Chameleon: Bringing Interactivity to Static Digital Documents

Damien Masson, Sylvain Malacria, Edward Lank, Géry Casiez

Chameleon: Bringing Interactivity to Static Digital Documents

Damien Masson[1,2], **Sylvain Malacria**[2,3], **Edward Lank**[1,2] **and Géry Casiez**[3,2,4,1]
[1]Cheriton School of Computer Science, University of Waterloo, Canada, [2]Inria, France,
[3]Univ. Lille, UMR 9189 - CRIStAL, France, [4]Institut Universitaire de France (IUF)
{dmasson, lank}@uwaterloo.ca, sylvain.malacria@inria.fr, gery.casiez@univ-lille.fr

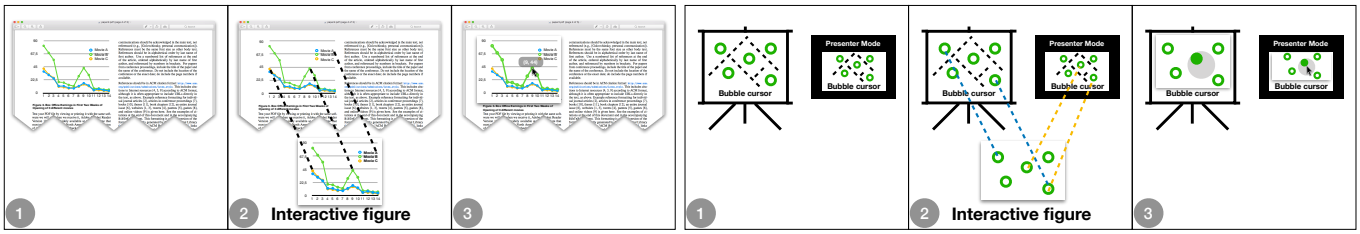

Figure 1. (1) Static documents such as PDF (left) and presentations (right) can be augmented with Chameleon by (2) locating corresponding interactive figures in the Chameleon database and (3) overlaying them on top of the existing documents.

ABSTRACT

Documents such as presentations, instruction manuals, and research papers are disseminated using various file formats, many of which barely support the incorporation of interactive content. To address this lack of interactivity, we present Chameleon, a system-wide tool that combines computer vision algorithms used for image identification with an open database format to allow for the layering of dynamic content. Using Chameleon, static documents can be easily upgraded by layering user-generated interactive content on top of static images, all while preserving the original static document format and without modifying existing applications. We describe the development of Chameleon, including the design and evaluation of vision-based image replacement algorithms, the new document-creation pipeline as well as a user study evaluating Chameleon.

Author Keywords

Augmented documents; feature matching; interactivity

CCS Concepts

•**Human-centered computing** → **Interactive systems and tools;** •**Applied computing** → *Multi / mixed media creation;*

INTRODUCTION

The meaning of the word *document* denotes a textual record [9], originally considered to be a paper-based artifact. However, the word *document* is now often used to refer to digital files alongside paper-based artifacts, and, while both paper-based and digital documents are still textual records, by nature a digital file can encapsulate multimedia information and interactive features alongside text and static images.

Despite the opportunity for dynamic and interactive content, many documents available on the internet have been produced using various file formats (*e.g.* docx, pdf, ppt). The applications used to generate these document types provide only limited support for incorporating interactivity into disseminated versions of the documents they produce.

There are, however, many potential benefits to interactivity. In his UIST 2014 closing keynote abstract, Victor [50] notes that

> *'Human beings naturally have many powerful modes of thinking and understanding. Most are incompatible with static media. In a culture that has contorted itself around the limitations of marks on paper, these modes are undeveloped, unrecognized, or scorned.'*

Even a task as simple as describing an interaction technique or interactive scenario requires careful wording when presented in static form, often requiring a *comic strip* figure describing the interaction in several steps, rather than letting the reader *experience* it (see Figure 1). Similarly, reporting scientific results often requires an author to make difficult decisions toward which data, and in which form, to report in a paper or while giving a research talk, rather than interacting with the data or allowing a reader to interact with alternative visualiza-

tions or analyses of the data. Even writing a simple how-to document to perform an everyday task can be complicated if all information is static. Some companies have attempted to make limited interactivity available in their formats. For example, PDF files can embed multimedia and interactive content [21, 24], but integrating interactivity into a PDF file is far from easy [24] and leverages Adobe Flash which is being discontinued [2]. Similarly, PowerPoint users can embed videos in their presentations or design staged animation that can be used as a mean to illustrate a dynamic situation. However, features are usually limited to specific file formats and the result can vary depending on the software version used to display the presentation. Moreover, limited support is provided for interactive figures.

Since interactive figures cannot be embedded directly into the document, authors tend to build dedicated external web pages linked in the document [41]. However, the extra work required from readers makes them less likely to engage with the supplementary material. Moreover, having multiple views significantly impacts cognitive overhead mainly due to the effort required for *comparison* and *context switching* as well as display space requirements [51]. Similarly, interrupting the *flow* of a presentation in order to manipulate an interactive figure using another application harms the *immersive*, *engaging* and *effectivness* aspect of the story being presented [12].

One solution to enable interactive content in documents would be to replace the associated document formats with a novel, more flexible, file format that would more easily enable interactivity. For example, Dragicevic et al. [16] created a web-based tool in order to demonstrate the exploration of multiverse explanations of statistical analyses in research papers. However, existing file formats remain firmly anchored in users' habits and replacing them would require significant changes, both in software and infrastructure. Furthermore, ideally interactivity would not be limited to new documents. Archived documents could, in many cases, also benefit from interactivity, but the process of fully transcoding and then adding interactive elements would be prohibitive in most cases.

What would be more ideal would be to preserve the existing formats and workflow, and, instead, to provide tools to augment digital versions of the files such that both new and legacy documents can become *reactive documents* [50]. Expanding on this point, support for the creation, dissemination, and viewing of reactive documents [50] should allow rich and flexible access to interactive features, preserve existing document formatting for visual correspondence with paper versions, leverage existing file formats, and allow users to continue to use their current document viewing tools rather than forcing them to adopt different software applications.

In this paper we introduce Chameleon (Figure 1), a tool that combines computer vision feature matching algorithms with an open database format to allow the incorporation and overlaying of dynamic content over any type of document (*e.g.* PDF files, Microsoft Word and PowerPoint documents, eBooks, etc.) without modifying existing applications or the source document. Using Chameleon, these documents can be easily upgraded with user-generated HTML5 interactive content. Chameleon thus allows the simplified provision and viewing of an enhanced version of a research paper with embedded interactive demonstrations and videos. It can also be used to perform live demonstrations of interaction techniques while giving a presentation without having to switch tools.

Chameleon accomplishes this via two interfaces: a background service and an editing tool. The background service runs on the user's system and analyzes raw, pixel-based on-screen content to identify static content for which interactive replacement content exists in its database. It then replaces this static content with interactive content on the visual display, seamlessly allowing a user to interact with the interactive version of the content. Scale and placement is preserved within the existing document, allowing easy visual correspondence between static-based versions of the document and on-line interactive versions of the same document. Alongside a display system service, Chameleon incorporates an editing tool to allow users to dynamically select static content within a specific file, associate interactive content with that static content, and verify the correct rendering of their interactive content. Since Chameleon relies on computer vision to analyze display content, identify static content with interactive enhancements, and seamlessly overlay that content completely automatically and transparently to the user, it guarantees its compatibility with any type of document (past, present or future) and any application a user currently uses to view that document, because it relies on what any document viewer ultimately does: displaying the document as pixels on-screen.

