Show Me the Invisible: Visualizing Hidden Content

Thomas Geymayer
Graz University of Technology
geymayer@icg.tugraz.at

Markus Steinberger
Graz University of Technology
steinberger@icg.tugraz.at

Alexander Lex
Harvard University
alex@seas.harvard.edu

Marc Streit
Johannes Kepler
University Linz
marc.streit@jku.at

Dieter Schmalstieg
Graz University of Technology
schmalstieg@icg.tugraz.at

ABSTRACT

Content on computer screens is often inaccessible to users because it is hidden, e.g., occluded by other windows, outside the viewport, or overlooked. In search tasks, the efficient retrieval of sought content is important. Current software, however, only provides limited support to visualize hidden occurrences and rarely supports search synchronization crossing application boundaries. To remedy this situation, we introduce two novel visualization methods to guide users to hidden content. Our first method generates awareness for occluded or out-of-viewport content using see-through visualization. For content that is either outside the screen's viewport or for data sources not opened at all, our second method shows off-screen indicators and an on-demand smart preview. To reduce the chances of overlooking content, we use visual links, i.e., visible edges, to connect the visible content or the visible representations of the hidden content. We show the validity of our methods in a user study, which demonstrates that our technique enables a faster localization of hidden content compared to traditional search functionality and thereby assists users in information retrieval tasks.

Author Keywords

Hidden Content; Off-Screen Content; Occluded Content; Visual Linking.

ACM Classification Keywords

H.5.2. Information Interfaces and Presentation: Graphical User Interfaces

INTRODUCTION

When analyzing multiple data sources and documents simultaneously, not all information can be kept in plain sight. There are several causes for information being hidden from a user. First, many documents contain more content than can sensibly be displayed at a time, making it necessary to show only a small fraction of the information contained. We refer to the visible section of a document as its viewport, which, in order to read the whole document, must be changed through scrolling and zooming. Second, it is common that many (partially) occluding application windows are open, potentially covering relevant information. A third cause of content being hidden are minimized windows. To manage limited screen-space, users regularly minimize windows, which may contain important information. Finally, relevant information may be contained in unopened or not accessed documents.

Common strategies to identify relevant information include *searching* and *brushing*. Users can *search* for specific keywords and explore the context of individual occurrences. While zooming and scrolling can be used to manually search through content, most applications also provide a built-in search function that highlights occurrences of a search term. Often, users can advance the viewport of the application to the next matching term. *Brushing* is similar to searching, but instead of an explicit query, the query is implicitly given by selecting an existing item. Further occurrences of the selected item may be highlighted and/or linked. Brushing is common in visualization tools and text-editors.

By going through all open application windows and searching or brushing in each application in turn, users are able to eventually access all relevant information. As search functions are also available for searching through online documents and files stored on disk, all types of initially hidden content could possibly be explored. For certain tasks this kind of workflow is sensible. It is, for example, sufficient for looking up words in a dictionary, where each term has one entry.

There are, however, other tasks where seeing a global structure between individual pieces of information is highly relevant. Consider a user investigating the causes of a crisis in a company. The investigator needs to tie together information from multiple sources. These may involve dozens of reports with hundreds of pages each, multiple spreadsheets documenting money flow, web exposes of other companies involved, travel routes of employees, etc.

If the investigator finds an interesting fact, she will typically want to cross-reference other occurrences of the fact. For many documents this is inefficient using the sequential search approach. In this paper, we introduce techniques that instead let users see an overview of all occurrences, no matter in which document they are contained or if they are visible or hidden, and jump directly to the most interesting occurrence(s).

CHI'14, April 26–May 1, 2014, Toronto, ON, Canada.
ACM 978-1-4503-2473-1/14/04$15.00.
http://dx.doi.org/10.1145/2556288.2557032

Directly accessing all content, both visible and hidden, poses a number of challenges. To address these challenges, we elicited six requirements that a technique for visualizing hidden content needs to address. This list evolved out of an initial set of requirements which we based on our experience with designing visualization interfaces for application domains such as systems biology and our own needs when working on search tasks. We then created multiple prototypes, which we informally evaluated with users, leading us to the following final set of requirements:

R I: Mental map. Users should be aided in building a mental map of the explored documents. Ideally, this overview should make use of the investigator's spatial memory by pointing out occurrences at their actual position on the desktop (if occluded), or at least preserve relative arrangement of occurrences (if currently not on the desktop). For such a mapping, it is necessary to communicate the information's *location* or at least the *direction* in which information can be found.

R II: Indicate occurrences & relevance. Sources or sections mentioning a search term multiple times tend to be more relevant compared to documents or parts that only contain a few mentions of a term. Thus, users may want to prioritize densely populated documents or sections in their search. To support this requirement, visual cues should indicate the amount of information available at a specific location or in a certain direction.

R III: Fast previews. To enable users to quickly judge whether looking at a specific region in a document in detail could be interesting, getting fast previews of hidden content should be possible.

R IV: Fast navigation. Fast navigation between all available chunks of information is essential for an efficient exploration. Once a user has chosen to closely investigate a piece of hidden information, it should be possible to quickly navigate there with minimal interaction.

R V: Heterogeneous sources. The integration of information from different sources is essential to capture all types of hidden content. The technique should connect information from open application windows, minimized windows, and documents which are only present as a file on a disk or available on the Internet. All sources should be handled in a unified manner to allow for all types of hidden content.

R VI: Changing content. The technique must be able to accommodate arbitrary changes to the content presented on the desktop. In particular, the arrangement of windows on the desktop can change at any time: Windows can be moved, resized, minimized, or closed. Moreover, the content and viewport of windows may change at any time, possibly removing existing information, adding new information, or changing location and visibility of information.

