Focus Plus Context Screens: Combining Display Technology with Visualization Techniques

Patrick Baudisch, Nathaniel Good, and Paul Stewart
Information Sciences and Technologies Laboratory
Xerox Palo Alto Research Center
Palo Alto, CA 94304, U.S.A.
+1 (650) 812 4656
{baudisch, ngood, stewart}@parc.xerox.com

Abstract
Computer users working with large visual documents, such as large layouts, blueprints, or maps perform tasks that require them to simultaneously access overview information while working on details. To avoid the need for zooming, users currently have to choose between using a sufficiently large screen or applying appropriate visualization techniques. Currently available hi-res "wall-size" screens, however, are cost-intensive, space-intensive, or both. Visualization techniques allow the user to more efficiently use the given screen space, but in exchange they either require the user to switch between multiple views or they introduce distortion.

In this paper, we present a novel approach to simultaneously display focus and context information. *Focus plus context screens* consist of a hi-res display and a larger low-res display. Image content is displayed such that the scaling of the display content is preserved, while its resolution may vary according to which display region it is displayed in. Focus plus context screens are applicable to practically all tasks that currently use overviews or fisheye views, but unlike these visualization techniques, focus plus context screens provide a single, non-distorted view. We present a prototype that seamlessly integrates an LCD with a projection screen and demonstrate four applications that we have adapted so far.

Keywords
Display, focus plus context screen, mixed resolution, overview plus detail, fisheye view, video projector.

INTRODUCTION

Faster computers, inexpensive memory, and large storage have brought the ability to work with larger amounts of information to the computer user. While computational power and storage have increased rapidly over the past few years, the screen size and resolution available to consumers has not. This is an issue when users work with large visual objects, where overall structure is as important as detail information for getting the task accomplished.

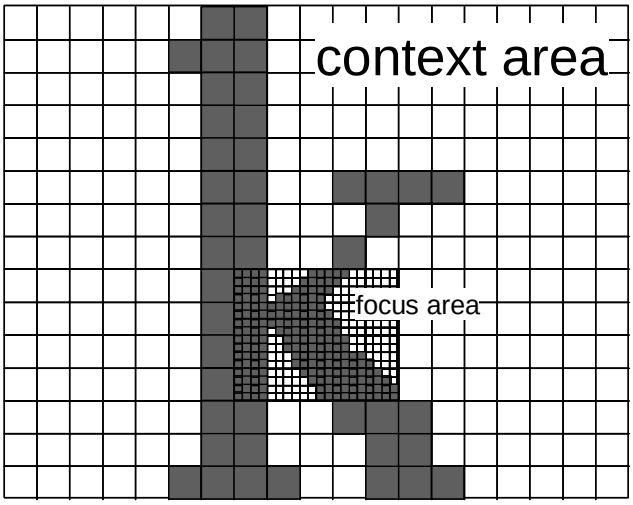

Figure 1: Focus plus context screens consist of low-res regions and hi-res regions. Image content displayed across regions preserves its scaling, while its resolution changes.

Designers working in the print industry, for example, have to make sure that the hardcopies of their layouts look perfect, whether they are viewed from a distance or close-up. Because print offers a much higher resolution than computer screens, examination of all possible facets of the layout via a computer screen involves a substantial amount of zooming. Similar needs for zooming occur when architects use a CAD program to edit blueprints, when radiologists analyze X-ray images on the screen, or when people examine a large city map on their computer screens. In all these cases, the display becomes the bottleneck of the computer systems.

When a user's display is not able to show the number of pixels required for displaying the entire content at the desired level of detail, the users can navigate the display

to acquire the information sequentially. Additional navigation means additional user effort, which motivated researchers to explore other solutions to the problem. One approach involves replacing the current screen with a larger screen capable of displaying the required number of pixels. Another approach is to provide an appropriate visualization technique that allows fitting the required data into a smaller screen by reducing the space allocated for irrelevant information. In the following two sections, we will look at how existing display technologies and visualization techniques deal with these problems.

Related work in large hi-res displays

We know of no technology that allows production of one piece, high-quality displays of arbitrary size. Proposed techniques typically involve combining multiple smaller displays into a single large display of high pixel count.

One common solution is to connect two or more computer monitors to a single computer, as supported by current operating systems, such as Microsoft Windows. In this setup, all connected monitors form a single logical display. This display allows windows to be moved across display borders and large windows to span multiple displays. However, Grudin [8] observed that the visible gap between individual monitors discouraged users from having windows span multiple displays. His study suggests that users instead use additional monitors to separate windows belonging to different tasks.