Our work makes the following contributions: 1) It introduces the notion of dynamic layering for augmenting static digital documents. 2) It presents the design and implementation of Chameleon, a document augmentation tool that implements this notion of dynamic layering on the macOS and Linux operating systems. 3) It reports on the results of a study comparing feature matching algorithms on non-natural images and for different scale levels. 4) It reports on the results of an experiment evaluating the performance of Chameleon for augmenting a corpus of documents, in this case a set of research papers and slideshows. 5) It details the results of a first-use study evaluating our design choices and the usefulness of Chameleon.

RELATED WORK

We first detail previous attempts to make static documents more dynamic before reviewing vision-based techniques previously used in the contexts of GUIs and documents.

Making Static Documents Dynamic

In his work on *Explorable Explanations* [49], Victor highlights three ways that reading environments can encourage greater inquiry via interactivity: reactive documents, explorable examples, and contextual information. Each of these allow readers to develop enhanced understanding either by manipulating data to see how model outputs change, by interacting with content widgets to experience behaviours, or by interacting with content to obtain additional contextual information on concepts in the document.

While Victor's analysis of current state-of-the-art indicates that much work remains to be done in interactive documents,

it is also true that many researchers have tried to create interactive experiences around documents. In terms of paper-based interaction with documents, the PADD system architecture [25] and the related PapierCraft system [33] which leveraged Anoto's digital paper support the remixing of printed information in various ways. If, instead, the documents are scanned into a computer, the ScanScribe system looks specifically at rough documents, essentially hand drawn sketches, and ways to structure and organize this material for post-hoc digital manipulations [46]. Finally, if information is captured in both paper and digital formats, systems such as ButterflyNet seek to combine myriad data sources including hand-drawn text, images, and other digital information into a cohesive single document [54]. However, the output of these systems remains little more than static content that has been remixed in various ways to produce a new, albeit static, document.

In the digital domain, recent research explored how to incorporate dynamic media into documents. For example, the Webstrates system [30] leverages a custom web server that provides shareable dynamic media to support flexible document creation, essentially a form of *transclusion* [40]. Alongside Webstrates, Wincuts [48] also allows users to carve out regions within windows, reposition those subregions, and then continue to interact with information in the subregion. However, the primary goal of Wincuts was to optimize screen real estate during, for example, document creation, rather than to replace static content with interactive content for explorable explanations. In ways similar to Wincuts, tools like d.mix [27] or Clip, Connect, Clone [20] mix information from multiple web pages to create mashups: web pages juxtaposing elements from others [11]. In contrast, Chameleon transcludes HTML content using dynamic layering onto figures of static documents.

Other options in the digital domain that can support Victor's explorable explanations include HTML documents, which can include interactive scripts, or a new document format such as ePub, which can be redesigned to incorporate interactive artifacts [53]. Consider, first, HTML documents; one reason that PDF persists as a format for document distribution is that PDF supports the flexible dissemination of documents by combining both content and visual formatting in a single, compact data format. The goal of presenting information in written form is the design of an artifact – a formatted document – that supports the ergonomics of reading. HTML rarely achieves this simplicity of presentation: new devices and browsers frequently render aspects of a document idiosyncratically, which can make reading an awkward experience, and even mobile versus desktop versions of documents often require bespoke aspects to ensure proper presentation in their respective formats. However, even if we advocate moving to a new format (either HTML or some novel new data format), in both cases this means that our extensive archive of existing documents must either be transcribed and re-imagined with interactive content, or must be maintained in original non-interactive format.

Even writing a simple how-to document to perform an everyday task can be complicated if all information is static. Some companies have attempted to make limited interactivity available in their formats. For example, PDF files can embed multimedia and interactive content [21, 24] through the inclusion of SWF files. However, the interactivity supported by PDF files still suffers from several limitations. First, integrating SWF into a PDF file is far from easy [24]. Second, SWF files are typically produced using Adobe Flash (even though they can also be produced in other ways using third-party software), which is being discontinued [2], so Adobe, itself, encourages designers to build interactive content using Web standards such as HTML 5 [1]. Third, this interactivity is barely supported by PDF viewers (basically, only Adobe Acrobat Reader will run the SWF file). Finally, the SWF file has to be embedded into the original document, which means that a static or legacy document cannot be converted into an interactive one without producing a new PDF file.

There has been significant efforts in circumventing PDF limitations by creating new kinds of documents viewers. For example, Utopia Documents [6] is a novel PDF viewer integrating visualization and data-analysis tools. By inspecting the content and structure of PDF files, Utopia Documents is able to integrate dynamic objects and annotations in the documents. In the same vein, Elastic Documents [7] generates visualizations from extracted tables in PDF documents and then display the relevant ones depending on the reader's focus. However, these viewers do not allow the transclusion of interactive content and are limited to the PDF file format.

We believe that the optimal approach to supporting the building blocks of explorable explanations is to incorporate reactive documents, explorable examples, and contextual information within the confines of existing static documents. Essentially, the goal of Chameleon is to support the overlaying of interactive features within the pre-existing presentation constraints of current documents. We do this by selectively replacing static content with dynamic content without manipulating the underlying document format and without any modification of existing document viewing applications.

Using vision-based techniques with GUIs and documents

Our concept is based on the identification of figures by analysing pixels on screen. This analysis needs to be robust against scale variations considering documents can also be displayed at different zoom levels and on different screen resolutions. These requirements make techniques based on template matching not suited for the task [55].

There exists a body of work in overlaying content on images within documents. For example, Kong et al. introduced Graphical Overlays[31], in which information is extracted from chart bitmaps in order to overlay graphical aids, such as gridlines on top of the charts. However, the user first has to capture a bitmap version of a chart and then upload it to a website to get the augmentation. In the same vein, Lu et al. [35] augment existing web-based visualizations by providing a suite of interactions such as filtering, comparison and annotation to help users visualizing information differently. Chameleon differs from the above systems in that it enables the support of any kind of augmentation without being limited to specific figures or specific interactions. In addition, the augmentation is co-localised with the document the user is reading rather than being rendered in another window.

A series of systems exist that seek to analyze GUI elements dynamically in an interface [14, 56]. First, the domain of these systems – identifying GUI elements for augmentation – differs from the domain of explorable explanations, in which we wish to augment document elements. Further, the approach used by these systems cannot be generalized to the domain of document annotation. Specifically, for example, Dixon et al.'s Prefab system reverse engineers an interface by analysing pixels on screen [14] using invariant features. The use of invariant features is possible when analyzing GUI elements because these are auto-generated during execution. However, invariant features cannot be applied in the case of document images: when a document is resized, all pixels of the figure are affected and there are no invariant features *a priori*. Another interesting approach was proposed in Yeh et al's Sikuli [55]. They applied classical computer vision techniques to identify an individual GUI element on the screen. The GUI element could then be used in search (e.g. to learn how to use the GUI widget), or dropped into scripts (to automate interface execution). To identify elements, Sikuli uses the SIFT [34] feature descriptor to extract features from elliptical patches detected by the MSER detector [36]. They then associate each of these features with features extracted from the screen. Finally, using a voting system and checking if a transformation can be computed from the associated features, they precisely locate the image on the screen. Using this method along with OCR and template matching, they report a success rate of 70.5% when performing search on a UI element, and they report less than 200 ms for identifying a single target GUI element on the screen. Given their focus on GUI elements, it is unclear whether a similar approach can work for arbitrary analysis of images in documents.

In the context of detecting plagiarism, methods have also been proposed to detect if two images are similar. Meuschke et al. [37] combined Perceptual Hashing with other algorithms to detect similar figures. However, since figures need to be extracted from the document first, these methods cannot be applied to localize figures. Similar to Sikuli [55], Iwanowski et al. [29] used Feature Matching, but their evaluation was focused on a small corpus of photographs and they acknowledge that more work is needed to confirm that it generalizes to other kind of figures (charts, drawings, screenshots).