In addition to content being hidden, visible content can also be overlooked [4]. While traditional highlighting techniques such as color work reasonably well on a uniform background, a cluttered environment, e.g., due to many windows, or other factors such as large display sizes can increase the chances of relevant content being overlooked [14]. A remedy for overlooking content are visual links [21], explicit edges connecting individual pieces of information.

Our contribution are two novel visualization techniques for hidden content and their combination with visual links into one sophisticated information exploration system that integrates multiple applications and data sources. First, we present **smart links**, a see-through visualization to extend visual links to occluded content. Second, for out-of-screen content, we present a combination of visual links and **smart previews**, aiding the user in quickly estimating the amount of relevant information and allowing accelerated navigation to the information. Additionally, we show how minimized windows and information from files on disk can be integrated into the visualization. Using these two techniques, we provide guidance to all aforementioned types of hidden content. We conducted an exploratory user study evaluating the performance of our hidden content visualization methods relative to standard search functionality. The results of this study indicate that visualization of hidden content is superior to traditional methods in both quantitative and qualitative measures.

RELATED WORK

As mentioned previously, the two main reasons for available content not being considered is that it is *hidden*, i.e., not visible to the user, or that it is *overlooked* even though it is technically visible [4]. We first discuss techniques to show and access hidden content, followed by techniques that help avoid overlooking content.

Hidden Content

For hidden content, we distinguish between techniques that reveal occluded or out-of-viewport content that could be shown within the given display space and techniques that visualize off-screen content.

A common strategy to reveal **occluded content** are see-through interfaces that cull away the foreground to reveal the background. A prominent example are magic lenses [5], which in their general form can not only remove foreground but also alter the representation in arbitrary ways. Magic lenses are typically invoked manually and locally (only the elements within a 'lens' are changed), a characteristic that is not suitable for search tasks. While completely transparent windows [11] and user interface elements [12] have been evaluated in the past, in practice they are rarely used when reading and interacting with content in window managers. An approach that uses transparency for 'unimportant' window regions is described by Ishak and Feiner [18], while Baudisch and Gutwin introduce 'multiblending' [1], as a smarter alternative to alpha blending that considers multiple image features and makes the blending results more readable. A more sophisticated approach is taken by Waldner et al. [29], who superimpose 'clip-outs' of occluded content on top of occluding components. They use a measure of salience to reveal salient occluded content in regions where the occluding elements are not salient. Steinberger et al. [20] additionally scale unimportant content to reveal occluded content. None of these approaches use a semantic measure of relevance, i.e., consider what is currently relevant to a user. Search tasks, however, inherently require revealing specific semantic elements.

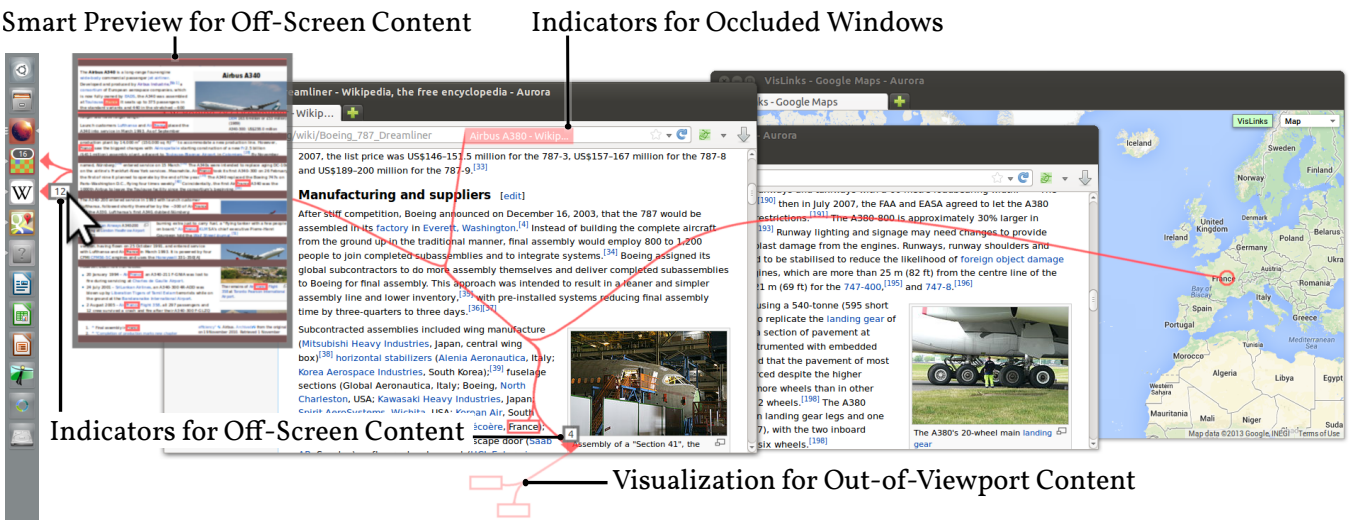


Figure 1: Visualization of occluded, out-of-viewport, and off-screen content. Multiple browser windows and files stored on the computer contain occurrences of the search term "France". Most occurrences are either outside one of the browser's viewports, which is indicated by arrows and links to the out-of-viewport locations, or are occluded by another window, which is indicated by semi-transparent red window labels. The arrows pointing to applications in the taskbar indicate off-screen content in a minimized window (Wikipedia) containing 12 occurrences of the search term and 16 files found by the desktop search engine. Hovering over an arrow, reveals a smart preview showing the regions of the documents containing occurrences.

