Mnemonic Rendering: An Image-Based Approach for Exposing Hidden Changes in Dynamic Displays

Anastasia Bezerianos, Pierre Dragicevic, Ravin Balakrishnan
Department of Computer Science
University of Toronto
{anab, dragice, ravin}@dgp.toronto.edu

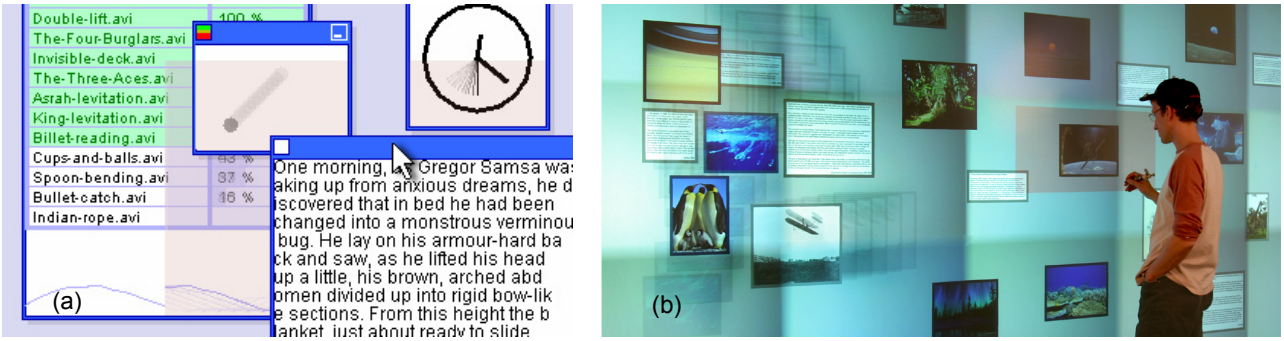

Figure 1. (a) The *Mnemonic Desktop* with pixel persistence and pixel flashback techniques: hidden parts of windows are being revealed showing motion trails and a dusty appearance that will fade out as changes are being replayed. (b) The *Mnemonic Wall*: visual changes outside the user's central vision leave motion trails that will exhibit similar behavior as the user glances at them.

ABSTRACT
Managing large amounts of dynamic visual information involves understanding changes happening out of the user's sight. In this paper, we show how current software does not adequately support users in this task, and motivate the need for a more general approach. We propose an image-based storage, visualization, and implicit interaction paradigm called mnemonic rendering that provides better support for handling visual changes. Once implemented on a system, mnemonic rendering techniques can benefit all applications. We explore its rich design space and discuss its expected benefits as well as limitations based on feedback from users of a small-screen and a wall-size prototype.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces - Graphical user interfaces.

General terms: Design, Human Factors

Keywords: Change visualization, mnemonic rendering.

INTRODUCTION
In current computing displays, dynamic visual changes regularly happen without the user being aware of them. For example, a download manager might complete its task in the background, emails might be received in an occluded window, or changes might occur in a partially hidden live video stream. Unless explicit notification or history mechanisms are provided, chances are that the user will miss these changes, or only become aware of them by chance or by explicitly trying to reconstruct them at a later time. Indeed, studies have shown that people are rarely able to spot visual changes when they occur during disruptions as short as eye saccades [25]. Change occurring during longer periods of invisibility should be even harder to reconstruct, because the user might have partly forgotten the previous state or missed significant intermediate changes. In particular, visual transitions often play an important role in the understanding of visual changes [5, 21, 39].

While some applications do provide notifications, these typically work well for single discrete events and not for multiple or continuous events over a period of time. Further, current window managers do not provide any integrated system-wide support for visualizing such dynamic changes occurring in the background, in the user's periphery, or over some past time period.

In this paper, we explore the idea of recording and presenting visual changes that might have been missed by the user due to the changes occurring in the background, periphery, or at a time where the user was otherwise preoccupied. As part of this exploration, we first survey existing approaches and discuss their limitations. We then develop a general image-based storage, visualization, and implicit interaction paradigm to provide better support for handling visual changes, called *mnemonic rendering*. The term *mnemonic* is intended to capture the essence of storing, cueing, access to memory, and reminiscence. Mnemonic rendering assumes that each pixel knows whether it is visible to the user or not and buffers its color changes when invisible so that no information is lost. It then restitutes these changes to the user as soon as it becomes visible again. We present three restitution techniques, and discuss usability issues.

UIST'06, October 15–18, 2006, Montreux, Switzerland.

BACKGROUND

Navigating within Visually Dynamic Digital Worlds

Since the advent of multitasked and networked computing, concurrent interaction with a variety of applications has become the norm. One aspect that varies across applications is the user's level of control: whereas word-processing and authoring applications predominantly obey the user, network communication tools or computer process monitors show autonomous, "dynamic" behavior. Furthermore, hybrid applications that are highly reactive to both user and exogenous events will become more numerous with the increase of collaborative groupware applications [30].

Whether dynamically evolving or not, the total amount of visual information available to the user at a given time most often exceeds hardware display or human perceptual capabilities, pointing towards the need to partition the digital world either virtually or physically [12]. In window-based computing systems, a large part of visual information available simultaneously is hidden, while the perceived visibility of each piece of information regularly changes as the user proactively switches between tasks [14].

Recent large-scale studies have shown that some primary information is typically kept visible during long time intervals, while secondary information is accessed regularly [14, 15]. Interestingly, the studies on window systems also suggest that secondary information is often of a more dynamic nature than the primary. Dynamic information is often purposely hidden to preserve privacy or to avoid visual disruptions during the primary task [15]. Similar usage patterns have been observed on multiple monitor configurations in which secondary information is often relegated to peripheral vision [12]. They have also been observed, to a lesser extent, on virtual desktops [26].

Clearly, an important component of the average computer user's activity currently involves regular polling of heterogeneous and dynamic visual information, which is not visible for periods of time. Given the limits of visual perception and attention, this also affects multi-monitor systems and will most likely influence next-generation user interfaces involving large displays or multiple ambient displays.