Therefore, we evaluate SIFT [34] and MSER[36] alongside other image detectors and descriptors during the design of Chameleon. Furthermore, we explore how best to augment documents (versus GUI elements [14, 55, 56]) and present Chameleon as an end-to-end system to support both creating and interacting with explorable explanations.

CHAMELEON

The core idea of Chameleon is to augment pre-registered figures of digital documents by layering HTML5 resources over them (Figure 1). It uses feature matching algorithms to identify the pre-registered figures, and overlays on them an undecorated WebView that runs the corresponding HTML5 resource. Chameleon has been developed in C++ using the Qt 5.12 framework and OpenCV 4.0.1, together with platform dependent APIs. OpenCV provides feature matching algorithms while Qt provides the WebKit engine to render and interact with the HTML resources. Chameleon currently runs on macOS and Linux using respectively Cocoa and X11 APIs[1] and could also be implemented for Windows by replacing the platform dependent API calls.

Chameleon comprises two main parts: a REGISTRATION TOOL to register a figure from a document with its augmented version and an AUGMENTATION TOOL to analyse opened documents and *augment* them if *augment-able* figures are found.

The registration tool

The registration tool provides the interface to select a region of a document that will be registered in the database. To accomplish this, the user first opens a document that contains the target figure and navigates to its location. Using the *Register* command in the Chameleon menu, she opens a screenshot of the current viewport in a new window in which she highlights the region she wants to augment (typically the portion where the target figure is displayed) by dragging the mouse. The interface also provides a text field to enter the URL of the augmented resource. After validation, the augmentation is stored in a database. In order to compress the size of the database and for privacy reasons, we do not store the actual image, but only the pre-computed regions and descriptors of this image as provided by feature matching algorithms (see following sections for details) along with the document file size, its MD5, the dimensions of the selected region, and the url of the augmented resource. One potential downside of this approach is that, if an augmented document is modified, its MD5 changes. To address this, the registration tool also tracks modifications to augmented documents using DTrace [52], and, when an augmented document is modified, the user is prompted to either ignore or register the new, edited version of the document.

The augmentation tool

The augmentation tool runs in background and loops on the following six steps:

1. The *analysing opened files* step monitors the opened documents and determines those that contain figures that should be augmented (*i.e.* with entries in the database). This step allows Chameleon to drastically reduce the number of figures to search for from the database.
2. The *capturing windows* step takes screenshots of windows for files with at least one entry in the database (that is, at least one figure that should be augmented). The list of applications monitored can further be specified by the user.
3. The *locating figures* step analyses these screenshots to localise figures that should be augmented.
4. The *augmenting figures* step augments the located static figures using the associated HTML5 resources.
5. The *tracking figures* step detects changes in the position and size of the static figures to overlay the augmented versions; thus Chameleon does not need to repeat the whole analysis process for a figure that is already augmented.

[1] Source code available at ns.inria.fr/loki/chameleon

6. The *figure synchronization* step redirects mouse and key events from identical augmented figures. This step can be very useful for presentation tools (i.e. PowerPoint) to synchronize the presenter and slideshow views. In this way the presenter can interact with the presenter view and the results are directly reflected in the slideshow.

Step 1: analysing opened files
The augmentation tool monitors file access on the system using DTrace [52] on Linux and macOS. It retrieves the size of each opened file and queries the database to get the list of files with augmented figures of the same file sizes. Finally it computes the MD5 for the remaining files to filter the remaining documents. We avoid computing the MD5 for all opened files as it can be relatively long for large files.

Step 2: capturing windows
The augmentation tool takes a screenshot of every window containing documents to be augmented, using the XGETIMAGE [13] function on Linux, and CGWINDOWLISTCREATEIMAGE [5] on macOS. These functions provide the content of a window even if partially hidden, without decorations.

Step 3: locating figures
Chameleon uses feature matching algorithms to determine regions and descriptors in the screenshots and detect potential figures to be augmented. This step takes the longest time in the augmentation process. We therefore re-compute regions and descriptors only when the window is scrolled or resized. The feature matching algorithm then tries to associate the regions found with the ones stored in the database to detect the figures present and determine their location and size.

Step 4: overlaying interactive figures
For each figure found, the augmentation tool opens an undecorated window with transparent background. These windows cannot be focused, resized or moved. Their size and position are dynamically updated to be the same as the figures to augment. Each window contains a QWEBENGINEVIEW [43] used to load the remote HTML content located at the URL associated with the figure to augment. Users can interact with augmented figures using the mouse cursor. In addition, they can hide and show augmented figures by using a button located at the top left corner useful to review the original. Users can also get additional information about the augmented figure in a drop down menu and switch between different augmented figures available in the database (in the case where multiple augmented figures exist for an individual figure in the source document). Augmented figures are pre-loaded in background when opened files containing augmented figures are found in step 1. The window is first fully invisible and made visible when a figure is found in step 3. We use a 500ms delay and a 1000ms fade-in effect, tuned via informal pilot testing, to inform the user that a figure is augmented and to let her verify that the augmented figure matches the static one.

Step 5: tracking figures
Linux provides an API called AT-SPI (Assistive Technology Service Provider Interface) [22] that can send a notification when a window is scrolled or resized. macOS provides the same service through its Accessibility API [4]. Window translation is tracked by pooling their position when a mouse drag event is detected. Each window displaying an augmented figure registers callbacks to be notified when one of the properties of the window displaying the document changes. These callbacks are called with a small latency making it possible to re-position and re-size the augmented figures with hard-to-notice visual artefacts. When a figure is cropped (typically during or after scrolling), we apply a mask on the augmented window so that it is cropped in the same way and does not extend outside the document window. This step is not mandatory for using Chameleon, as overlaying interactive figures over a static document can be achieved without it, but it results in a smoother integration of the augmented figures.

Step 6: figure synchronization
When Chameleon detects multiple identical figures on the screen, the overlaid augmented versions can be synchronized. The augmented figure with the highest resolution is tagged as the main figure; others are considered copies and resized to match their overlaid static figure resolutions. All the events received by the copies, e.g. keyboard and mouse events, are redirected onto the main figure. This step, enabled by default, can be disabled via Chameleon's menu.

Authoring augmented figures
Chameleon augments documents using HTML5 resources, easing sharing in a community of users. Authoring interactive content is a matter of creating and publishing on-line the HTML5 resources to be used for augmentation, with the only requirement of an aspect ratio similar between the HTML5 resource and the figure. Existing tools can be leveraged depending on the type of interactive figures to be overlaid in a document. For example, a video can be embedded by first uploading it on a video sharing website such as *YouTube* and then using the embeddable URL with Chameleon. From an existing R script, outputting an interactive chart in lieu of a static one can be done without any change to the code using plotly for R[42]. Finally, when the interactive figure is already available online, we provide a JavasScript function capable of isolating a specific element of a webpage (i.e. hiding everything except the element) resulting in a webpage, containing only the figure, that can then be used in Chameleon.

In the following three sections, we present three user studies. The first two user studies drive the design of Chameleon. The first examines the performance of ten different feature matching approaches to determine which works best for the specific use-case of Chameleon. Next, a second experiment evaluates the two best performing algorithms to determine their ability to precisely overlay augmentations onto pre-existing document figures by calculating the average overlap precision between a figure and its augmentation. Finally, we explore the usability of Chameleon via a summative, first-use study [54].