A common strategy to visualize **off-screen** content are marks rendered within the visible area that point to off-screen content. Baudisch and Rosenholz introduce *halos* [2], circles with the center at the (off-screen) point of interest and a radius chosen so that a segment of the circle intersects with the display. Similar in spirit are *wedges* [10], a technique that uses triangle instead of circles. Users can infer the location of the point from the size and position of the circle or triangle segment. A problem of halos and, to a lesser extend of wedges, is scalability and clutter if many targets should be indicated. To remedy this, Waldner et al. [27] introduce arrows that aggregate multiple proximate points of interest and indicate how many items are aggregated with the size of the arrow. Highly abstracted document previews are another technique to visualize off-screen content. Eick and Ball use such abstract representations for visualizing software [9]. Hearst combines them with a visualization of the search term density in specific regions of text [13]. This technique has been extended by Dieberger and Russel [8] to consider multiple search terms and enable fast navigation and preview. The occurrences of search terms can also be superimposed on the scroll bar [6], a technique that is nowadays, for example, employed in web browsers and in software development tools. However, all of these tools and techniques use very abstract representations. We believe that using a representation that preserves the appearance of the document under inspection will provide additional benefit to users. In our work we combine both, marks to indicate the presence of off-screen context and on-demand smart previews to minimize clutter and enable efficient discovery of and navigation to off-screen content.

Overlooked Content

Search combined with highlighting is a common method to minimize overlooking of content. The highlighting typically employs adding a colored frame, but other methods such as magnifying search terms [25] or interesting parts while shrinking [16, 17] or even completely removing other content is possible [3]. Such methods are also used for generally 'interesting' content [16]. Stoffel et al. use a similar approach for thumbnail previews, where they distort important terms, while preserving the layout of the document [22].

A different approach is to employ connectedness [19], i.e., visual links [7], as a method to highlight occurrences of a search term, which are beneficial especially in a cluttered environment such as in an information visualization system [7, 23, 24, 26, 27] or on large screens where not all content is in a user's field of view [14, 28]. Hoffman et al. [14] have evaluated various techniques to help users locate windows on large displays and found that links (trails) outperform conventional highlighting (frames); similar results have been found for conventional displays [21]. We believe that visual links are the most powerful method to point to content distributed over the screen, especially in a highly heterogeneous environment spanning multiple windows and various types of documents, and consequently have chosen to combine them with methods to visualize hidden content in this paper.

VISUALIZING HIDDEN CONTENT

For visualizing hidden content we take two steps: indicating that relevant content is hidden in the first place, and revealing the hidden content on demand. We introduce novel techniques to visualize **occluded content**, i.e., content that is occluded by other windows, **out-of-viewport content**, i.e., content that is outside of the application window but within the limits of the display, and **off-screen content**, i.e., content that is outside of the available display space or that is in closed, minimized,

(a) (b)

Figure 2: (a) Smart links pointing to out-of-viewport occurrences of a search term. (b) Hovering over a smart link reveals a semi-transparent overlay showing the actual content.

or not accessed documents. All of these techniques are integrated with visual links to create a strong visual connection between all related pieces of information—hidden as well as visible—resulting in an information exploration interface that makes hidden content easily accessible and reduces the risk of overlooking content (see Figure 1).

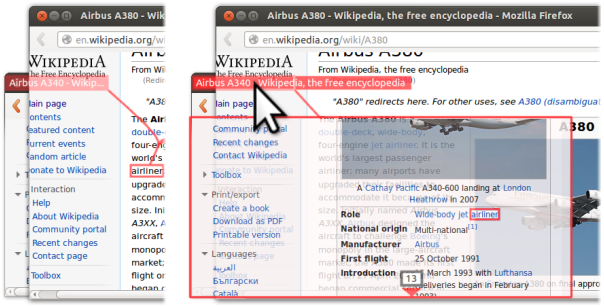
Visualizing Out-of-Viewport Content

Relevant content is often hidden due to the limited viewport size. Large portions of content can be situated in virtual space outside of a window's viewport. We distinguish two cases of out-of-viewport content: (1) The target region is outside the current viewport, but would be visible on the screen if the application's viewport was extended, and (2) the target region is outside the screen. We consider the latter case as off-screen content, which is treated in a later section.

For the former case we use *smart links*: semi-transparent outlines—one for each target region—that indicate the location of invisible target regions (Figure 2(a)) and are connected to other visible or hidden occurrences with visual links. Smart links clearly indicate the occurrence and relevance of hidden content (R II) and support a user's mental map (R I). If the user hovers over a target region outside the application's viewport, as shown in Figure 2(b), all of the application's content that fits on the desktop is rendered in order to provide context to the otherwise 'disembodied' target region, thus providing fast previews of the hidden content (R III). When the user selects such a target region, the application's viewport is automatically centered around the region, allowing the user to immediately continue working with the application at the chosen position, thus facilitating fast navigation (R IV).

Visualizing Occluded Content

To direct the user's attention to windows containing occluded content, we use markers, connected to visual links, as shown in Figure 3(a) that contain the title of the windows where relevant content was found. We chose this approach over showing direct links to occluded content (as we do for out-of-viewport regions), since user-feedback indicated that direct links produce too much clutter. When hovering over such a marker, we overlay the hidden window semi-transparently and highlight and connect the relevant content (R III), as shown in Figure 3(b). To avoid interference with the background window, the overlay can optionally be shown completely opaque.



(a) (b)

Figure 3: (a) A marker showing a part of the window title indicates occluded content. (b) Hovering over the marker reveals a transparent preview of the occluded window, including highlights for the occluded content.