Supporting Integration of Dynamic Visual Information

Current window environments have been used for more than two decades and most of the time users seem to have no trouble integrating dynamic visual information. The main reason is that windowed applications don't expect to be continuously visible and thus rely on implicit or explicit strategies for easing visual information integration. Commonly used strategies are *persistence, structuring, reduction*, *history* and *notification*. These are *semantic* strategies, in that they depend on the meaning of the visual content:

- *Persistence*. The visual content of a window at a given time is often sufficient to infer what happened. For example, new emails don't simply flash in an email client; they accumulate inside a container and can be accessed via a scrollbar if not visible any more. Persistence ensures that most changes are not lost.
- *Structuring*. Window contents are not just messy visual scenes; they are structured in meaningful ways so that they can be quickly interpreted into discrete, higher-level concepts. Structuring visual information allows users to infer changes using their semantic memory whenever visual short-term memory is not exploitable. For example, previously received emails can be remembered based on the subject or sender.
- *Reduction*. Often the number of possible changes in a window is limited, making them predictable. Changes can also have privileged locations or salient visual features, making them faster to detect. For example, new emails usually appear in a predictable location and in bold font.
- *History*. Time is often part of visualizing information. Combined with persistence, time visualization provides histories that indicate the order in which changes occurred or their respective ages, when relevant. For example, emails are stacked in order of arrival and are time-stamped. Histories can be displayed statically or based on re-visitation.
- *Notification*. Some applications additionally maintain awareness by notifying users of meaningful changes as they occur. For example, users can be easily alerted each time a new email is received.

Limits of These Approaches

The semantic strategies aforementioned are widely adopted in computer applications and seem to provide some support for the user to easily integrate dynamic visual information. However, these approaches have some limitations:

- Notifications are only useful if they are time-critical or relevant to the current goal. Evidence shows that most users are not willing to be interrupted while absorbed in their primary task (for email notifications, see for example [34]). As most events and changes in personal computing are not time-critical, proactive navigation into visual information should be prevailing.
- Semantic structuring alone is not enough to support efficient integration of visual information. Using memory and visual search for spotting changes limits the amount of visual information that can be monitored. For example, the user will hardly remember an entire text document or a folder's content. Likewise, this strategy does not scale well to inherently continuous information: even the value of a single continuous variable displayed as a progress bar or a color can't be accurately memorized by the user.
- The remaining strategies are strongly dependent on the nature of displayed information and are always designed and implemented separately in each application.

Because there is no general approach for supporting visual information integration, we can expect an uneven level of support among applications and the existence of flaws that make the detection of some changes difficult or impossible. Examples of such problems are discussed next.

Some Motivating Real-World Examples

Through an informal survey of twelve regular computer users we identified some examples of everyday tasks being regularly affected by lack of support for occluded changes:

- *Web-based monitoring.* Three users mentioned they often monitor real-time data, for example basketball or hockey scores, from web pages that auto-refresh content. They periodically switch back and forth between their main task and the browser and often find it hard to compare the current data (for example score and who is winning) with the previous time they checked. One user mused that a mechanism for highlighting what changed since the last time the user viewed the page would be useful. Similar issues were pointed out by other users for different types of web-based dynamic content, such as online news sources. Note that there is evidence for strong change blindness on magazine-style web pages [33].
- *Communication.* Three users mentioned having trouble with email, newsgroups and instant messaging (IM) clients. Identifying new emails or newsgroup postings is often challenging for users who keep several older unread messages or switch them to the "unread" state. Such habits have indeed been identified in previous surveys on email usage [34, 38]. Whittaker [38] also observed that unread messages can occasionally scroll out of sight and be forgotten. In IM clients, status changes in long buddy lists seem to often go unnoticed. Additionally, the conversation can sometimes move ahead of where the user left it and she might not know where to pick it up from.
- *Single-user file management.* Five users mentioned diverse issues when manipulating (downloading, copying or moving) files. Lengthy file operations can be initiated and then forgotten, for example if a hidden progress dialog automatically closes once a transfer is completed. Finding a new file in a directory can be hard if its older timestamp has been kept, or if its name is unknown or too similar to other files. Locating a new file is particularly awkward when it has just been added to a cluttered desktop according to some unpredictable heuristic.
- *File sharing.* In peer-to-peer file sharing, file operations are typically lengthy and numerous and users seem to monitor high amounts of information. Two users mentioned their desire to be able to compare the changes in their download and upload status (speed, new seeds, parts finished) since the last time they checked. Two users also mentioned working on a shared project file and sometimes missing files added or deleted by collaborators.
- *Other.* One user who works in a command line environment often finds it difficult to detect presence or absence of new lines added by running processes when switching between terminals. A software developer has trouble noticing new warnings/errors in an IDE while editing code. Interestingly, users can also be blind to their own changes: one user mentions she sometimes types in the wrong window while concentrating on another area, then has trouble identifying the effects of her actions.

AN IMAGE BASED APPROACH

Our survey identified several instances where users found visual changes in their computer environment hard to follow. Occasionally users do not pay attention to the changes, but most often these changes are simply not visible.

Although individual applications could be improved by paying even more attention to visualization, history or notification, it is worth considering simpler approaches that might work globally in an application agnostic manner.

Consider for example what happens when a number of new emails are downloaded while a) the mail client is minimized and subsequently reactivated for use, versus b) launching the mail client afresh. In the first case, the user is instantaneously shown a screen with new and old emails, and has to rely on visual searching for application provided cues such as highlights and dates to discern the new emails from the old. In the second case, newly downloaded emails appear one by one in the inbox and some folders are progressively bolded. No visual search and no semantic memory are involved because all changes simply pop out: motion perception is more powerful than most other strategies [25] and requires nothing more than the actual visual changes to be presented over time.

