SlideDiff: Animating Textual and Media Changes in Slides

Laurent Denoue, Scott Carter, Matthew Cooper
FX Palo Alto Laboratory, Inc.
Palo Alto, CA
(denoue,carter,cooper)@fxpal.com

ABSTRACT

SlideDiff is a system that automatically creates an animated rendering of textual and media differences between two versions of a slide presentation. While previous work focused on either textual or image data, SlideDiff integrates both text and media changes, as well as their interactions, for example when adding an image forces nearby text boxes to shrink.

Given two versions of a slide (not the full history of edits), SlideDiff detects the textual and image differences, and then animates the changes by mimicking what a user might have done, such as moving the cursor, typing text, resizing image boxes, adding images. This editing metaphor is well known to most users, helping them better understand what has changed, and fosters sense of connection between remote workers, derived from a better understanding of the revision process as well as its results.

After detection of text and image differences, the animations are rendered in HTML and CSS, including mouse cursor motion, text and image box selection and resizing, text deletion and insertion with its cursor. We discuss strategies for animating changes, in particular the importance of starting with large changes and finishing with smaller edits, and provide details of the implementation using modern HTML and CSS.

CCS CONCEPTS

• **Human-centered computing** → **Visualization**; • **Information systems** → *Collaborative and social computing systems and tools*;

KEYWORDS

slide changes, animation, enterprise messaging

ACM Reference Format:
Laurent Denoue, Scott Carter, Matthew Cooper. 2018. SlideDiff: Animating Textual and Media Changes in Slides. In *Proceedings of ACM Document Engineering (DocEng'18).* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

In today's enterprise communication tools such as Slack, users often exchange links to document revisions, updating each other with messaging indicating what each person changed, along with links to new versions. Sometimes, a small preview thumbnail showing the first page or slide is embedded in place of the link to represent the uploaded file. But this is rarely sufficient to communicate what was changed in the most recent version. Even if the user opens the new document they can only detect changes if change tracking is turned on in and if they find and open the view showing those changes. Furthermore, this requires the desktop document editing application and may preclude for instance viewing the document on a smartphone.

In this article, we propose SlideDiff, a system that replaces the standard, static preview thumbnail generated when a user uploads a new version of a document to a multimedia chat tool by an animated thumbnail that shows what changed in the document (see Figure 1). Our initial focus is on slide decks, but this concept can be extended to other types of documents as well. Figure 1 shows SlideDiff implemented inside an enterprise chat system.

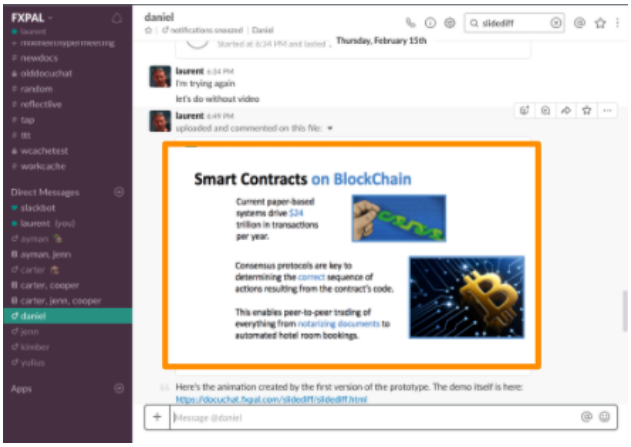

Figure 1: SlideDiff generates animated previews of slide changes that allow users of enterprise messaging tools to quickly understand what changed in newly posted versions or links to slide decks. See video https://youtu.be/JPZySPSEQc/.

There are many possible ways to visualize and/or animate changes between two or more versions of a slide. In order to make the animation user-friendly and easy to understand, SlideDiff mimics editing operations with which every user is familiar to generate animations as if they had been performed by another user. Visualizing the process of slide revision can more effectively communicate the scope of the changes as well as when individual changes interact to generate a composite revision. It can also communicate the means by which other team members can subsequently refine changes.