FEATURE MATCHING ALGORITHM COMPARISON
Chameleon requires feature matching to accurately locate figures displayed on screen. Many feature matching algorithms have been proposed, but it remains unclear which ones are the most promising for scale-independent location of figures in

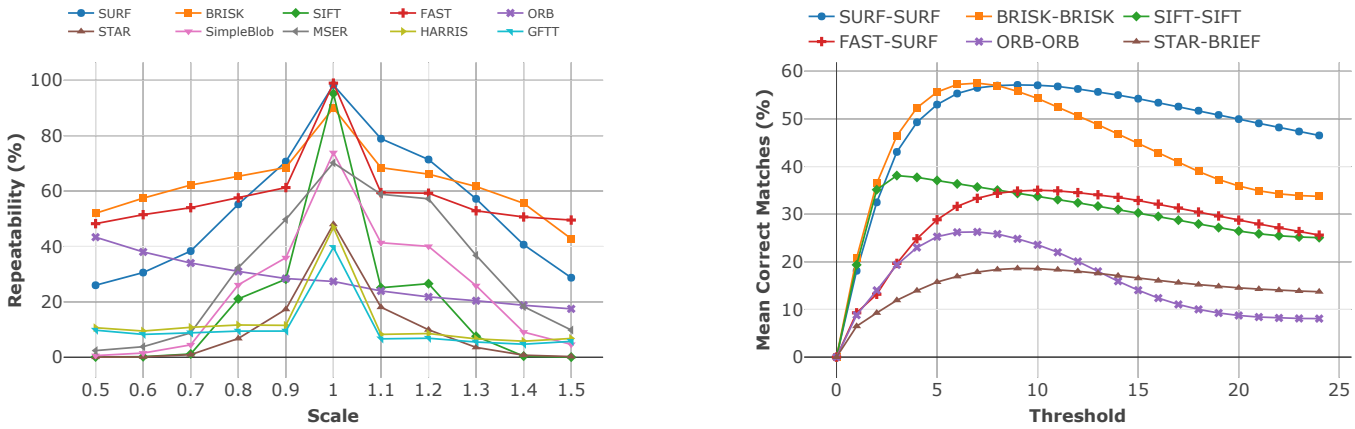

Figure 2. ***Repeatability*** **vs scales, and** ***Mean Correct Matches*** **vs thresholds of our dataset.**

static documents. In this section, we compare the performance of various feature matching algorithms against these needs.

Feature matching between two images A and B can be described as a 5-step process:

1. For each image, a first algorithm called a *Detector* is applied to extract a certain number of key regions (described by an x;y coordinate and a diameter).
2. Then, a second algorithm called a *Descriptor* takes the regions identified by a given detector and returns a *descriptive vector* of numbers describing each of these regions.
3. Next, either via brute force or a dedicated algorithm, the system measures a *distance* between the *descriptive vector* of each key region of image A and B (using Hamming or Euclidean distances), and returns a *list of matches* between both images (tuple, one for each image).
4. After that, only the *matches* with a *distance* below a specific *threshold* are kept in order to eliminate most incorrect matches. This results in a *list of associated regions*.
5. Finally, the system takes the *list of associated regions*, as well as the x;y coordinates of these key regions to identify an appropriate homographic transformation using the RANSAC algorithm [18].

Comparing feature matching algorithms

Feature matching algorithms are usually compared with a set of natural images using the following metrics:

Repeatability: Evaluates the performance of *detectors* (step 1 above) by measuring the proportion of similar key regions extracted from two images containing the same object [39].

Precision and recall: Evaluates the performance of a *detector-descriptor* pair (step 2 and 3 above) by measuring the number of key regions from a first image that are correctly associated to key regions from a second image [38].

However, already published comparisons of feature matching algorithms suffer from several limitations making them unhelpful to find the most adapted algorithm for Chameleon's needs. First, algorithms are tested on natural images and photos, while documents contain a wider variety of figures (e.g. state diagrams, photos, bar charts, and other data visualizations). Second, robustness to scale is seldom measured during evaluations of algorithms, whereas it is the main deformation that Chameleon faces since documents are displayed at different scales. Third, precision and recall are less adapted to compare detector-descriptor pairs when different detectors are tested: some detectors yield no key regions for harder-to-detect objects. Thus, descriptors associated with those detectors would be given an unfair advantage because they would not be tested on figures with harder-to-detect features. To address this issue, we use (instead of the precision and recall) a *Mean Correct Matches* metric that computes the mean percentage of correctly associated regions per image. This metric takes into account detectors yielding no key regions and decreases the overall percentage of correct matches accordingly.

Dataset

We evaluate the different feature matching algorithms on two datasets. The first is a *scientific papers* dataset introduced by Clark et al. [10], composed of 150 research articles from 3 different conferences (NIPS, ICML and AAAI) from 2008 to 2014. All figures in this document set were annotated manually. In order to add diversity to the figures, we gathered a second *presentation* dataset. 100 presentations were randomly selected in 10 different categories from the SpeakerDeck website [17]. We extracted from the first 20 pages of each presentation all images whose height and width were larger than 50 pixels. In total, the dataset comprises 1660 figures from 2741 pages. Each of these figures was matched against the PDF page containing the figure, rasterized at 72 DPI. To evaluate the influence of scaling on the results of feature matching algorithms, we applied a scale transformation to the rasterized PDF pages. Tested scales were comprised of every 0.1 scaling step in [0.5, 1.5], and the resizing was based on a bilinear interpolation, the technique observed in Adobe Acrobat Reader DC version 2018 on macOS Sierra running OS X version 10.12.6. Knowing the position and the size of the figure in the scene, we compute as ground truth the homography relating a figure to its PDF page.

Results

To reduce the pairs of algorithms tested, we first evaluate detectors, and then match descriptors with the best detectors.

Detector evaluation

We choose to evaluate all the detectors implemented by default in OpenCV 4.0.1. Those detectors include the most commonly used (BRISK [32], FAST [44], GFTT [47], Harris [26], MSER [19], SIFT [34], SURF [8]) as well as more recent ones (ORB [45], SimpleBlob, STAR [3]).

Figure 2 shows the mean repeatability score over our two datasets, by scale level, for each detector. Unsurprisingly, almost all detectors perform best at a scale level of 1. The only exception is ORB which, while performing relatively poorly for all scale levels, performs better for smaller scale levels, explained by the fact that the key regions produced by ORB are larger on average than for the other detectors and that the repeatability metric relies on the overlapping of regions. Therefore, when the scale of image B is reduced, the overlapping ratio is likely to increase [39].

Overall, FAST, SURF and BRISK achieved the best results across all scale levels. Their repeatability score is close to 100% (respectively 99.8%, 99% and 95%) at a scale of 1.0, and, except for SURF, is above 50% for all other scale levels. Other algorithms either performed very poorly or are not robust to scale variation, which is a critical criteria for Chameleon. SIFT used by Sikuli [55] performs well for a scale of 1.0 but its performance quickly drops at other scale levels.

Descriptor evaluation

Descriptors need key regions to be extracted by a detector; therefore we associate them with their corresponding detectors, as described by their authors. FAST was also included in the evaluation even though it is only a detector because of its high repeatability. In order to determine the descriptor algorithm to use with FAST, we piloted its association with every descriptor algorithm implemented in OpenCV 4.0.1, using a random subset of our image data. These pilot tests suggested that the descriptor SURF worked best. Thus, we compared BRISK-BRISK, FAST-SURF, ORB-ORB, SIFT-SIFT, STAR-BRIEF and SURF-SURF, with their default parameters.

Figure 2-right shows the evolution of *Mean Correct Matches* for different *thresholds*. We associated a unique number to each treshold tested in order to show the results in the same chart. All algorithms were tested using 25 different thresholds, in the range [0-175] for BRISK-BRISK, [0-0.25] for FAST-SURF, [0-100] for ORB-ORB, [0-425] for SIFT-SIFT, [0-75] for STAR-BRIEF and [0-0.25] for SURF-SURF.

Overall, we observe that SURF-SURF and BRISK-BRISK outperform the other algorithms, both of them reaching a *Mean Correct Matches* of more than 55% while the other algorithms do not exceed 40%. These results have several implications. First, algorithms that, reportedly, outperform SURF or SIFT perform more poorly on our dataset (e.g. ORB and STAR). Second, algorithms used in the literature for similar motivations (i.e. figures and GUI elements) did not offer the best performance (typically SIFT, used in [55], which was outperformed by both SURF and BRISK). Finally, these results help drive design, including an exploration of options for implementation. For example, because SURF is patented, one could decide to use BRISK to obtain similar results.