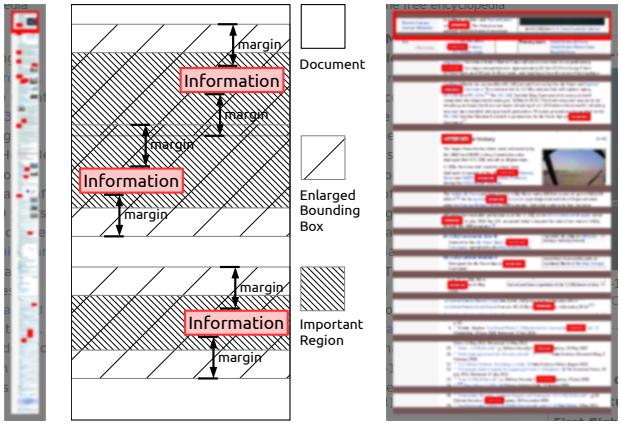
To ensure fast navigation to the occluded region (R IV), a click with the mouse on the overlay moves the respective window to the top of the window stack, thus permanently revealing the occluded information.

Visualizing Off-Screen Content

If a target region is located either outside the screen, within a minimized windows, or in unopened files, it is not possible to draw a link to a specific target region. Instead, we visualize off-screen content by drawing an arrow pointing into the direction of the target, or at the icons representing the minimized windows and unopened files. To avoid clutter in cases where multiple target regions are off-screen, we only draw a single arrow for each icon or window edge. For the latter case we adjust the arrow to point towards the center of gravity of all outside regions. Additionally, we draw a text label next to the arrow to show the number of hidden target regions in the given direction (see Figure 1). These encodings indicate occurrences and relevance (R II).

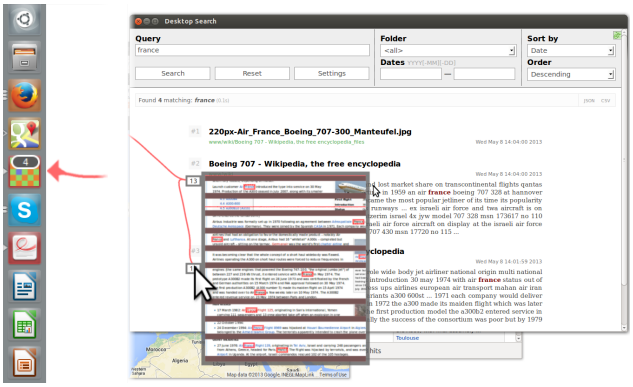
To get an overview of the whole document (R I) and to enable fast navigation (R IV), we provide a *smart preview* of the complete document, which appears when hovering over the arrow. For search tasks, it is reasonable to assume that users are only interested in those parts of the document that contain relevant information. We use this consideration to present the user with a more compact preview where regions containing no relevant information are clipped, freeing up space for increasing the size of interesting areas (see Figure 4). In this way, all hidden regions of the document are presented at once. To decide which areas should be removed, we first calculate bounding boxes of all highlighted regions, then loop through all bounding boxes and mark regions with a certain margin above and below as important and finally hide all unmarked and therefore unimportant regions (see Figure 4(c)).

These smart previews can be zoomed and panned to explore all target regions in detail (R III). To facilitate orientation (R I), the current viewport of the application is highlighted using a rectangle in the preview. Once a target region has been identified by the user, clicking on the target region hides



(a) (b) (c)

Figure 4: Smart preview. (a) A preview of a complete web page with the relevant regions highlighted. The necessary scaling makes it hard to recognize information. (b) All regions containing relevant information are detected and embedded within a bounding box, that makes sure some context is retained. Overlapping bounding boxes are merged. (c) By clipping the unimportant regions much more detail for the relevant parts is revealed.



(a) (b)

Figure 5: Links to unopened documents. (a) The icon of a desktop search engine is marked with an arrow. The number shown inside the icon indicates the number of documents containing the current search term. (b) Within the desktop search engine, occurrences are highlighted and smart previews are provided.

the preview and scrolls the document to the location of the requested information (R IV).

When using the smart preview for **minimized windows**, we draw an arrow next to the application's icon in the task bar to indicate that it contains target regions. Upon hovering over the arrow, the smart preview is revealed (see Figure 1). When the user selects a target region, the minimized window is restored and the viewport is centered on the selected target.

Information may also only be available in **unopened documents**. To integrate unopened documents, we query a desktop search engine to find occurrences of search terms. Similar to our approach for minimized applications, we draw an arrow next to the desktop search engine's icon in the task bar and show the number of documents found within the icon, as shown in Figure 5(a). When opening the search engine, we highlight the search term in the search engine's previews and provide smart previews to reveal the details, as shown in Figure 5(b).

IMPLEMENTATION

To fully integrate heterogeneous data sources (R V), we implemented a central service written in C++, which runs in the background as a standalone application and accepts connects from other applications taking part in the visualization. These other applications are integrated using a minimally invasive approach through the use of a plug-in API or minor modifications of the application source code, if no API is provided [27]. The data exchange between the server and the clients is handled using WebSockets. To add the described user interface components on top of the existing screen content, we create a transparent Qt window which we render using OpenGL.

After the server has received a user-triggered search string, it forwards the request to every connected client application, enabling each application to add its regions. Upon receiving a request, each client searches its content for instances of the requested identifier and reports back the bounding boxes of all found occurrences. For simple selection types, like individual words, bounding boxes already provide an accurate approximation of the relevant region. To highlight and link more complex shapes, such as objects in a map or graph, a client is free to use arbitrary polygons for representing its regions.

Visual links are drawn between all highlighted regions. The naive approach that connects all highlighted regions to a common center results in a cluttered visualization. To remedy this, we bundle links using force-directed edge bundling [15], an algorithm based on an iteratively refined system of control points, attracting each other. The system is initialized by calculating the center of gravity of all occurrences. Then, the highlight region closest to the center of gravity is determined, and all other regions are connected to it. Moving the center of gravity avoids an artificial branching point. Next, all links are subdivided into segments of approximately equal length and finally, force-directed edge bundling is applied. Due to potentially large differences in the length of individual links, the forces affecting a single link can change rapidly, leading to sharp corners in the link routes. To address this issue, we apply a geometric smoothing on the points forming the link routes after executing the bundling algorithm.