Following the above example, we argue that a reasonable strategy to pursue would be an image based approach that stores invisible changes at the pixel level for later redisplay, or restitution, to provide the user with a simple visual understanding of changes that they may have missed. Such an approach assumes a pixel-level model of visual changes, which we introduce in the next section.

Surfaces and Pixel Visibility

Suppose that visual information made available to the user is spread among *surfaces*. A surface can be either physical or virtual, and contains dynamically rendered visual information. Whether they are physical or virtual, it is convenient to think about surfaces as unstructured matrices of pixels, which can be rendered off-screen as bitmap images. Although rendered and "made available" to the user, a pixel may or may not be actually visible. There are two reasons why a pixel may be invisible to the user (Figure 2):

- *It is not displayed.* In layered graphical models, surfaces are allowed to overlap so that a pixel may be hidden by a surface of higher priority in terms of z-order. For example, all or part of a window's content can be obscured by another window (Figure 2a). In peephole graphical models such as virtual desktops, potentially very large surfaces are cropped so that only part of them is visible (Figure 2b), known as the "keyhole effect" [39]. Windows can also be minimized or temporarily hidden while the underlying application keeps running and the window still exists in the user's mental model and/or in the computer's memory (Figure 2c).
- *It is not seen by the user*. Even when physically displayed, a pixel can be obscured by a physical object, such as user's hand on a touch screen (Figure 2d) or a physical input device. But most of the time, a pixel is not seen simply because it is outside the user's field of view.

In desktop computing, users are not continuously looking at the screen: a pixel can remain unseen during the time of an eye-blink, a brief talk with a colleague, or absence during the night (Figure 2e). When large and/or multiple displays are being used, a pixel might not be seen because the focus of the user's attention is on another part of the display (Figure 2f).

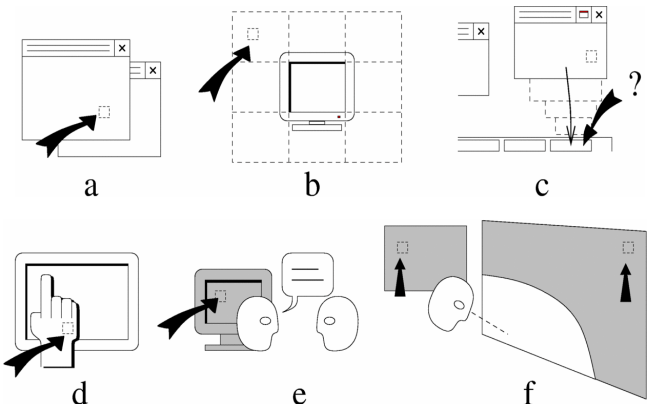

Figure 2. A pixel might be invisible to the user because: (a) of occlusion by another window, (b) it is on a virtual canvas outside the current display area, (c) its window is minimized; or it is displayed but not seen by the user due to: (d) physical occlusion, (e) interruption, (f) being outside the focus of attention.

The first type of invisibility (the pixel is not shown) mainly concerns small displays and desktop computing, whereas it is less of an issue on large and/or multiple displays, that allow more visual information to be shown concurrently. Conversely, the second type of invisibility (the pixel is not seen) occurs on small displays but is a more serious issue on large displays where visual information resides outside the user's field of view, especially during close interaction.

Of course, pixel visibility is rarely a binary property. For example, in layered models supporting translucency such as see-through tools [4], a pixel can be more or less "shown". The concept of "seen" is even more subtle. First, human vision is physiologically non uniform and abilities such as acuity, color discrimination and movement sensitivity are unevenly distributed within the visual field. Second, vision is a global perceptual phenomenon hardly decomposable into individual pixels and involving cognitive mechanisms such as attention [16, 25].

Though human vision is a complex phenomenon, we can make significant advances by adopting a minimalist model of visual perception that is purely *optical* (if something is visible to the eye, then it is perceived) and *binary* (we ignore partially visible information). Given that user's locus of attention is difficult to infer precisely even using specialized hardware [13], such a model is a practical substitute that captures a reasonable subset of possible use cases.

Another important assumption we make is the *single-viewer scenario*: several distant users can manipulate visual content on a surface and a surface can be replicated in other physical displays, but each surface is seen by only one user. This assumption will hold throughout this paper.

Mnemonic Rendering

As the user navigates visual information, the visibility of each pixel is subject to change over time. On surfaces conveying dynamic visual information, the color of each pixel is also subject to change. Discontinuities may arise when changes occur on pixels while they are invisible.

Mnemonic Rendering involves *buffering* pixel changes then *restituting* these changes on the screen. A visible pixel is rendered normally, whereas an invisible pixel stores a time-stamped history of its color changes, so that no transitional information is lost (Figure 3). When the pixel becomes visible again it restitutes its buffered history. After the restitution the pixel is displayed normally.

The goal of buffering and restitution is to aid users in maintaining an accurate mental representation of the state of visual displays, in the presence of invisible changes. This is achieved by storing and presenting the changes themselves in a manner that does not require memorization and comparison with previous states.

Restitution Techniques

Restituting a pixel buffer consists of rendering it in a synthetic way during a limited period of time, possibly dismissing part of the stored information. As an example, we describe two simple types of restitution: a static one called *persistence* and a dynamic one called *flashback* (Figure 3).

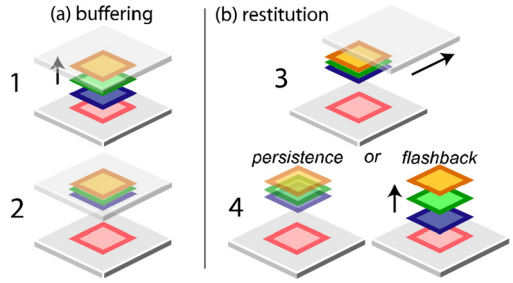

Figure 3. Mnemonic rendering. All changes occurring since a pixel is hidden (1) are buffered (2). Removing the occluding element (3), results in restitution of the entire buffered history, blended (*persistence*) or played back (*flashback*) (4).