FEASIBILITY EVALUATION OF CHAMELEON

The comparison of feature matching algorithms described in the previous section suggests that BRISK-BRISK and SURF-SURF would be the best candidates to perform online feature matching in the context of Chameleon. However, this comparison does not reflect Chameleon's performance for dynamically augmenting digital documents. Therefore, we conducted an evaluation to measure the number of figures correctly identified in a document by these algorithms.

Method

We used the same dataset as in the previous experiment (totalling 1660 figures). While using a computer, documents can be displayed at different scale levels; therefore, we tested for each *page* (presentation slide or PDF page) the 12 following *scale* factors: 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 2.0, 4.0. For each image pair, image A being a figure and image B a scaled page, we computed the correspondences by using either BRISK-BRISK or SURF-SURF (referred as BRISK and SURF for simplicity in the next paragraphs). A figure was considered found when the rectangle obtained from the correspondences had positive width and height and its aspect ratio was similar to the figure identified (*i.e.* $abs(1 - aspectRatioA/aspectRatioB) <= 0.1$)). Based on this identification, we classified every outcome of the algorithm into one of these categories :

- True positive (TP): the figure is present and found
- True negative (TN): the figure is not present and not found
- False positive (FP): the figure is not present but found
- False negative (FN): the figure is present but not found

For TPs, we also computed the overlap ratio of the rectangle found with the real rectangle of the figure in the page.

Results

Because feature-matching performance is likely to be impacted by the *surface area (surface)* of the figure in pixels (height$\times$width$\times$scale), we report the results by categorizing the figures in 5 groups depending of their surface: $surface \geq 0$, $surface \geq 2500$, $surface \geq 5000$, $surface \geq 7500$, $surface \geq 10000$ pixels. 10000 pixels thus corresponds to an image of 100×100 pixels at scale 1.0.

Accuracy

As seen previously, feature-matching algorithms match key-regions from different images if the euclidian-distance between their respective descriptive vectors is below a certain threshold. Choosing a value for this threshold is the first step towards using feature-matching algorithms to identify figures displayed in documents. Figure 3 illustrates the proportion of TP and FP (dashed lines) of BRISK and SURF depending on this distance threshold and figure sizes. As we can see, the proportion of TP rapidly increases for both algorithms to reach a cap value. This cap value seems to be reached at a similar threshold value, regardless of the size of the figure, suggesting

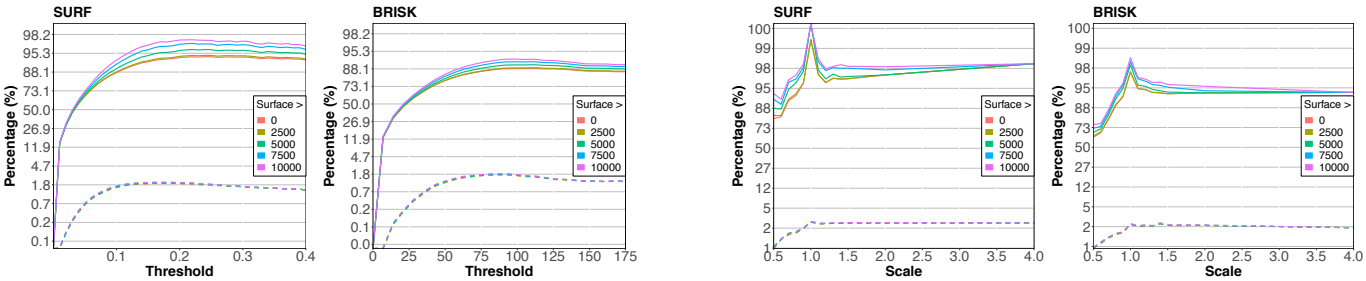

Figure 3. Evolution the true positive rate and false positive rate (dashed lines) depending on the feature-matching algorithm threshold and scale at which a figure is displayed. The y axis uses a logit scale.

that we can choose a single threshold for all figures. Therefore, the threshold yielding a high TP rate of while keeping a low FP rate, was 112 and 0.22 respectively for BRISK and SURF.

The two rightmost graphs in Figure 3 respectively show the proportion of true and false positives for BRISK and SURF using these thresholds, for each scale factor and depending on the size of the figure. Overall, the proportion of TP (respectively FP) was of 88.4% (1.5%) for BRISK and 94.4% (1.8%) for SURF over the whole dataset. Unsurprisingly, we observe a peak of performance at scale 1.0 for both algorithms, regardless of figure surface, which can easily be explained by the fact that this is the optimal testing condition in which the figure contained in image B (the page of the tested document) has the exact same size as image A (the figure we are looking for). More interestingly, we observe that the proportion of TP barely decreases for scales higher than 1.0 (98.6% TP for SURF and 94.1% for BRISK above scale 1.0), whereas it decreases more significantly for scales lower than 1.0. This can be explained by the fact that scaling down the size of the figure may make the task more difficult for detector algorithms, resulting in different key regions extracted. In addition, we observe that the lower the size of the figure, the more the proportion of TP decreases for smaller scales. In addition to reaching a lower peak of performance than SURF, we can see that BRISK is also more impacted by scale factors decreasing by 4 points between scale 1.0 and scale 4.0, and 32 between scale 1.0 and 0.5, whereas SURF decreases 2 and 18 points, respectively. Finally, the average overlap rate for true positives is over 98% for both algorithms, meaning that when the image is found, the position and size are usually correct.

We also measured the time taken by SURF and BRISK on average, for each document, to 1) extract key regions of the image candidate to augment using the detector, 2) compute their descriptive vectors, 3) match them with the database and 4) compute the homographic transformation if a figure that should be augmented was found. Without any optimisation (such as multi-threading or GPU), assuming the document is displayed in 1080p, the time to find a figure on a 3.2GHz Intel Core i5 is approximately (in ms): $t_{SURF} = 300 + 80 * x$ and $t_{BRISK} = 350 + 90 * x$ with 'x' as the number of augmented figures for a given document.

In the context of Chameleon, we use SURF over BRISK because it yields a higher proportion of TP, while keeping the proportion of FP below an acceptable threshold of 2%.

FIRST-USE STUDY

Our previous experiment demonstrates that Chameleon can accurately augment digital documents. In this section, we investigate whether **(Q1)** users can use Chameleon; **(Q2)** they find it faster and more convenient to use than existing solutions; and **(Q3)** they would use Chameleon in everyday life.

Procedure

For this study, we recruited 12 participants (23 to 42 age range, mean = 28, 6 identified as female and 6 identified as male), all graduate students in computer science as they are the most likely to have used, or at least know, alternative methods to augment digital documents. Participants were first shown an interactive demonstration of the bubble cursor [23] and the original paper presenting the technique. They were then asked "How would you include this interactive figure in the document?". Their answers were manually transcribed. The interviewer then introduced them to Chameleon and showed them how to use it through an example. Participants were then asked to use Chameleon in the three following scenarios described below. For each scenario, we measured the time to accomplish the task. Participants then completed a questionnaire and answered open-ended questions. Finally, we showed participants a document augmented using Chameleon and the same document without Chameleon but with an external web page containing all the interactive figures. Participants had to choose the one they preferred and explain why.

Scenario 1: Interactive Demonstration in Paper. The bubble cursor [23] is a pointing technique which reduces the distance to a target by increasing the activation area of targets. While the original paper includes *comic strips* of images to explain the technique, as Ben Schneiderman notes, *"An interface is worth a thousand pictures"* Hence, we implemented an interactive version of the bubble cursor using an HTML canvas and Javascript. We then asked participants to augment Figure 6 of the original bubble cursor paper with the interactive resource that we provided.