All rendering output is directed to an off-screen buffer first. This buffer is twice the size of the screen and copied into the fullscreen window. Using hardware accelerated texture filtering, the visualization is automatically smoothed while being downscaled. The use of alpha-transparency allows blending with the desktop content. Screen pixels that are not covered by the visualization are masked, to allow mouse events passing

through. In this way, the user can interact with all content that is not covered by our visualization.

For rendering see-through visualizations and smart previews, the client application is required to send an image of its content to the server. We use a hierarchical tile map, where each level consists of a single or multiple tiles, which add up to a full preview image of the client application's content at a specific zoom level. Using different resolution images for the individual zoom levels, we create tiles in a resolution that is sufficient for a single level of zoom. As the user zooms into the preview or moves the viewport, missing tiles are asynchronously requested from the corresponding client application.

While working in a desktop environment, the arrangement of opened windows can change at any time (R VI). As this possibly affects the position and occlusion of regions, we need to react to such changes. Current operating systems usually do not allow receiving notifications for changes in windows of other applications. As a workaround, we use a window monitoring component, which periodically requests a list of all opened windows, including their geometry and stacking order, and compares this information with the previous state. If any changes are detected, the server triggers the recreation of the visualization for all active identifiers.

As a proof of concept, we have integrated several widely used applications into our system. A browser add-on allows searching for words or phrases matching a given search identifier. The bounding boxes of all found occurrences, both within and out of the current viewport, are sent back to the server for further processing. As a non-textual example, we have used the Google Maps JavaScript API[1] to create a mash-up which supports the search for geographic locations by name and the retrieval of corresponding screen coordinates on the map. Retrieving the name of a location on a map and querying by this name is also supported. For connecting minimized windows and the desktop search engine, the location of the associated icons needs to be known. Therefore, we query the list of windows in the task bar and calculate the exact location of each icon using the known icon sizes. To retrieve the data for the smart preview for unopened files, they are opened in the background in their respective applications while visual feedback is suppressed.

For a basic integration with our system (to show the number and location of elements) applications need to implement a simple WebSockets protocol, which all modern browsers support. For a full integration, applications need to provide imagery of hidden areas for the preview, which is typically supported in either the GUI library or the graphics API.

Our implementation is open source and can be downloaded at `http://hidden-content.caleydo.org`. Due to the use of cross-platform libraries, our implementation runs on Linux, Microsoft Windows, and Apple OS X. Our technique can be used interactively on all operating systems and is usually able to update all visualizations within half a second. Getting imagery for showing a smart preview of a hidden document requires roughly the same time it takes to open an additional tab in a browser. Due to the delay of an application notifying our system of viewport changes like scrolling and window resize operation a short delay until the graphics elements update is unavoidable. During our study no user raised any concerns regarding the performance of our implementation.

[1] `https://developers.google.com/maps/documentation/javascript/tutorial`

EXPLORATORY USER STUDY

We conducted an exploratory user study to evaluate our hidden content visualization technique for three desktop scenarios of varying difficulty. The scenarios involved up to twelve web browser windows. To gather meaningful feedback, a major part of the study focused on an informal post test interview about the used hidden content visualization techniques. We recruited 18 participants from a local university (aged 20 to 37, 3 females) with a background in computer graphics and visualization. 10 participants indicated that they have experience with visual data analysis.

Techniques

As baseline condition we used the **standard searching and highlighting** technique of the web browser (Firefox), synchronized across all browser windows, to isolate the effect of our visualization. Participants could mark (brush) words in the web browser and press a keyboard shortcut (*CTRL+F*) to search the document for the marked word. The found terms use the default colored box to highlight all occurrences. Pressing the *Enter* button repeatedly advanced to the next occurrence of the word within the same application. As an alternative method, participants could also type the search term into a text field. To search for the same term in a different application window, participants only had to switch the window and continue with the inspection of the highlighted occurrences. A new search term replaced the previous one. Switching windows was possible through standard operating system features.

We tested the baseline condition against our fully functional, real-time **hidden content visualization** implementation, as described before. The ability to interact with all types of links was provided, which either brought covered windows to the front or scrolled the regions outside the viewport into sight. The procedure to trigger a search was identical to the one used in the baseline condition.

Tasks and Apparatus

Participants were asked to perform three information analysis tasks, with thirteen desktop windows opened concurrently. The desktop was presented to the participants on a 22" monitor with 1920×1080 pixels. Participants were seated approximately at the same distance from the monitor. No restrictions where enforced during the test, i.e., users were allowed to move or to rearrange windows. All tasks dealt with properties of aircrafts and airports. To generate two setups with equivalent complexity, we have altered two original data sets taken from Wikipedia to include the exact same number of hidden and visible regions for different properties.

The default window setup is shown in Figure 6. We placed a master browser window on the left side of the screen which

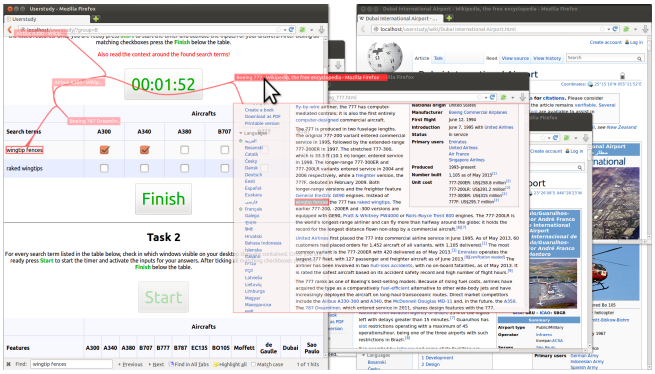


Figure 6: Window setup for the user study. Twelve browser windows are randomly arranged. Another window on the left, is used by the participants to start new trials and indicate their answers.

was used to start a new trial and indicate answers. Twelve additional windows were randomly distributed across the whole screen, each approximately one third the size of the screen. The synchronized highlighting and guidance functionality was provided among all thirteen windows. Due to the large number of windows, participants were confronted with numerous overlapping windows and covered regions. Also, the shown documents were about six to ten times larger than the viewports they were displayed in.

