits of animated scrolling. In *Proc. CHI'05*, 1965–1968, 2005.
17. J. C. Lee, J. Forlizzi, and S. E. Hudson. The kinetic typography engine: an extensible system for animating expressive text. In *Proc. UIST'02*, 81–90, 2002.
18. T. Nakamura and T. Igarashi. An application-independent system for visualizing user operation history. In *Proc. UIST'08*, 23–32, 2008.
19. M. Shanmugasundaram and P. Irani. The effect of animated transitions in zooming interfaces. In *Proc. AVI'08*, 396–399, 2008.
20. J. Tam and S. Greenberg. A framework for asynchronous change awareness in collaborative documents and workspaces. *Int. J. Hum.-Comput. Stud.*, 64(7):583–598, 2006.
21. A. Tang, S. Greenberg, and S. Fels. Exploring video streams using slit-tear visualizations. In *Proc. AVI'08*, 191–198, 2008.
22. W. F. Tichy. Rcs—a system for version control. *Softw. Pract. Exper.*, 15(7):637–654, 1985.
23. F. B. Viégas, M. Wattenberg, and K. Dave. Studying cooperation and conflict between authors with history flow visualizations. In *Proc. CHI'04*, 575–582, 2004.