Scenario 2: Supporting Explorable Multiverse Analyses. Dragicevic et al. [16] proposed Explorable Multiverse Analyses Reports (EMAR) as a way to increase the transparency of research papers. However, they acknowledge that the main obstacle to their adoption is the lack of tools to support their creation and integration. Chameleon partially solves this problem by supporting interactivity in both archived and new research papers. For this scenario, we used the mini-paper *Prior* acces-

sible online [15] and created by Dragicevic et al. In this case, the interactive figure was already created and we wish to include it into a PDF document. We extracted the figure by using Javascript and hiding everything but the <*div*> containing the interactive figure. We also transcribed the mini-paper to PDF. We then asked participants to create an interactive document by overlaying the interactive figure in the PDF document.

Scenario 3: Adding Interactivity to Presentations. For this final scenario, we asked participants to augment a PowerPoint presentation with two slides: one containing a static image and a link to a *YouTube* video and the other a static image of a bar chart generated using plotly and R[42]. We gave participants the export link provided by YouTube, and a link to the HTML file as generated by plotly and R. We then asked participants to augment the presentation by including the video and the interactive version of the chart directly into the presentation.

Results

When asked how they would include interactive figures into a document, 6 participants answered that they would include a link to the interactive figure in the document, 9 participants proposed transcribing the document to HTML, 1 participant proposed converting the figure to Flash and including it in the PDF and, finally, another participant proposed creating a new document viewer supporting interactive figures.

Participants completed the scenarios without difficulty in 58s (SD=18s) on average for scenario 1, in 58s (SD=17s) for scenario 2 and 1m32s (SD=25s) for scenario 3 **(Q1)**.

Although they were not asked to perform a controlled (A vs B) study with the approaches they described, all 12 participants agreed that they would rather use Chameleon over their initial described approaches. However, 3 participants noted that Chameleon was not easier or faster than adding a link to the interactive figure in the document but agreed it gave a better result (*"I think the URL is easier, faster... but the result is better... definitely better in... using this tool"*). Similarly, 5 participants believed that transcribing the document to HTML might give a similar or possibly better result for the reader but that Chameleon was easier and faster to use **(Q2)**.

Using a 5-point Likert-type scale, participants also rated the integration of interactive figures in documents using Chameleon when scrolling (Mdn=4.5, SD=0.9), scaling (Mdn=4, SD=1.2), moving the window (Mdn=4, SD=1) and if the augmentation was sized to the static figure (Mdn=5, SD=0.4). Finally, all participants answered yes to the question "Would you be willing to put extra effort in order to create interactive figures if you could include them in your documents" **(Q3)**.

While some participants noted that having figures in another view allows scaling and viewing of the figure independent of the document, all participants preferred having the interactive figures within the document. One participant mentioned that *"You do not necessarily know where the figures [on the external view] are in the document... Instead [with Chameleon] you always have the figure close to the text referring to it."*.

DISCUSSION

The final distribution of Chameleon includes both the registration tool to create and the augmentation tool to interact with explorable explanations. As noted in the section describing the augmentation tool within Chameleon, by combining delay with fade-in, we allow users to perceive the original figure in the document and see the change to an augmented version of the figure helpful to identify rare cases of mismatch, and inform users of an augmentation. Alongside these effects, Chameleon supports various configuration options. For example, some users may wish to have the original figures viewable by default and to intentionally invoke augmentation when desired; for these users, Chameleon can be configured such that augmentations have to be activated explicitely for figures via the button mediator as no augmentation is displayed initially. Similarly, if in augmented mode, figures can be turned off via this same button, allowing users to switch between the original static versions and the augmented versions flexibly during document reading. Users also have the option to place augmented figures in floating windows (i.e. resizable and movable) if they wish to manipulate them independently from the document.

Chameleon relies on a feature matching algorithm carefully chosen through a systematic analysis of the detector and descriptor algorithms of the literature. Because these algorithms were never tested on figures commonly included in documents and on varied scale factors, we performed this analysis and evaluated both the accuracy and time of these algorithms on a real-world data set. Results show the very good performance of SURF over a wide range of scale levels. The lower percentage of TP below scale 1.0 is not critical as users are less likely to expect the augmentation of a small-sized figure given the limited interaction space it offers. SURF also allows for real time augmentation with processing times around 600 ms.

Chameleon in Practice

Augmentation Longevity. Because augmentations are not embedded in the documents, they suffer from the same issues as files hosted online. Augmentations could be lost if their host disappears or they are otherwise removed. This would result in a document without interactivity; the document would only display the original static figures (as if Chameleon was not enabled). It may be possible to mitigate this using peer-to-peer hypermedia protocols such as IPFS [28]. An augmentation lives as long as one of the node in the network has it stored.

Application Scalability. As additional users leverage Chameleon and as augmented figures become more commonplace, scalability concerns regarding the number of figures in a document, the number of figures on-screen, the number of augmented documents that exist in the world, and the number of simultaneous users may concern some readers. Chameleon deals with these issues as follows. First, because the cost of each additional figure in a document is relatively small (a fact we note when evaluating Chameleon in our second study), multi-figure documents are still handled in a reasonable time by Chameleon: for example, with 5 augmented figures in a document, it would take 700ms for them to appear the first time, (e.g. 10 = 1.1s; 20 = 1.9s). Similarly, multiple on-screen figures can be managed because, as we note earlier

during design, transformations do not have to be re-computed once figures are found; Chameleon can simply track position changes on the display. Finally, modern database systems can easily handle large numbers of documents and users. Documents are filtered using their MD5, data storage is limited to descriptive vectors not figures, and embedded resources are linked client-side. As well, modern database systems have evolved to handle large user loads.

Privacy. Chameleon works by taking screenshots of some windows. However, to preserve privacy, these screenshots are not publicly transmitted; instead, they are analyzed locally, descriptive vectors are extracted locally, and only these descriptive vectors are stored in the database and used to match figures with their augmentations. Chameleon can also be used to provide interactive content to more confidential documents via databases that support user or group security. In essence, privacy issues are limited as Chameleon only exchanges image feature vectors with a database, not entire images. This is not to say that there are no privacy considerations. If a user is augmenting a figure in a document, then the server delivering the augmentation will know which document is being read on a user's computer. However, any cloud based document system suffers from similar problems: as an example from the field of Human-Computer Interaction, we often leverage the ACM digital library during our research, meaning that this digital library is aware of topics being examined by researchers, and tools such as Overleaf have significantly more information on documents we create. If a user wishes to avoid this awareness, it is also possible to run Chameleon locally by downloading a local copy of the database (whose size should remain minimal considering the small amount of information recorded for each document). Finally, users can disable Chameleon if necessary for certain documents/applications if they would prefer complete privacy at the cost of losing augmentations to figures in the documents/applications.

Chameleon at the Community Level. The philosophy behind Chameleon, alongside augmenting documents for the user, is to support both canonic and community augmentations of a document. By default, the creator of the static document should be identified and have the ability to augment the document with canonical augmentations. For new documents, these canonical augmentations can be introduced at the time of document creation, supporting reactive behaviours, dynamic experimentation with parameters, interactive widgets to demonstrate input techniques, open-data collected via experimentation, and augmented information referencing follow-on research. Where possible, archived versions of PDF documents can also be easily augmented post-hoc by their creator, ensuring that even archived documents continue to evolve over time. Alongside creators, other members of the community are also able to augment a document with their own augmentations if they wish. We believe that users could benefit from noncanonical augmentations, in the form of a community augmentation (similar to pull requests on GitHub which let a user tell others about changes she has pushed to a repository, and discuss and review the potential changes with collaborators). In the end, the creator of the document should decide whether or not the community augmentation will systematically augment the document. Even if the creator is not available to create or promote to canonical augmentations, the community could still discuss, vet, and promote augmentations that allow the large archive of PDF documents to incorporate reactive features.