We consciously avoided tabbed browsing, i.e., nested window management, in the study, since it would have made the design more complex and we were primarily interested in testing the performance of our techniques with larger numbers of (partially) overlapping windows.

We designed the following tasks with increasing complexity:

Task 1 was designed as a simple information retrieval task testing the effectiveness of finding information. Participants had to find a single keyword in a subset of the open windows. They were additionally asked to take information from the context area of the search results into account. Participants should tell whether each of the six aircrafts described within the documents is controlled using a yoke or a side-stick. For two aircrafts the information was unavailable and no answer was required.

Task 2 was designed as a more complex search task testing the efficiency of locating documents containing relevant information. Participants were asked to tell the experimenter which of two given keywords were contained within every of the twelve browser windows. The windows contained none, one, or both keywords.

Task 3 was designed as a research task testing the effectiveness of finding information. It required long content interaction, using three search terms, scanning through entire paragraphs, and reasoning. Participants were asked to find all aircrafts equipped with a certain number of engines. The number of engines was indicated by a specific phrase. Participants had to read the context surrounding the phrase, as some documents contained the given phrase even though the aircraft had a different number of engines. Out of the six documents describing an aircraft, three contained the requested number of engines. For the remaining aircrafts, participants where asked which of them have two more features installed, resulting first in two and finally one matching aircraft.

Design and Procedure

The study was conducted as a within-subjects experiment over the described two conditions and the three tasks for each condition. For each task, we measured task completion time and correctness. To start a trial, participants clicked on a button located in the master window on screen. The same window included check boxes for answering the questions and completing the trial. We automatically measured the time between the initiation and competition of a trial. Prior to each task, participants were given a warm-up period, which allowed them to become familiar with the technique and the content of the application windows. After each condition, participants were required to assess their subjective satisfaction with the technique on a questionnaire containing six questions. After the hidden content visualization condition, we presented them with an additional questionnaire, comparing the individual approaches we employ for the different kinds of hidden content. Upon completion of the experiment, the participants were asked to take part in an unstructured interview.

Participants completed all three tasks twice, once with each technique. To avoid learning effects due to knowledge of the data, all tasks where available with two different sets of keywords or features. To reduce the influence of learning effects due to the repetition of tasks, the order of the conditions and the assignment of the task sets was counterbalanced.

Hypothesis

The goal of the user study was to compare the effectiveness and efficiency of our hidden content visualization techniques and traditional search for finding hidden content. We formulated the following three hypotheses for this experiment:

[H1] Using the hidden content visualization leads to a faster retrieval of hidden data. Our techniques visualize all hidden regions that are placed within the boundaries of the screen and offers a preview for regions which are outside the screen. By interacting with these links, every hidden region can be accessed with one or two clicks. Thus, we expect our hidden content visualization to be faster than a sequential search through all hidden regions.

[H2] With the hidden content visualization fewer errors are made. Because we visualize every occurrence of a search term, we expect users to miss fewer occurrences than by stepping through all windows and occurrences within windows sequentially.

[H3] Visualizing hidden content has a positive impact on understanding the spatial distribution of the data. Our hidden content visualization either shows the exact location of hidden regions or points towards the direction where they can be found. Thus, we hypothesize that it is easier for users to orient themselves within the data, if our visualization technique is

Figure 7: Mean completion time with standard error. Our hidden content visualization performs significantly better for tasks with a lot of unrelated content (Task 2) or only simple information retrieval (Task 1). For the research task (Task 3) the differences are not significant.

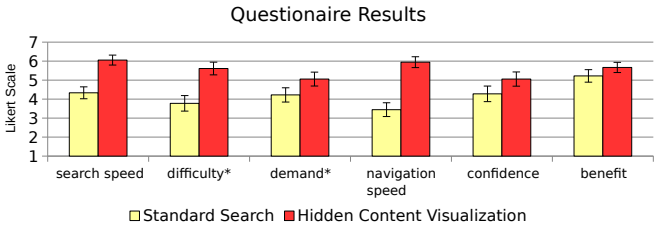


Figure 8: Mean questionnaire results given on a seven point Likert-scale. Higher results are better in all cases (* elements have been inverted).

used. As a consequence, we also expect it to be easier for users to locate data they have not yet explored.

Results

We measured the time participants needed to complete each task, the correctness of the reported numbers, and subjective assessments, which were given on a seven-point Likert scale. As no category tested entirely positive for being normal distributed, all measures were evaluated using non-parametric tests. Wilcoxon signed rank tests ($\alpha = .05$) were used for completion time, error rate, and the subjective task evaluation. The results comparing the individual approaches used by our hidden content visualization were analyzed using Kruskal-Wallis tests ($\alpha = .05$). Timing results are illustrated in Figure 7, and questionnaire results are provided in Figure 8 and 9.

Our analysis revealed a difference in completion time between the techniques for Task 1 ($W = 141$, $p < .005$) and Task 2 ($W = 171$, $p < .001$). We found no significant difference in completion time between the techniques for Task 3 ($W = 80$, $p = .085$).

The average number of errors was very low for all tasks and techniques (traditional search: 0.05; hidden content visualization: 0.04). There was no measurable difference to be found for error.