DocEng'18, August 2018, Halifax, Canada

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

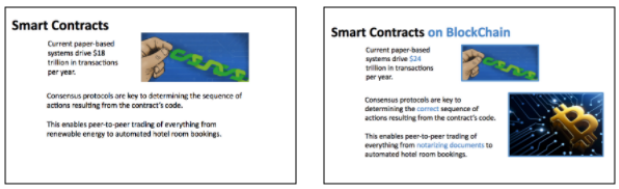

Figure 2: Two versions of a slide, where an image was added and text was changed.

2 RELATED WORK

We describe related work on visualizing the version history of various document types. Enhanced Thumbnails [15] showed that users preferred a combination of text and images to better retrieve previously seen documents (in their case web pages), as opposed to either a static thumbnail (e.g., what the current Slack shows of a document) or just plain text (e.g., the result of a regular diff algorithm).

Diffamation [4] examined the use of smooth text animations to reveal changes in a text document. Unlike SlideDiff, Diffamation focuses only on textual content of the document and does not handle graphics and other important media commonly found in presentations. The visualizations also do not produce embeddable thumbnails, but rather enhance the actual document editor with an overlay showing the changes using animation. They also do not summarize important changes.

SketchStory leverages the narrative storytelling attributes of whiteboard animation with pen and touch interactions, allowing users to present and communicate insights to an audience by telling a story [10]. Recently, several online tools such as VideoScribe [14] and TruScribe [13] have compelled many users to produce automatic whiteboard animations.

Image revision systems, such as Chen et al.'s work [3], are specifically designed to show image differences at the bitmap level. They do not focus on textual elements or the global spatial attributes of objects' placement. Furthermore, they do not abstract the changes, but rather provide a timeline that replays all edits.

Acrobat Pro offers an option to compute the difference between two PDF files, but produces either a fine-grained text-only difference, or an interface similar to Microsoft Word or PowerPoint with tracked changes. For graphs, Jupyter Notebooks [8] interact with GitHub to give "visual" differences, providing users with a classic side-by-side view of two versions of a graph, or swipe bar that users control from left to right to reveal changes.

Visual difference tools such as DiffChecker [6] work on text and PDF documents, but always produce traditional side-by-side windows showing the old and new versions. They do not depict the revision process itself. Again, they only work on textual content. By contrast, SlideDiff tracks changes to text and non-text content alike, it attempts to reconstruct and visualize the editing process itself, and provides animations that are embeddable in chat-based communication tools and viewable on devices without requiring the document's source application.

Like objects in the real world, the Self user interface [2] uses lifelike arcs to give interface elements a livelier and more appealing character than a simple linear path would. SlideDiff also uses this principle to move the fake mouse cursor during the animation.

3 VISUALIZING SLIDE REVISIONS

In this section, we detail how slide changes are computed, how we map these changes to several GUI editing actions, and finally the steps used to generate animated slide changes.

3.1 Extracting Slide Content

The first step is to identify changes between two versions of the slides. Previous work describes techniques and a user-interface for managing differences between many slide decks, as well as a set of similarity measures including text edit distance, slide and image IDs found in the source PowerPoint file, and image bitmap distance using the mean square error [7].

Unlike this work, SlideDiff assumes that the two slide decks are given as input: in our application of enterprise messaging, one deck is the last posted and the other is the previously posted link. When users upload files rather than link to them, we use the filename. Given two decks, slide pairs are found by using text edit distance only so that the system remains generic and can be used with any file type rather than just PowerPoint by exporting the decks to PDF.

For the scope of this paper, we focus on finding differences between two slides only, and explain how we extract and animate slide changes between two slides. In practice, if several slides were edited between versions, the preview could stack them visually (e.g., 4 thumbnails if 4 slides were changed), or chain them in time (generate 4 animations and combine them into a longer animation).

Consider the two slides S1 and S2 in Figure 2, where text was edited (shown in blue), one image was resized and another added, both causing nearby text boxes to resize.