Persistence. When a pixel becomes visible it temporarily displays its buffered history as a single color obtained from blending all colors previously buffered. On a macroscopic scale, persistence typically results in *motion trail* effects.

Suppose that an entire surface is hidden, then shown after a change happened on this surface. When the visual change has a simple motion quality (e.g., a translation or a rotation) persistence produces a temporary semitransparent *trail* showing the motion path (the moving gauge in Figure 4a). When the change is more complex (e.g., scrolling text or an animated icon), persistence *blurs* the areas where changes happened (the weather icons in Figure 4a).

Flashback. When a pixel becomes visible it plays back its history in a sped-up timeline. On a macroscopic scale, pixel flashback displays the previously visible state, then replays the changes that occurred in fast-forward motion until the present state is reached (Figure 4b).

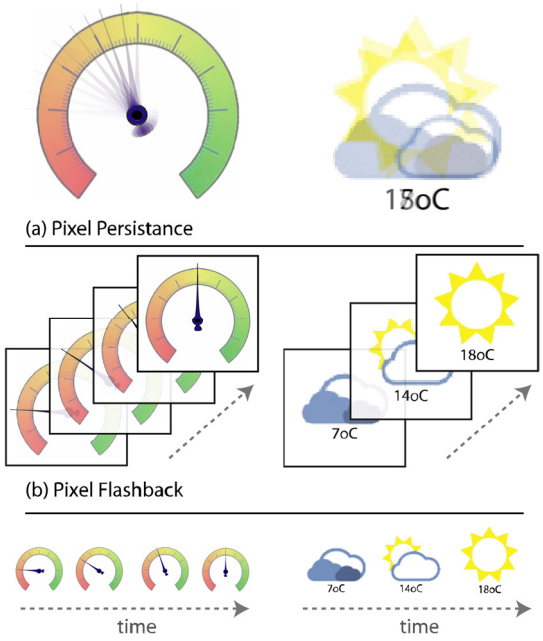

Figure 4. Two examples of using our restitution techniques. (a) Persistence, (b) Flashback.

When adjacent pixels are made visible or invisible at different times, mnemonic rendering results in transient "time-warping" effects and visual breaks (Figure 5). The time-warping effect is produced by an animation that is progressively occluded or progressively shown (e.g., a window moves over another one). Visual breaks can arise, for example, if a part of a window is shown after a change happened on the entire window.

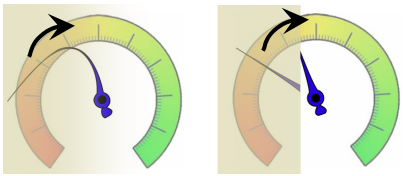

Figure 5. Time-warping (left) and visual break (right) on a moving gauge with the flashback technique. Shading conveys pixel "age".

Although mnemonic rendering is a simple technique, its design space is very rich. First, persistence and flashback can be implemented in several ways and each small variation can result in different visual effects. Second, there are several ways to make the technique more sophisticated and improve its scalability. In the next sections, we go into more details on mnemonic rendering and its variations.

Visual Effects

There are several ways of implementing the persistence technique. The blended color can be obtained by averaging all colors present in the history, which amounts to alpha-blending each pixel with the previous ones using alpha values that follow a geometric progression of scale factor -½. Other averaging weights can be used, as well as other color composition schemes. For example, having weights depending on timestamps can produce fading-out or fading-in trails. Simple pixel superimposition can be used instead of alpha-blending when pixels already include an alpha channel. Objects moving on a transparent background would then leave sharp trails. Other compositing techniques can be used for preserving more information, such as multi-blending [3].

If time between glances is short or old data is unimportant, a transient variant of persistence can be obtained by adding temporal response to color changes, similar to phosphor persistence of old monitors. On oscilloscopes for example, such effect is sought after for viewing complex data.

Color manipulation may also convey status information on a pixel (invisible, restituting or normally shown). For example, applying de-saturation or sepia effects during restitution gives a feel of oldness (see Figure 5), while preserving most of the visual information. Such an effect can fade out throughout the restitution, enhancing the film metaphor while allowing smooth transition to the current visual state.

Timing

Restitution length can be variable (e.g., it can depend on the invisibility duration), but must be short enough to give the user a rapid glimpse of the changes. This is especially true if surfaces are interactive, because the user may want to quickly resume interaction with the application. Typical lengths for GUI animations (i.e., an opening menu) are below one second. So, restitutions must similarly be achievable in a small bounded time.

With the persistence restitution, the averaged pixel color rapidly fades out to reveal the actual pixel color. With the flashback restitution playback can be compressed linearly to achieve bounded restitution time. Frames can be dropped or blended with adjacent frames (a technique known as motion blur). The simplest non-linear time compression technique involves playing color changes successively without taking their timestamp into account. But this technique can desynchronize adjacent pixels, unless grouping is used (see the section on grouping below).

Another timing issue is handling pixel changes during restitution. If restitutions are fast compared to the speed of the actual changes, these pixel changes can be ignored. The techniques can also be easily extended to support concurrent buffering and playing. For example, updating the playback sequence and speed as new pixels are being buffered will ensure continuity with animations currently occurring.

Invisibility Detection

Detecting whether a pixel is shown or not (see Figure 2a,b,c) can be easily done using software techniques. Detecting whether a pixel is seen or not (see Figure 2d,e,f) can be done using a variety of approaches, including:

- *Software-based inference:* on interactive systems, absence can be coarsely inferred from periods of inactivity.
- *Presence detection:* absence can be inferred with better time accuracy using simple vision algorithms or sensors.
- *Head tracking:* tracking location and/or orientation of a user's head allows inferring the user's field of view, thus gaining spatial accuracy.

- *Eye-tracking:* eye tracking gives even more accurate spatial information about the user's locus of attention.
- *Occlusion detection:* in some cases, physical occlusions can be inferred using software approaches (e.g., hand location on a touch screen). In other cases, occlusions can be accurately detected using additional hardware [18].