We found a significant difference for the questionnaire items *subjective search speed* ('I could find the hidden content quickly.') ($W = 129$, $p < .005$), *subjective difficulty* ('It was very hard to find all hidden elements.') ($W = 106$, $p = .022$), and *subjective navigation speed* ('I could navigate to hidden content very quickly.') ($W = 135$, $p = .003$). The differences in *subjective demand* ('The task was very mentally demanding.') ($W = 70$, $p = .07$), *subjective confidence* ('I am sure I did not miss any highlighted elements.') ($W = 49$, $p = .08$) and *subjective benefit* ('The technique would be beneficial for my every day computer work.') ($W = 38$, $p = .29$) were not statistically significant.

Observations and Feedback

All participants quickly developed successful strategies using our hidden content visualization. Once a new search process is initiated, the viewport of each application is moved to contain the first occurrence of the search term. This allowed the fastest participants accessing the required information directly in the see-through visualization or smart preview without moving the viewport to the highlighted regions. This strategy could even be completed with the lowest zoom-level. Most participants, however, zoomed in one level, which eased the reading of the preview.

For regions outside the viewport, participants developed different strategies. If both regions outside the viewport and outside the screen appeared at the same time, participants most often used the smart preview only, because it also includes regions outside the viewport. About half of the participants clicked on the regions to scroll them into sight; the other half hovered over the region and read the information directly from the superimposed see-through preview. Many participants stated that they did not recognize regions outside the viewport very often and they would prefer the system treating these regions like regions outside the screen. Some participants thought that this visualization is only useful with a maximum of two or three windows.

If multiple windows containing scrolled-away content had large overlapping areas, some participants had problems to recognize which window each visualization belongs to. In the interview, they mentioned it would be useful to use different colors for different windows or have the covered window marker also for partially covered windows.

In the hidden content visualization condition, no participant used the window manager to switch between open windows. Most of them mentioned that the hidden content visualizations enables them to locate target windows and regions faster than before. Two participants stated that they would like to have the indicators for found information inside the task-bar or window list, as they think a one-dimensional search within a linear list is faster than a two-dimensional search for highlights on the whole screen.

With a small amount of interesting content, participants sometimes were slower at marking a search term, initiating a new visualization process, opening the smart preview and zoom/scroll to the location than simply scrolling there. Some participants also mentioned that they think the hidden content visualization is not useful for simple search tasks involving little content, but gets increasingly useful with complex tasks involving multiple windows.

Participants said that they would especially like to see the visualization feature to support tabs within applications and connect spreadsheet applications as well as code editors and documentations. Additionally, integrating a call hierarchy or

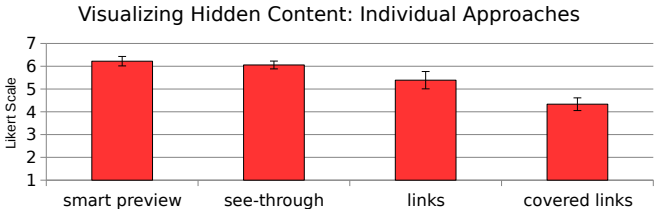


Figure 9: Mean questionnaire results for the different approaches combined our hidden content visualization given on a seven point Likert-scale. No significant difference was found in the data.

references to a variable inside a software development environment was considered as a useful extension. Also enhancing the smart preview by showing only headings of the previewed document and expand or navigate to the sections content on demand would increase the acceptance as an everyday tool.

One participant tried to use scrolling within a see-through preview and stated in the interview the scrolling would be a useful extension to the preview.

Discussion

Based on the significantly lower task time achieved with our hidden content visualization for Task 1 and 2 and the higher subjective search and navigation speed, we conclude that *H1* is supported, and our hidden content visualization leads to a faster retrieval of hidden content. The results also indicate that the usefulness of our technique increases with the number of windows containing no relevant content. For Task 2 only half of the windows contained relevant content, which caused participants using standard search to check twice as many windows as required. Task 3 required far less navigation to different windows and, additionally, involved a reduced number of windows during each step. The largest part of the task required retrieving and combining information from the content. We believe that the major time-consuming activity for Task 3 was understanding and interpreting content instead of navigation and that this resulted in no significantly faster results for our hidden content visualization. Due to the low error rate, we can neither accept nor reject *H2*. Based on the participant feedback that our visualization helps not to miss elements, it might be possible to confirm this hypothesis in a more extensive study. Because all participants stated that our visualization helped them to get a better location awareness inside documents, we conclude that *H3* is also supported.

The overall positive user feedback, in particular for the smart preview and the see-through visualization of the smart links (see Figure 9), indicates that visualizing hidden content is a useful tool especially for complex information retrieval tasks possibly with a high amount of unrelated content.

CONCLUSIONS AND FUTURE WORK

In this paper we presented techniques for visualizing search terms in areas of documents that are covered by other windows, outside the window's current viewport, outside the screen, contained in a minimized window, or in unopened files. We introduced smart previews that allow efficient exploration of content outside visible areas by providing a content-sensitive, space conserving, compressed preview of the whole document's virtual area, whereas smart links paired with a transparent overlay and click-to-show functionality allows for a fast navigation to covered content.

Future work will aim to reduce the visual clutter that may arise when a large number of regions are selected. We envision a combination of more sophisticated bundling algorithms, context-preserving routing [21], and smart fading of the links over time to further improve the situation.

Moreover, to improve temporal consistency and reduce lag, we plan to extend our system to track changes of relevant objects over time and incorporate temporal coherence into the layout planning. This would also allow us to avoid radical layout changes resulting from small changes (such as scrolling by a single line) in the underlying scene.

Finally, we want to pick up comments from study participants and plan to integrate our hidden content visualization techniques with an integrated development environment such as Eclipse, where search tasks and visiting all references and modifications of specific variables are crucial tasks in developing and debugging software.