In order for large displays to overcome this effect, a substantial amount of research has been invested in the creation of *seamlessly* tiled display systems [6]. Several solutions for avoiding the visible seams have been proposed, including the integration of up to four identical LCDs by butting them together into a single large LCD (a 9-megapixel display by IBM[1],), the construction of video walls by assembling back projection modules with small borders[2], as well as a series of research prototypes evolving around tiled hi-res projection displays [10]. Compound displays composed of multiple display units surrounding the user have been used in virtual reality, such as flight simulation [15], and in immersive environments, such as the CAVE [4]. These proposed solutions are still cost-intensive, space-intensive, or both, which has prevented these technologies from reaching the mass market.

Besides this work, which attempts to obtain large homogeneous displays, some research has been done in hybrid display systems. In the context of computer supported cooperative work, multiple displays have been combined loosely in order to provide users with personal space as well as shared workspace. I-LAND [23] gives users a shared workspace by providing a large projected area that users can interact with and use for collaboration. Jun Rekimoto's [19] augmented surfaces project allows notebooks to overlap the projection space and permits users to drag material between the notebook screen and the shared projection area. However, users of the system reported that the disproportionate scaling between the notebooks and the projection area was distracting.

Feiner proposed a hybrid display combining a semi-transparent head-mounted display with a conventional CRT monitor [5] (see [2] for more recent work on this track). This display used the monitor to show a selected portion of a larger X-Windows desktop, while the low-res head-mounted display gave an overview of the same desktop. In the overview, the image displayed by the goggles continued the monitor image virtually into the room. This solution, however, was limited by the lag of the head tracking apparatus that was required for aligning the goggles and the monitor. This lag caused the image content displayed across both displays to be temporarily disrupted whenever the user moved his or her head.

Related work in visualization techniques

Research in visualization techniques has resulted in methods for fitting more relevant data onto a given screen by reducing the space allocated for irrelevant information. Plaisant [17] and more recently Olston and Woodruff [16] provide overviews of the various types of visualization techniques in use.

The most prominent techniques for reducing navigation overhead are overview plus detail and fisheye views [7,3]. Overview plus detail visualizations [13] use two distinct views: one showing a close up and the other showing the entire document. While this technique helps users to orient themselves in large spaces, the drawback of this approach is that it requires users to visually switch back and forth between the two distinct views and to re-orient themselves within the view every time they do so.

By using non-linear scaling, focus plus context visualization techniques, such as fisheye views [7,3] and Document Lens [20] allow users to see one or more selected *focus* regions in full detail, while information in the periphery (the *context region*) is compressed to occupy less screen space. The benefit of fisheye views is that they keep adjacent information together, thereby avoiding the need for users to explicitly switch between multiple views. This provides for faster switching between the detail region and the periphery. The main drawback of fisheye views is the distortion that they introduce. This makes them inappropriate for content where proportions and distances matter. Photographic content, for example, easily becomes unrecognizable, which limits the applicability of fisheye views to such tasks as visual design.

FOCUS PLUS CONTEXT SCREENS

We propose *focus plus context screens* (f+c screens), as a new way of fitting a larger piece of large visual objects into a display in order to let users save zooming interactions. Focus plus context screens open a new field of re-

[1] http://www.research.ibm.com/resources/news/20010627_display.shtml

[2] http://www.panasonic.com/medical_industrial/11-16-00.asp

search, which—as emphasized by their name—is located in the intersection between display technology and visualization techniques.

Figure 1 shows the general concept. Focus plus context screens offer regions of high resolution and regions of low resolution[3]. Image contents preserve their scaling, even when their resolution varies. The geometry of the displayed content, i.e. the ratio between lengths in the image, is thereby preserved.

Figure 2: The high-res region in the center of this focus plus context screen provides users with detail information.

Figure 2 shows a photo of our prototype system displaying a map. The focus display is located at the same location as in Figure 1, but correct calibration of the display renders it invisible from a distance. However, the callout showing a close-up of the border region between the two resolutions unveils pixels of different sizes.

Focus plus context screens implement regions of different resolution by combining multiple display units of different resolution. Building a focus plus context screen therefore starts with the choice and configuration of display hardware. The rest of this paper is outlined as follows: In the following sections, we will describe the hardware and software requirements for a focus plus context screens, the methodologies used for combining them, as well as a concrete setup. Since software implementations can be application-specific, we will begin the second half of this paper with a presentation of the applications we built so far, followed by a description of the software implementation. A presentation of early results and a discussion of the achievements will then conclude the paper.