Non-intrusive hardware solutions exist for all of these approaches [32] and can be spread among displays. For example, displays could be aware of eye contact [35].

Although eye-tracking technology may seem appealing, coarser approaches for visibility detection might actually yield better results. They limit Midas Touch problems [17] and might provide the user with finer control over the restitutions when conservative inference strategies are used. Using head tracking for example, a conservative inference strategy would not assume visibility inside the entire field of vision. Rather, it would add an "area of invisibility" around the user's central vision. The way these invisible pixels are rendered reflect trade-offs between peripheral awareness and distraction reduction. For example, such pixels can display their last visible color, resulting in a non-distractive static rendering. Or actual pixel colors can be shown instead, resulting in a dynamic rendering serving peripheral detection. Intermediary approaches include using persistence to soften change rendering.

Grouping

The time-warping effect mentioned earlier (Figure 5) provides a spatiotemporal visualization of the changes together with invisibility periods. However, such an effect might puzzle the user and visual breaks can hinder her understanding of more global changes.

There are several approaches for ensuring some level of synchronization. First, changes in pixel visibility can be *deferred* so that they occur at the same time. Delaying can rely on clock ticks or on user actions. For example, pixels won't start recording or be restituted before the user has finished dragging an occluding window.

Another approach for ensuring synchronization is grouping pixels so that their visibilities are all updated at the same time. Several grouping strategies can be considered:

- *Surface-wide grouping*. If whole surfaces are seen as groups, then their content will remain visually consistent.
- *Dirty region-based grouping*. For performance concerns, most graphical toolkits repaint only rectangular regions that have changed. Although semantic, this information is very often available at a low-level on the system. Dirty regions often match fixed areas on the surface, but can also be parceled out and follow objects as they move. In that case, global motion groups can be obtained by iteratively accumulating dirty regions that intersect on two successive animation frames.
- *Image-based grouping*. More accurate motion groups can be directly derived from the images. One minimalist approach is drawing pixels that changed and grouping them based on proximity. More sophisticated motion segmentation techniques have been proposed in various fields, including vision and image compression [27]

Once a grouping strategy has been adopted, several approaches can be considered for triggering the groups, i.e., starting the restitutions by marking all pixels visible:

- *Triggering on full visibility*: a group is triggered when it is fully visible. One issue with this approach is that it may be hard or impossible to make a group fully visible, for example if the group runs over the user's field of view.
- *Triggering on partial visibility*: a group is triggered as soon as it is partially visible. Triggering is easy, but may lead to missed changes, for example if the group is partly occluded or runs over the user's field of view.
- *Incremental triggering*: triggering can be done on the first motion group, then on the second one and so on. For example, if an object has moved around the user on a circular display, the user will have to look at the starting point and follow it until the end.

Interactivity

Interaction with mnemonic rendering is most of the time passive in that it exclusively involves changes in pixel visibilities. However, adding active interaction can be useful. For example, when histories are lengthy or complex, the user might prefer to have some control on the restitution, rather than be provided with longer playback times. We briefly describe a lightweight mechanism allowing flexible navigation in the history of changes, which assumes the presence of a pointing device. This technique can be applied to any type of restitution, for replaying a flashback or restoring persistence.

During the restitution and for an extra second afterwards, a semitransparent disc is visible. Before the disc fades out, the user can indicate interest in the restitution by depressing the pointer, in which case the disc changes into a circular representation of the restitution timeline (Figure 6). Restitution is controlled with circular gestures, allowing both coarse and fine control. This is useful when history lengths differ by several orders of magnitude [22, 29].

Feedback and input locations depend on the constraints of the particular interactive system. For example, on a very large display, the input area should be close to the user whereas the disc feedback might be displayed elsewhere within the field of view. On a desktop computer, the input area and the disc feedback can be confounded and displayed at the mouse location when the restitution started, thus being both easy to trigger and to ignore.

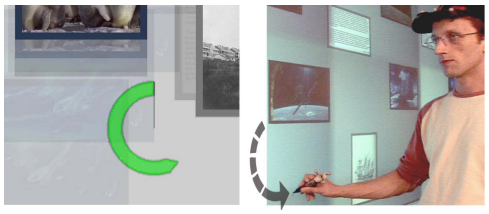

Figure 6. Control dial operated remotely by the user to control restitution playback.

MNEMONIC DESKTOP AND MNEMONIC WALL

In order to test the viability of the mnemonic rendering approach, explore its design space and get early user feedback, we developed two prototypes called Mnemonic Desktop and Mnemonic Wall.

The Mnemonic Desktop

The Mnemonic Desktop is a toy window environment with windows simulating dynamic behaviors such as a clock, text being written by a remote user, a peer-to-peer download manager, a webcam, and moving objects (Figure 1a). These animations can be accelerated by depressing the clock. Each window is a surface which can made fully or partly invisible by moving other windows over it, minimizing it, or moving it outside the desktop.

Three types of restitution are available: an averaging-based pixel persistence that smoothly fades out, a flashback with linear time compression, and a sequential combination of both. The restitutions last about one second, have a fading out "dusty" effect, and can be controlled with the dial previously described. A pixel-level restitution model is used, combined with either a user-based or a clock-based deferred synchronization: when the user uncovers a window, histories are restituted on mouse button release. Using another button, histories are triggered every 1/10 second.

The Mnemonic Desktop is implemented on top of Java2D. In each window, histories are kept for as large as possible rectangular regions. Windows repaint themselves on the screen and on all invisible regions being buffered. When pixels inside a buffered region are no longer in sync (e.g., part of the region becomes visible), the region is cut into smaller pieces to preserve the pixel-level buffering model. The Mnemonic Desktop can be downloaded for testing from: www.dgp.toronto.edu/~anab/mnemonic

The Mnemonic Wall

The Mnemonic Wall is a document organizer running on a 5m x 1.8m back projected high-resolution wall-sized display (Figure 1b). Pictures and text boxes are spread out over the display and can be moved around using a pen, or are animated by the system to simulate a distant user. The Mnemonic Wall also includes a variant of the layout-matching game discussed in the "user feedback" section.