CONCLUSION AND FUTURE WORK

Inspired by Victor's work on the concept of explorable explanations, we present Chameleon, a system that leverages computer-vision based feature matching and a database of active content to graft interactivity onto static figures within a corpus of documents. This paper describes an evaluation of various feature matching algorithms to design the system. As well, we present the two tools within Chameleon, a registration tool to allow a user (e.g. the document creator) to graft the interactive figure onto the original document, and an augmentation tool that allows a reader to interact with the augmentation. Because Chameleon works on screen-based pixels, it can augment both new documents and pre-existing documents, thus allowing our pre-existing archive to benefit from interactivity in support of a better interactive experience when consulting static documents. Based on a systematic evaluation of descriptors and detectors, we identified SURF as the best algorithm given its overall high performance and robustness to scale adjustments.

Our implementation was designed to work with desktop applications, but Chameleon could also be integrated to mobile devices, assuming that the mobile OS allows a service to run in the background, analyze displayed content, and overlay HTML5 views on top of content. If the OS does not allow background services, Chameleon could relatively easily be integrated into open-source systems like Android. An implementation of Chameleon on mobile is left as future work.

The already good accuracy and performance of Chameleon can be optimized in several ways that we plan to explore as future work. One approach could be to directly integrate Chameleon in a window manager, providing direct access to the pixels of each window before being displayed. Augmented figures could be directly drawn in the pixel buffer of the window, making the augmentation flawless. The window manager being responsible for handling window movement and resizing, we would know exactly when those actions happen and apply them on the augmented figures. Finally, the registration tool of Chameleon could provide an indicator showing the percentage of similarity with figures already registered in the database for the document in order to notify the users that an augmentation is likely to result in a false positive.

ACKNOWLEDGMENTS

This research received ethics clearance from the Office of Research Ethics, University of Waterloo. This work was partially supported by the LAI Réapp and the National Research Council of Canada University of Waterloo Collaboration Centre. The two first experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (https://www.grid5000.fr).

REFERENCES

[1] Adobe. 2015. Flash, HTML5 and Open Web Standards. (2015). Retrieved April 7th, 2019 from https://theblog.adobe.com/flash-html5-and-open-web-standards/

[2] Adobe. 2017. Flash and The Future of Interactive Content. (2017). Retrieved April 7th, 2019 from https://theblog.adobe.com/adobe-flash-update/

[3] Motilal Agrawal, Kurt Konolige, and Morten Rufus Blas. 2008. CenSurE: Center Surround Extremas for Realtime Feature Detection and Matching. In *Computer Vision – ECCV 2008*, David Forsyth, Philip Torr, and Andrew Zisserman (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 102–115.

[4] Apple. 2007a. Accessibility on macOS. (2007). Retrieved April 7th, 2019 from https://developer.apple.com/accessibility/macos/

[5] Apple. 2007b. CGWindowListCreateImage. (2007). Retrieved April 7th, 2019 from https://developer.apple.com/documentation/coregraphics/1454852-cgwindowlistcreateimage

[6] T. K. Attwood, D. B. Kell, P. McDermott, J. Marsh, S. R. Pettifer, and D. Thorne. 2010. Utopia documents: linking scholarly literature with research data. *Bioinformatics* 26, 18 (2010), i568–i574. DOI: http://dx.doi.org/10.1093/bioinformatics/btq383

[7] S. K. Badam, Z. Liu, and N. Elmqvist. 2019. Elastic Documents: Coupling Text and Tables through Contextual Visualizations for Enhanced Document Reading. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (Jan 2019), 661–671. DOI: http://dx.doi.org/10.1109/TVCG.2018.2865119

[8] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. 2008. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.* 110, 3 (June 2008), 346–359. DOI: http://dx.doi.org/10.1016/j.cviu.2007.09.014

[9] Michael K. Buckland. 1998. What is a "document"? *Journal of the American Society for Information Science* 48, 9 (1998), 804–809. DOI: http://dx.doi.org/10.1002/(SICI)1097-4571(199709)48:9<804::AID-ASI5>3.0.CO;2-V

[10] Christopher Andreas Clark and Santosh Divvala. 2015. Looking Beyond Text: Extracting Figures, Tables and Captions from Computer Science Papers. In *AAAI Workshops*. AAAI, USA, 423–434. http://aaai.org/ocs/index.php/WS/AAAIW15/paper/view/10092

[11] Allen Cypher, Mira Dontcheva, Tessa Lau, and Jeffrey Nichols. 2010. *No Code Required: Giving Users Tools to Transform the Web*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[12] Andreas Dieberger, Cameron Miner, and Dulce Ponceleon. 2001. Supporting Narrative Flow in Presentation Software. In *CHI '01 Extended Abstracts on Human Factors in Computing Systems (CHI EA '01)*. ACM, New York, NY, USA, 137–138. DOI: http://dx.doi.org/10.1145/634067.634151

[13] die.net. 1987. XGetImage. (1987). Retrieved April 7th, 2019 from https://linux.die.net/man/3/xgetimage

[14] Morgan Dixon and James Fogarty. 2010. Prefab: Implementing Advanced Behaviors Using Pixel-based Reverse Engineering of Interface Structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 1525–1534. DOI: http://dx.doi.org/10.1145/1753326.1753554

[15] Pierre Dragicevic, Yvonne Jansen, Abhraneel Sarma, Matthew Kay, and Fanny Chevalier. 2018. Explorable Multiverse Analyses. (2018). Retrieved August 1st, 2019 from https://explorablemultiverse.github.io/

[16] Pierre Dragicevic, Yvonne Jansen, Abhraneel Sarma, Matthew Kay, and Fanny Chevalier. 2019. Increasing the Transparency of Research Papers with Explorable Multiverse Analyses. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, Article Paper 65, 15 pages. DOI: http://dx.doi.org/10.1145/3290605.3300295

[17] Fewer and Faster. 2010. Speaker Deck. (2010). Retrieved April 7th, 2019 from https://speakerdeck.com/

[18] Martin A. Fischler and Robert C. Bolles. 1987. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In *Readings in Computer Vision*, Martin Fischler and Oscar Firschein (Eds.). Morgan Kaufmann, San Francisco (CA), 726 – 740. DOI: http://dx.doi.org/10.1016/B978-0-08-051581-6.50070-2

[19] P. E. Forssen. 2007. Maximally Stable Colour Regions for Recognition and Matching. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, USA, 1–8. DOI: http://dx.doi.org/10.1109/CVPR.2007.383120

[20] Jun Fujima, Aran Lunzer, Kasper Hornbæk, and Yuzuru Tanaka. 2004. Clip, Connect, Clone: Combining Application Elements to Build Custom Interfaces for Information Access. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM, New York, NY, USA, 175–184. DOI: http://dx.doi.org/10.1145/1029632.1029664

[21] Emmanouil Giannisakis, Gilles Bailly, Sylvain Malacria, and Fanny Chevalier. 2017. IconHK: Using Toolbar Button Icons to Communicate Keyboard Shortcuts. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 4715–4726. DOI: http://dx.doi.org/10.1145/3025453.3025595

[22] GNOME. 2011. Assistive Technology Service Provider Interface. (2011). Retrieved April 7th, 2019 from https://github.com/GNOME/at-spi2-atk

[23] Tovi Grossman and Ravin Balakrishnan. 2005. The Bubble Cursor: Enhancing Target Acquisition by Dynamic Resizing of the Cursor's Activation Area. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05)*. ACM, New York, NY, USA, 281–290. DOI: http://dx.doi.org/10.1145/1054972.1055012

[24] Tovi Grossman, Fanny Chevalier, and Rubaiat Habib Kazi. 2015. Your Paper is Dead!: Bringing Life to Research Articles with Animated Figures. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '15)*. ACM, New York, NY, USA, 461–475. DOI: http://dx.doi.org/10.1145/2702613.2732501

[25] François Guimbretière. 2003. Paper Augmented Digital Documents. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (UIST '03)*. ACM, New York, NY, USA, 51–60. DOI: http://dx.doi.org/10.1145/964696.964702

[26] Chris Harris and Mike Stephens. 1988. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*. IEEE, USA, 147–151.