In some cases, the application used to author the slides records changes between two versions, e.g., tracked changes in Microsoft PowerPoint or revision history in Google Docs. In order to provide a generic tool, we chose to convert the slides to PDF and directly perform change detection on the PDF files. For that, we modified Apache's PDFBox to extract text boxes along with image bounding boxes (see [5] for more details on the PDF extraction tool).

For a given pair of slides S1 and S2, the algorithm extracts:

(1) The textual content of both slides along with its attributes (known as attributed strings) into an HTML DIV that contains the textual content, static position and font attributes (size, color, family, style, weight)
(2) The image content of both slides into an image, represented by an HTML image tag where the image data is encoded as a *datauri* and the position and size of the image as style attributes

This content extraction thus produces two HTML files, each containing a set of statically positioned DIV and IMG elements.

3.2 Characterizing and Filtering Slide Changes

Once content is extracted, the algorithm maps the contents of the two HTML files to establish which DIV and IMG elements correspond to each other.

For textual content, the system uses the longest common subsequence algorithm [11] to map textual elements between the first and second slide and determine edits between each pair of attributed strings. Figure 2 shows the edits found in the text in blue.

For images, the algorithm examines both sets of images and pairs them if an approximate content match is found, regardless of scale and position. To accomplish this, each image is scaled to a 256x256 pixel image and then compared. Referring again to Figure 2, the top right image in S1 is found to match the top-right image in S2 even if it was scaled down.

For animation purposes, it is important to detect changes applied to the same underlying image: without an accurate match, the animation would make the first image disappear and the second one appear later, as opposed to slowly transitioning the box size or location of that "same" image.

Not all changes need to be shown for an animated preview. The system can be parameterized so that some changes are not rendered, such as:

(1) Small changes in image position, rotation, contrast, border thickness
(2) Small text attribute adjustments: (such as changing font size from 11px to 11.5px or normal to bold style)

This filtering step is important since we want to provide users with a short animation that quickly helps them assess what has changed.

Furthermore, the animation can be tailored to emphasize specific aspects of detected changes. For example, if a user on Slack mostly edits text, then she may be more interested in seeing textual changes in subsequent versions. Alternatively, a designer who mostly edits graphics in the document will want to see predominantly the graphical changes. Slack text messages can be sent to instruct SlideDiff about the kind of animation needed.

Finally, context-aware filtering [9] can be used to filter slide changes that are most likely to be interesting: if a user posts a link to an updated slide deck and adds a text message "check the algorithm slide", SlideDiff can be augmented with the DocHandles system [5] to tailor the animation for that slide only.

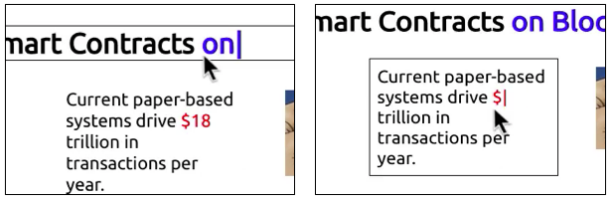

Figure 3: A fake mouse cursor moves toward areas where content will change; the text "on Blockchain" is typed and "$18" is deleted.

3.3 Animation Techniques

In order to create a compelling animation, SlideDiff mimics editing steps that a user would have taken. Inspired by Disney animation techniques [12], SlideDiff uses mouse motion as an important clue to establish intent and positioning. For example, in Figure 3, a fake mouse cursor moves to the text box where text will be added ("on" in blue). After, the mouse cursor is moved to the deleted area in the text box below ("$" in red).

Once the cursor arrives at the edit location, a fake cursor is inserted and text is either added or deleted, depending on the results of the previous step. To make the changes more prominent, SlideDiff uses blue to type insertions and red for deletions.

Similar to text, the system visualizes image repositioning by first moving the cursor toward the image. If its bounding box needs to be changed, the fake cursor goes to the bottom right corner and follows the resizing animation of the box, as if a user had resized the image by dragging one of its corners. Finally, the fake cursor goes toward the center of the image and moves it to its final destination.