The entire wall-sized display is treated as a single surface. A head-tracking system is used to determine pixel visibilities. The visible area is visualized by a red translucent quadrilateral, whose surface is chosen to encompass foveal and parafoveal vision (Figure 1b). Since head movement and eye amplitudes during gaze shifts follow a tight linear relationship [36], we exaggerate horizontal and vertical head movement values to allow natural head movements.

Three restitutions are supported: persistence, flashback, and a combination of both. They are similar to those used on the Mnemonic Desktop, except that pixels from the surface have alpha channels (some are transparent) and the combined technique shows persistence and flashback simultaneously. Also, a motion-based pixel grouping has been adopted and groups are triggered based on their partial visibility. Restitutions can be controlled with a dial.

The Mnemonic Wall has been implemented in C++ with Chromium (chromium.sourceforge.net) providing graphics rendering over the cluster of computers driving the 18 projectors of our display wall. Regular graphic primitives have been used to mimic the mnemonic rendering effects aforementioned. A Vicon (www.vicon.com) motion tracking system provided head and stylus tracking.

USER FEEDBACK

Mnemonic rendering involves a number of potentially disturbing effects such as overlapping graphics, delayed animations and uncommon use of context information. In order to make sure the techniques were viable (i.e. that at least for one particular type of task they would prove beneficial) we conducted an informal evaluation of the techniques using a layout-matching game. Our goal was to understand how users would utilize the three restitution techniques, to determine how effective or hindering they would find them, and to elicit comments on their design. In this section we describe the task we used and our findings.

The Task

10 participants played a series of games using regular rendering and the three mnemonic rendering techniques (persistence, flashback, persistence/flashback combination). 4 used the Mnemonic Desktop and 6 used the Mnemonic Wall. Each session lasted about half an hour.

The desktop version of the layout-matching game showed on its background two grids *A* and *B,* containing numbers. Periodically, a number was randomly moved to a free location by the system. This background was half-covered by a moveable window containing two smaller grids *a* and *b* whose numbers could be moved by the user (Figure 7, grid *A* is covered). The task was to reorder *a* and *b* so that they match *A* and *B* before a timer runs out. The large display version was similar except it used four remote grids that had to be reproduced on four proximal grids (Figure 8). Given the display's size, while working on one grid, the two opposite grids could not be seen without major body movement. On both versions, no control dial was used.

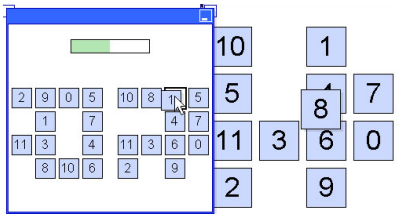

Figure 7. Layout matching game on the Mnemonic Desktop. One grid is covered by the window.

Figure 8. Layout matching game on the Mnemonic Wall, using pixel persistence.

We wanted the task to differ from common computing activities in two respects. Firstly, a faster pace was required to evaluate the techniques in a reasonable time. Secondly, we wanted an engaging, well-defined task, and thus avoided concurrent tasks with unclear priorities. The layout-matching game involves a single primary task with a well-defined goal: match all the layouts as fast as possible.

Nevertheless, the task shares salient characteristics with personal computing activities: it requires glancing for polling information, and while seeing changes as they happen might be useful, the task can still be completed if changes are missed. Completing this task when all changes have been missed is easier than real-life search for changes, since it involves visual search alone (side-by-side comparison of two grids) instead of visual search plus semantic memory. To ensure participant familiarity with this side-by-side comparison strategy, non-mnemonic rendering was presented first. Hence, participants could ignore mnemonic rendering information if they found it unclear or useless.

Results

Relevance of Mnemonic Rendering

Participants easily understood the task. As expected, all participants used a strategy of matching individual grids sequentially. We counted the number of changes users had to treat at each step. Although this was a function of user's speed, values from none to three were the most common.

All users made use of mnemonic rendering for reproducing changes. After playing the game, they also ranked all three restitution techniques higher than regular rendering that required side-by-side comparison. They seem unaffected by Midas Touch, so using head tracking and window occlusion for passively controlling visual rendering seems realistic.

Usefulness and Limits of Persistence

With the *persistence* restitution, occluded moving numbers or numbers moving outside parafoveal vision left a semi-transparent echo or trail behind them. Participants found the technique useful when one or two hidden changes happened, but found it challenging for more than two overlapping changes. However, they used the technique for quickly determining whether any changes happened on a grid, allowing them to spend less time on unchanged grids.

Clearly persistence failed as an explanation mechanism for complex changes, but provided useful synthetic information on the changes that could be quickly interpreted.

Usefulness and Limits of Flashback

With the *flashback* restitution, the numbers stood still if they were occluded or unseen but moved in fast forward motion as soon as the user glanced at them, if changes happened. All participants described the *flashback* technique as being very effective. They noted that after looking at the playback of changes they could replicate the sequence easily. Sometimes users could successfully mimic the motions even when the animation was partly missed or apparently chaotic, with up to four successive motions in one second. More than four motions could hardly be reproduced.

However, on the large display users commented that the lack of feedback in the presence of changes before the actual animations were played back prevented them from using peripheral vision. Some users required an indication as to whether the motion they were viewing was that of an actual change or a flashback.

Clearly the *flashback* visualization we used, although effective in explaining changes, lacks feedback and does not support peripheral awareness.

Persistence and Flashback Combined

With the *persistence-flashback* restitution, trails were shown then faded out during the flashback. Unsurprisingly, on the large display the majority of participants preferred the combined technique. As they pointed out, the blurred trails left by *persistence* both clearly provided rapid information on changes and supported peripheral awareness, whereas the *flashback* component was useful in explaining in detail the actual changes. One participant felt distracted by the persistence in the combination condition. On the small display, two participants preferred the combination whereas the other half preferred the flashback alone.