[27] Björn Hartmann, Leslie Wu, Kevin Collins, and Scott R. Klemmer. 2007. Programming by a Sample: Rapidly Creating Web Applications with D.Mix. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*. ACM, New York, NY, USA, 241–250. DOI: http://dx.doi.org/10.1145/1294211.1294254

[28] IPFS. 2015. IPFS is the Distributed Web. (2015). Retrieved June 24th, 2019 from https://ipfs.io

[29] Marcin Iwanowski, Arkadiusz Cacko, and Grzegorz Sarwas. 2016. Comparing Images for Document Plagiarism Detection. In *Computer Vision and Graphics*, Leszek J. Chmielewski, Amitava Datta, Ryszard Kozera, and Konrad Wojciechowski (Eds.). Springer International Publishing, Cham, 532–543.

[30] Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates: Shareable Dynamic Media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15)*. ACM, New York, NY, USA, 280–290. DOI: http://dx.doi.org/10.1145/2807442.2807446

[31] N. Kong and M. Agrawala. 2012. Graphical Overlays: Using Layered Elements to Aid Chart Reading. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (Dec 2012), 2631–2638. DOI: http://dx.doi.org/10.1109/TVCG.2012.229

[32] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. 2011. BRISK: Binary Robust Invariant Scalable Keypoints. In *Proceedings of the 2011 International Conference on Computer Vision (ICCV '11)*. IEEE Computer Society, Washington, DC, USA, 2548–2555. DOI: http://dx.doi.org/10.1109/ICCV.2011.6126542

[33] Chunyuan Liao, François Guimbretière, Ken Hinckley, Ken Hinckley, and Jim Hollan. 2008. Papiercraft: A Gesture-based Command System for Interactive Paper. *ACM Trans. Comput.-Hum. Interact.* 14, 4, Article 18 (Jan. 2008), 27 pages. DOI: http://dx.doi.org/10.1145/1314683.1314686

[34] David G. Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision* 60, 2 (Nov. 2004), 91–110. DOI: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94

[35] Min Lu, Jie Liang, Yu Zhang, Guozheng Li, Siming Chen, Zongru Li, and X. Yuan. 2017. Interaction+: Interaction enhancement for web-based visualizations. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, USA, 61–70. DOI: http://dx.doi.org/10.1109/PACIFICVIS.2017.8031580

[36] J Matas, O Chum, M Urban, and T Pajdla. 2004. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing* 22, 10 (2004), 761 – 767. DOI: http://dx.doi.org/10.1016/j.imavis.2004.02.006 British Machine Vision Computing 2002.

[37] Norman Meuschke, Christopher Gondek, Daniel Seebacher, Corinna Breitinger, Daniel Keim, and Bela Gipp. 2018. An Adaptive Image-based Plagiarism Detection Approach. In *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries (JCDL '18)*. ACM, New York, NY, USA, 131–140. DOI: http://dx.doi.org/10.1145/3197026.3197042

[38] K. Mikolajczyk and C. Schmid. 2005. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27, 10 (Oct 2005), 1615–1630. DOI: http://dx.doi.org/10.1109/TPAMI.2005.188

[39] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. 2005. A Comparison of Affine Region Detectors. *Int. J. Comput. Vision* 65, 1-2 (Nov. 2005), 43–72. DOI: http://dx.doi.org/10.1007/s11263-005-3848-x

[40] Theodor Holm Nelson. 1995. The Heart of Connection: Hypermedia Unified by Transclusion. *Commun. ACM* 38, 8 (Aug. 1995), 31–33. DOI: http://dx.doi.org/10.1145/208344.208353

[41] Daniel Nüst, Markus Konkol, Edzer Pebesma, Christian Kray, Marc Schutzeichel, Holger Przibytzin, and Jörg Lorenz. 2017. Opening the Publication Process with Executable Research Compendia. *D-Lib Magazine* 23 (01 2017). DOI: http://dx.doi.org/10.1045/january2017-nuest

[42] Plotly. 2012. Plotly. (2012). Retrieved August 1st, 2019 from https://plot.ly/

[43] Qt. 2008. QWebEngineView. (2008). Retrieved April 7th, 2019 from https://doc.qt.io/qt-5/qwebengineview.html

[44] E. Rosten, R. Porter, and T. Drummond. 2010. Faster and Better: A Machine Learning Approach to Corner Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32, 1 (Jan 2010), 105–119. DOI: http://dx.doi.org/10.1109/TPAMI.2008.275

[45] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An Efficient Alternative to SIFT or SURF. In *Proceedings of the 2011 International Conference on Computer Vision (ICCV '11)*. IEEE Computer Society, Washington, DC, USA, 2564–2571. DOI:http://dx.doi.org/10.1109/ICCV.2011.6126544

[46] Eric Saund, David Fleet, Daniel Larner, and James Mahoney. 2003. Perceptually-supported Image Editing of Text and Graphics. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (UIST '03)*. ACM, New York, NY, USA, 183–192. DOI: http://dx.doi.org/10.1145/964696.964717

[47] Jianbo Shi and C. Tomasi. 1994. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, USA, 593–600. DOI:http://dx.doi.org/10.1109/CVPR.1994.323794

[48] Desney S. Tan, Brian Meyers, and Mary Czerwinski. 2004. WinCuts: Manipulating Arbitrary Window Regions for More Effective Use of Screen Space. In *CHI '04 Extended Abstracts on Human Factors in Computing Systems (CHI EA '04)*. ACM, New York, NY, USA, 1525–1528. DOI: http://dx.doi.org/10.1145/985921.986106

[49] Bret Victor. 2011. Explorable Explanations. (2011). Retrieved April 7th, 2019 from http://worrydream.com/ExplorableExplanations/

[50] Bret Victor. 2014. Humane Representation of Thought: A Trail Map for the 21st Century. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, New York, NY, USA, 699–699. DOI: http://dx.doi.org/10.1145/2642918.2642920

[51] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky. 2000. Guidelines for Using Multiple Views in Information Visualization. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '00)*. ACM, New York, NY, USA, 110–119. DOI: http://dx.doi.org/10.1145/345513.345271

[52] Wikipedia. 2005. DTrace. (2005). Retrieved April 7th, 2019 from https://en.wikipedia.org/wiki/DTrace

[53] Wikipedia. 2007. EPUB. (2007). Retrieved April 7th, 2019 from https://en.wikipedia.org/wiki/EPUB

[54] Ron Yeh, Chunyuan Liao, Scott Klemmer, François Guimbretière, Brian Lee, Boyko Kakaradov, Jeannie Stamberger, and Andreas Paepcke. 2006. ButterflyNet: A Mobile Capture and Access System for Field Biology Research. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. ACM, New York, NY, USA, 571–580. DOI: http://dx.doi.org/10.1145/1124772.1124859

[55] Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. 2009. Sikuli: Using GUI Screenshots for Search and Automation. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology (UIST '09)*. ACM, New York, NY, USA, 183–192. DOI: http://dx.doi.org/10.1145/1622176.1622213

[56] Luke S. Zettlemoyer and Robert St. Amant. 1999. A Visual Medium for Programmatic Control of Interactive Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*. ACM, New York, NY, USA, 199–206. DOI: http://dx.doi.org/10.1145/302979.303039