New images are simply animated by moving the mouse outside of the canvas and bringing it back with the new image attached, landing on its final position at the correct dimension.

Sometimes, these animation techniques need to be combined, as explained below. For example, as in Figure 2, adding the new image to the slide causes the bottom left text box to shrink.

3.4 Animating Changes

In principle, several strategies can be used to animate changes. For example, one could:

(1) Leave the text and images boxes untouched, but change their textual content first
(2) Change the text and images boxes, and then change their content
(3) Change everything at once

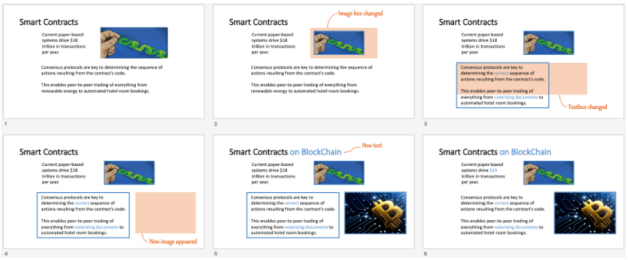

Figure 4: Two versions of a slide, where an image was added and text was changed.

An optimal strategy depends on the kinds of changes involved. Referring to Figure 4 bottom left, the new image forced the user to resize the text box, and then the box was inserted. One could choose to first show text changes, and then add the graphical elements, which will resize the text boxed with their final content. Alternatively, one could first change the boxes to fit the new graphical positions, and then change the textual content. Finally, one could execute both at the same time.

In practice, we found that showing the important changes first is preferable, ordered top to bottom. In Figure 4, the animation first changes the location and size of the top right image that was scaled down. It then reduces the size of the bottom left text box as the new image is inserted to its right. SlideDiff detects that the two

bounding boxes overlap, and animates them together: the effect emphasizes that the additional image pushes the text box as it is inserted into the slide. Similar effects were discussed in the Fluid Document architecture [1].

4 IMPLEMENTATION DETAILS

SlideDiff uses HTML CSS animations to produce the final animated previews. The slide extraction step produces an HTML document with simple text and image boxes, along with their content and bounding boxes using statically positioned * * elements.

```
<div class="slide">
  <div id="title" class="title" style="left:5%;top:5%;width:80%">
    Smart Contracts <div data-new="on BlockChain" class="added"></div>
  </div>

  <div id="text1" class="text" style="left:15%;top:18%;width:28%;">
    Current paper-based systems drive
    <div data-new="$24" class="replaced">$18</div>
    trillion in transactions per year.
  </div>
</div>
```

Figure 5: The text difference algorithm annotates the HTML version of slides; here new text "on BlockChain" will be added, and "$18" will be replaced with "$24".

Second, the slide change detection step annotates the *innerHTML* of text boxes with inserts, deletes, and moves by annotating the HTML using a *dataset* attribute and *class* name. For example in Figure 5, a text insertion is represented using * *. Similarly, a deleted text (and possibly replaced by new content) is annotated using * $18 *.

```
<img id="image1" src="image1.jpg"
  data-new="left:45%;width:25%" style="left:50%;top:20%;width:35%"
/>

<img id="image2" src="image2.jpg"
  data-new="scale(1.0)"
  style="transform:scale(0);left:60%;top:50%;width:30%"/>
```

Figure 6: Images are annotated using a dataset HTML attribute that the animation uses to position the image at its final destination.

Similarly for images, SlideDiff annotates the HTML using a *dataset* attribute that specifies their end position on the slide. For new images, their initial style can be set to be outside of the slide canvas area, or as shown in Figure 6 can be scaled to zero and have a new attribute with scale set to 1.

With this annotated HTML, the animation is then simply generated using JavaScript selectors and CSS transitions. First, each *<img>* and * * box is selected and a CSS transition of 1 second for the left, top, width and height added to the element, and the new styles found in the dataset are applied, causing these elements to move on the screen.