Design Considerations

Several design-related issues were raised. They mainly concerned the proper choice of visual effects and timing:

Visual effects. When performing the task with the persistence technique, several users commented positively on the use of de-saturation as a metaphor to indicate older states. Nevertheless, they requested a fainter representation of older states, so as not to clutter the screen. Three users commented that the intermediate history need not be too detailed, as long as some transitional information is present.

Timing. One design aspect that particularly concerned us was the timing of the restitution. We found the chosen 1 second to be appropriate for displaying one to three changes without hindering the task. Some participants found it occasionally too slow, while everyone found it too fast for more than four changes. One participant said that he would have liked the ability to replay the last restitution.

Grouping. On the large display, participants were comfortable with the way restitutions were triggered. On the small display, visual breaks occasionally occurred but this did not seem to affect participants.

RELATED WORK

Situation awareness designs. Time critical systems, as in air-traffic control rooms, are particularly concerned with the user maintaining an accurate mental model of the state of a system [1, 10]. They focus on appropriate information visualization and notification of changes [39]. Contrary to these approaches, mnemonic rendering does not assume a high-level system knowledge of the changes. Moreover, changes are assumed non time-critical and their restitution is controlled by the user.

Histories. A large number of recording and re-visitation techniques have been proposed for various purposes including action reversibility [9, 19], replication [20] and improved information search [8, 28, 37]. A few application-

independent approaches to user history storage and navigation have been proposed [11, 23, 24] but they all assume a software architecture or object-oriented protocol for supporting them. In contrast to mnemonic rendering, history-based techniques show changes the user has already visited and are designed for much coarser time granularities.

Groupware. Change visualization in groupware is supported in some commercial applications and has been an active area of research. Tam and Greenberg [30] provide a review of current techniques. Such approaches are necessary for dealing with complex changes such as asynchronous edits of shared workspaces and documents. However, an image-level technique can still be a useful complement for improving awareness on real-time edits.

Window management systems. Apart from notification techniques such as blinking taskbars, there is no system-wide support for change awareness in current window managers. Numerous articles mention the problem of multiple overlapping windows but they essentially focus on facilitating navigation [14, 15, 31]. Some of them propose new manipulation or visualization techniques for reducing occlusion [3, 31], but do not address occlusion issues. As far as we know, no work mentions the use of window visibility as context information.

Multiple and large displays. Multiple monitors helped to reduce occlusion [11] but human attention is limited. Woods [39] identified the problem of integrating information across multiple displays and proposed a set of design heuristics based on the cinematographic concept of "visual momentum". However he was mainly concerned with transitions between displays. He advocated the use of abstract overviews for enhancing peripheral awareness, but did not address the understanding of the actual changes.

Animation. Animation has been used in user interfaces for explaining changes, including smooth transitions based on cartoon or movie techniques [2, 6]. Instead of interpolating motion for better explaining discontinuous changes, our approach restores visual continuity that may be occasionally disrupted by occlusion or attention shift.

CONCLUSIONS and FUTURE WORK

In our increasingly dynamic computer environments visual changes often occur without being seen. After identifying the situations under which changes are missed, we introduced the idea of mnemonic rendering to present these missed changes in a lightweight and fluid manner. Variations of the design space were discussed and three visualization techniques developed and evaluated.

Since mnemonic rendering is image-based, it can be implemented at the window manager level. A new trend in window management is *compositing*, a technique that allows windows to be rendered off-screen and manipulated as images [7]. While its current application lies mainly in cosmetic effects, mnemonic rendering suggests a new compelling use of this graphics power.

Although more investigation is required, initial user feedback indicates that the techniques can be of interest to the general user. Returning to our motivating examples, we feel the use of persistence or flashback could help users identify new items on their desktop or in folders, compare status indicators (download progress, IM buddy lists), or detect changes in web-page content over time.

Mnemonic rendering is application agnostic and may benefit a variety of applications and tasks. Nevertheless, seamless integration of generic and dedicated techniques for supporting change awareness requires further investigation. In addition to the basic system-wide support provided by mnemonic rendering, individual applications could further exploit information on pixel visibility to better handle specific types of changes.

We plan to extend our model to encompass hierarchical surfaces, in order to address issues related to applications that include multiple dynamic sub-surfaces (for example tabbed web-browsers). Although our techniques are aimed at explaining hidden changes, they were not designed to explicitly provide notification of the occurrence of the changes. Clearly the two issues are inter-related and also require further investigation.

A shortcoming of our approach is that it targets single user scenarios. Although this assumption is quite realistic in desktop environments, large display settings might be used by multiple users synchronously. We plan to extend our techniques to a setting with personalized visibility detection and history storing. Clearly the design space of such an extension will be very different. Finally, more formal evaluations are required to determine how our techniques fare under different durations and types of changes.

ACKNOWLEDGEMENTS

We thank Alexis Angelidis, Shahzad Malik, Dan Vogel, John Hancock, members of the Dynamic Graphics Project lab (www.dgp.toronto.edu) and our UIST reviewers.

REFERENCES

1. Banbury, B. and Tremblay, S., eds. (2004). *A cognitive approach to situation awareness: theory and application*. Ashgate.
2. Baudisch, P., Cutrell, E., and Robertson, G. (2003). High-density cursor: A visualization technique that helps users keep track of fast-moving mouse cursors. *Interact*. p. 236-243.
3. Baudisch, P. and Gutwin, C. (2004). Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending. *ACM CHI Conference*. p. 367-374.
4. Bier, E., Stone, M., Pier, K., Buxton, W., and DeRose, T. (1993). Toolglass and Magic Lenses: The see-through interface. *ACM SIGGRAPH*. p. 73-80.
5. Chang, B., Mackinlay, J., Zellweger, P., and Igarashi, T. (1998). A negotiation architecture for fluid documents. *ACM UIST Symposium on User Interface Software and Technology*. p. 123-132.
6. Chang, B. and Ungar, D. (1993). Animation: From cartoons to the user interface. *ACM UIST Symposium on User Interface Software and Technology*. p. 45-55.