ACKNOWLEDGMENTS

The authors wish to thank Manuela Waldner for her important suggestions. This research was funded by the Austrian Science Fund (FWF): P22902 and J 3437-N15, the government of Styria (A3-22.M-5/2012-21), the Austrian Research Promotion Agency (840232), and the Air Force Research Laboratory and DARPA grant FA8750-12-C-0300.

REFERENCES

1. Baudisch, P., and Gutwin, C. Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*, ACM (2004), 367–374.
2. Baudisch, P., and Rosenholtz, R. Halo: a technique for visualizing off-screen objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '03)*, ACM (2003), 481–488.
3. Baudisch, P., Xie, X., Wang, C., and Ma, W.-Y. Collapse-to-zoom: Viewing web pages on small screen devices by interactively removing irrelevant content. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '04)*, ACM (2004), 91–94.
4. Bezerianos, A., Dragicevic, P., and Balakrishnan, R. Mnemonic rendering: an image-based approach for exposing hidden changes in dynamic displays. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '06)*, ACM (2006), 159–168.
5. Bier, E. A., Stone, M. C., Pier, K., Buxton, W., and DeRose, T. D. Toolglass and magic lenses: the see-through interface. In *Proceedings of the Conference*

on Computer Graphics and Interactive Techniques (SIGGRAPH '93), ACM (1993), 73–80.

6. Byrd, D. A scrollbar-based visualization for document navigation. In *Proceedings of the ACM Conference on Digital Libraries (DL '99)*, ACM (1999), 122–129.
7. Collins, C., and Carpendale, S. VisLink: revealing relationships amongst visualizations. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '07) 13*, 6 (2007), 1192–1199.
8. Dieberger, A., and Russell, D. Exploratory navigation in large multimedia documents using context lenses. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS '02)* (2002), 911–917.
9. Eick, S., Steffen, J., and Sumner, E.E., J. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering 18*, 11 (1992), 957 –968.
10. Gustafson, S., Baudisch, P., Gutwin, C., and Irani, P. Wedge: Clutter-free visualization of off-screen locations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*, ACM (2008), 787–796.
11. Harrison, B. L., Ishii, H., Vicente, K. J., and Buxton, W. A. S. Transparent layered user interfaces: An evaluation of a display design to enhance focused and divided attention. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*, ACM (1995), 317–324.
12. Harrison, B. L., Kurtenbach, G., and Vicente, K. J. An experimental evaluation of transparent user interface tools and information content. In *Proceedings of the ACM Symposium on User Interface and Software Technology (UIST '95)*, ACM (1995), 81–90.
13. Hearst, M. A. TileBars: visualization of term distribution information in full text information access. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*, ACM (1995), 59–66.
14. Hoffmann, R., Baudisch, P., and Weld, D. S. Evaluating visual cues for window switching on large screens. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)* (2008), 929–938.
15. Holten, D., and van Wijk, J. Force-directed edge bundling for graph visualization. *Computer Graphics Forum (EuroVis '09) 28*, 3 (2009), 983–990.
16. Hornbaek, K., and Frøkjær, E. Reading of electronic documents: The usability of linear, fisheye, and Overview+Detail interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '01)*, ACM (2001), 293–300.
17. Hornbæk, K., and Frøkjær, E. Reading patterns and usability in visualizations of electronic documents. *ACM Transactions on Computer-Human Interaction 10*, 2 (2003), 119–149.
18. Ishak, E. W., and Feiner, S. K. Interacting with hidden content using content-aware free-space transparency. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '04)*, ACM (2004), 189.
19. Palmer, S., and Rock, I. Rethinking perceptual organization: the role of uniform connectedness. *Psychonomic Bulletin and Review 1*, 1 (1994), 29–55.
20. Steinberger, M., Waldner, M., and Schmalstieg, D. Interactive self-organizing windows. *Computer Graphics Forum 31*, 2pt3 (May 2012), 621–630.
21. Steinberger, M., Waldner, M., Streit, M., Lex, A., and Schmalstieg, D. Context-preserving visual links. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '11) 17*, 12 (2011), 2249–2258.
22. Stoffel, A., Strobelt, H., Deussen, O., and Keim, D. A. Document thumbnails with variable text scaling. *Computer Graphics Forum 31*, 3pt3 (2012), 1165–1173.
23. Streit, M., Kalkusch, M., Kashofer, K., and Schmalstieg, D. Navigation and exploration of interconnected pathways. *Computer Graphics Forum (EuroVis '08) 27*, 3 (2008), 951–958.
24. Streit, M., Lex, A., Kalkusch, M., Zatloukal, K., and Schmalstieg, D. Caleydo: Connecting pathways and gene expression. *Bioinformatics 25*, 20 (2009), 2760–2761.
25. Suh, B., Woodruff, A., Rosenholtz, R., and Glass, A. Popout prism: Adding perceptual principles to Overview+Detail document interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '02)*, ACM (2002), 251–258.
26. Viau, C., and McGuffin, M. J. ConnectedCharts: explicit visualization of relationships between data graphics. *Computer Graphics Forum 31*, 3pt4 (2012), 1285–1294.
27. Waldner, M., Puff, W., Lex, A., Streit, M., and Schmalstieg, D. Visual links across applications. In *Proceedings of the Conference on Graphics Interface (GI '10)*, Canadian Human-Computer Communications Society (2010), 129–136.
28. Waldner, M., and Schmalstieg, D. Collaborative information linking: Bridging knowledge gaps between users by linking across applications. In *Proceeding of the IEEE Symposium on Pacific Visualization (PacificVis '11)*, IEEE (2011), 115–122.
29. Waldner, M., Steinberger, M., Grasset, R., and Schmalstieg, D. Importance-driven compositing window managment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*, ACM (2011), 956–968.