Next, text edits are applied: a JavaScript selector finds all "added" or "removed" * * elements, and the data attribute "data-new" is used by JavaScript to simulate a typewriter effect. Before the typewriter effect starts, the fake mouse cursor is animated to the location of the edit, found using the *getBoundingClientRect* method on the * * to be added, deleted or changed: this causes the cursor to move right over the text block to change.

5 CONCLUSION AND FUTURE WORK

Multimedia chat tools are an increasingly popular forum for workers to share and edit documents. However, they do not support this practice, making it harder for people using these platforms to keep track of changes between different document versions. Often, understanding document-related context can require opening documents in separate applications. We built a tool, SlideDiff, that automatically creates an embeddable animation of multimedia differences between versions of documents, which can allow multimedia chat users to assess changes in their documents. Having developed the tool, we next plan to test it in a real work context. We are also investigating how to expand the tool to show changes to several slides, as well as support other document types.

REFERENCES

[1] Bay-Wei Chang, Jock D. Mackinlay, Polle T. Zellweger, and Takeo Igarashi. 1998. A negotiation architecture for fluid documents. In *Proceedings of the ACM Symposium on User Interface Software and Technology*. 123–132.

[2] Bay-Wei Chang and David Ungar. 1993. Animation: From Cartoons to the User Interface. In *Proceedings of the ACM Symposium on User Interface Software and Technology*. 45–55.

[3] Hsiang-Ting Chen, Li-Yi Wei, and Chun-Fa Chang. 2011. Nonlinear Revision Control for Images. *ACM Trans. Graph.* 30, 4, Article 105, 10 pages.

[4] Fanny Chevalier, Pierre Dragicevic, Anastasia Bezerianos, and Jean-Daniel Fekete. 2010. Using text animated transitions to support navigation in document histories. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. 683–692.

[5] Laurent Denoue, Scott Carter, Jennifer Marlow, and Matthew Cooper. 2017. DocHandles: Linking Document Fragments in Messaging Apps. In *Proceedings of the ACM Symposium on Document Engineering*. 81–84.

[6] DiffChecker. 2018. https://www.diffchecker.com/. (2018).

[7] Steven M. Drucker, Georg Petschnigg, and Maneesh Agrawala. 2006. Comparing and managing multiple versions of slide presentations. In *Proceedings of the ACM symposium on User Interface Software and Technology*. 47–56.

[8] Björn A. Grüning, Eric Rasche, Boris Rebolledo-Jaramillo, Carl Eberhard, Torsten Houwaart, John Chilton, Nate Coraor, Rolf Backofen, James Taylor, and Anton Nekrutenko. 2017. Jupyter and Galaxy: Easing entry barriers into complex data analyses for biomedical researchers. *PLoS computational biology* 13, 5 (2017), e1005425.

[9] Manuele Kirsch-Pinheiro, Marlène Villanova-Oliver, Jérôme Gensel, and Hervé Martin. 2005. Context-aware filtering for collaborative web systems: adapting the awareness information to the user's context. In *Proceedings of the ACM Symposium on Applied computing*. 1668–1673.

[10] Bongshin Lee, Rubaiat Habib Kazi, and Greg Smith. 2013. Sketchstory: Telling more engaging stories with data through freeform sketching. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2416–2425.

[11] Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, Vol. 10. 707–710.

[12] Frank Thomas, Ollie Johnston, and Walton Rawls. 1981. *Disney animation: The illusion of life.* Abbeville Press, New York.

[13] TruScribe. 2018. TruScribe. http://www.truscribe.com/. (2018).

[14] VideoScribe. 2014. VideoScribe. https://www.videoscribe.co. (2014).

[15] Allison Woodruff, Ruth Rosenholtz, Julie B Morrison, Andrew Faulring, and Peter Pirolli. 2002. A comparison of the use of text summaries, plain thumbnails, and enhanced thumbnails for Web search tasks. *Journal of the Association for Information Science and Technology* 53, 2 (2002), 172–185.