7. Chapuis, O. and Roussel, N. (2005). Metisse is not a 3D desktop! *ACM Symposium on User Interface Software and Technology*. p. 13-22.
8. Cockburn, A., Greenberg, S., Jones, S., McKenzie, B., and Moyle, M. (2003). Improving web page revisitation: Analysis, design and evaluation. *Special Issue on Web Navigation Skills, IT&Society*, 3(1). p. 159-183.
9. Edwards, W.K., Igarashi, T., LaMarca, A., and Mynatt, E.D. (2000). A temporal model for multi-level undo and redo. *ACM UIST Symposium on User Interface Software and Technology*. p. 31-40.
10. Endsley, M., Bolté, B., and Jones, D. (2003). Designing for situation awareness: CRC.
11. Freeman, E. and Fertig, S. (1995). Lifestreams: Organizing your electronic life. *AAAI Fall Symposium: AI Applications in Knowledge Navigation and Retrieval*.
12. Grudin, J. (2001). Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. *ACM CHI Conference*. p. 458-465.
13. Hunt, A.R. and Andkingstone, A. (2003). Covert and overt voluntary attention: linked or independent? *Cognitive Brain Research*, 18. p. 102–105.
14. Hutchings, D.R., Smith, G., Meyers, B., Czerwinski, M., and Robertson, G. (2004). Display space usage and window management operation comparisons between single monitor and multiple monitor users. *ACM AVI Conference on Advanced Visual Interfaces*. p. 32-39.
15. Hutchings, D.R. and Stasko, J. (2004). Revisiting display space management: understanding current practice to inform next-generation design. *Graphics Interface Conference*. p. 127-134.
16. Itti, L., Rees, G., and Tsotsos, J., eds. (2005). *Neurobiology of attention*. Elsevier.
17. Jacob, R.J.K. (1993). Eye-movement-based human-computer interaction techniques: Toward non-command interfaces. In *Advances in Human-Computer Interaction*, D. Hix, Editor. Ablex. chapter 6. p. 151-190.
18. Jaynes, C., Webb, S., Steele, R.M., Brown, M., and Seales, W.B. (2001). Dynamic shadow removal from front projection displays. *IEEE Visualization*. p. 175–182.
19. Kawasaki, Y. and Igarashi, T. (2004). Regional undo for spreadsheets. *ACM UIST Symposium on User Interface Software and Technology*. Demo abstract.
20. Kurlander, D. and Feiner, S. (1992). A history-based macro by example system. *ACM UIST Symposium on User Interface Software and Technology*. p. 99-106.
21. Mertz, C., Chatty, S., and Vinot, J. (2000). Pushing the limits of ATC user interface design beyond S&M interaction: the DigiStrips experience. *3rd USA/Europe Air Traffic Management R&D Seminar*.
22. Moscovich, T. and Hughes, J.F. (2004). Navigating documents with the virtual scroll ring. *ACM UIST Symposium on User Interface Software and Technology*. p. 57-60.
23. Rekimoto, J. (1999). Time-machine computing: a time-centric approach for the information environment. *ACM UIST Symposium on User Interface Software and Technology*. p. 45-54.
24. Renaud, K. (2000). Expediting rapid recovery from interruptions by providing a visualization of application activity. *OZCHI*. p. 348-355.
25. Rensink, R.A. (2002). Change detection. *Annual Review of Psychology*, 53. p. 245-277.
26. Ringel, M. (2003). When one isn't enough: An analysis of virtual desktop usage strategies and their implications for design. *ACM CHI Conference (Ext. Abstracts)*. p. 762-763.
27. Salembier, P. and Marques, F. (1999). Region-based representations of image and video: Segmentation tools for multimedia services. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8). p. 1147-1169.
28. Skopik, A. and Gutwin, C. (2005). Improving revisitation in fisheye views with visit wear. *ACM CHI Conference*. p. 771-780.
29. Smith, G.M. and schraefel, m.c. (2004). The radial scroll tool: scrolling support for stylus- or touch-based document navigation. *ACM UIST Symposium on User Interface Software and Technology*. p. 53-56.
30. Tam, J. and Greenberg, S. (In press - accepted 2005). A framework for asynchronous change awareness in collaborative documents and workspaces. *International Journal of Human Computer Studies*.
31. Tomitsch, M. (2003). Trends and evolution of window interfaces. Thesis, University of Technology: Vienna.
32. Turk, M. (2004). Computer vision in the interface. *Communications of the ACM*, 47(1). p. 60 - 67.
33. Varakin, D.A. and Levin, D.T. (2004). Unseen and unaware: Implications of recent research on failures of visual awareness for human–computer interface design. *Human-Computer Interaction*, 19. p. 389-422.
34. Venolia, G.D., Dabbish, L., Cadiz, J.J., and Gupta, A. (2001). Supporting email Workflow. Tech Report MSR-TR-2001-88. Microsoft Research.
35. Vertegaal, R., Mamuji, A., Sohn, C., and Cheng, D. Media eyepliances: using eye tracking for remote control focus selection of appliances. *ACM CHI Conference (Extended Abstracts)*. p. 1861-1864.
36. Wang, X. and Jin, J. (2001). A quantitative analysis for decomposing visual signal of the gaze displacement. *Pan-Sydney area workshop on visual informtaion processing conference on visual information processing 2001*. p. 153-159.
37. Wexelblat, A. and Maes, P. (1999). Footprints: History-rich tools for information foraging. *ACM CHI Conference*. p. 279-277.
38. Whittaker, S. and Sidner, C. (1996). Email overload: exploring personal information management of email. *ACM CHI Conference*. p. 276-283.
39. Woods, D.D. and Watts, J.C. (1997). How not to have to navigate through too many displays. In *Handbook of Human-Computer Interaction*. Elsevier Science.