[3] The term *resolution*, measured, for example, in pixels per inch, determines in how much detail image content can be displayed. Note that the term resolution is sometimes misused for communicating the number of pixels offered by a display (e.g. "a resolution of 1024x768 pixels"), which is *not* the meaning we refer to when using the term resolution.

Requirements

To make sure that users perceive and use focus plus context screens as a single display and avoid the task-serration effect observed with two-headed displays, it is crucial to preserve image geometry across displays and minimize any gaps found on the display surface.

When we refer to geometry preservation, we mean that the lengths displayed are scaled representatives of the original image. When this is true, other attributes such as distances, proportions, and relative size will retain their fidelity in the system. Two-headed displays, as supported by MS Windows, for example, do not preserve geometry. Pixels located across display borders are logically adjacent, although on a setup with two monitors these two pixels are separated by the physical gap between the monitors.

In addition to preserving the image geometry the resulting image should be free of gaps. In display systems preserving image geometry, a visible gap between display units results in missing image content. While users are familiar with the fact that displays are finite and that clipping occurs at the display border, clipping inside the display space will typically be disrupting. Gaps within the display area should therefore be avoided.

When users change the angle from which they view the display, surfaces not located in the same physical plane can block portions of each other. This effect can also lead to perceived visible gaps and noticeable misalignment. This can be avoided if head tracking is used. However, this can result in the aforementioned lag in displaying the new images, which in turn distorts the geometry of the images. To avoid the described drawbacks, display units should generally be located in the same plane.

Combining multiple display units

Displays to be combined typically have a certain thickness and borders with certain widths and depths (depth denoting how far the border extends over the display plane). Figure 3 shows ways of combining two coplanar screens to form a single display.

Figure 3a shows a configuration that combines the two displays alongside each other. The benefit of this arrangement is that both display areas are in the same plane. The drawback of this arrangement is that the gap

between the displays is at least the sum of the border widths of the two displays. When the display borders are small, this solution works well.

Figure 3b shows a configuration where one display unit is located in front of the other. While this setup does not allow the two displays to be in the same plane, it minimizes the gap between the two displays units. The gap is now only determined by the border width of the front display.

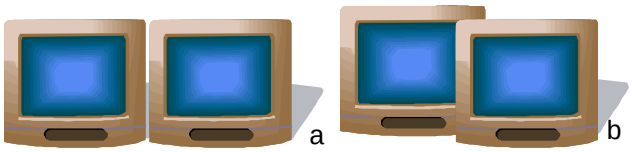

Figure 3: Combining two coplanar display units, such that they are in the same plane (a) or such that the perceived gap is minimal (b).

The potential of the in-front setup shown in Figure 3b is less obvious than of the alongside setup. While the in-front setup looks awkward for two *monitors*, it becomes useful for other types of displays. To minimize the gap and the depth distance, three dimensions have to be minimized, i.e. thickness of the display in front, border width of the display in front, and border depth of the display behind.

For two coplanar displays there exist implementations that fulfill these requirements. This combination is depicted in Figure 4. The shown setup combines a flat screen monitor in the center with a customized projection surface surrounding it. The basic idea behind this setup is that the border of the flat screen monitor is covered with the projection surface, which in turn is thin enough to keep the two display planes very close. Our f+c screen prototype is implemented based on this design, but in order to make the prototype more space-efficient, we used an LCD screen as the focus display.

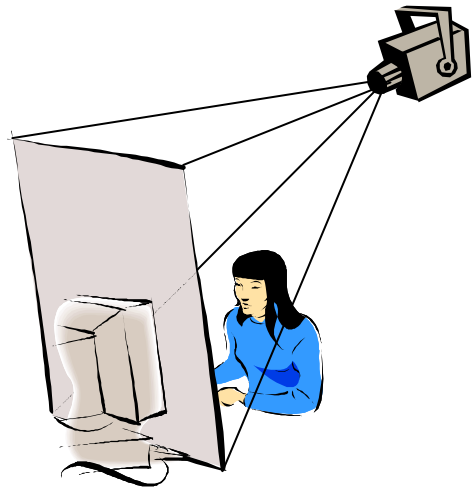

Figure 4: F+c prototype combining a monitor having a flat surface with a projection system

The pixels displayed by the video projector and the LCD are of substantially different sizes, which allows this combination to achieve high-resolution in the center, while simultaneously offering a large screen surface (see Figure 2).

Setting up an f+c screen installation

The setup shown in Figure 4 requires only moderate modification of a regular office workplace and can be built with comparably inexpensive off-the-shelf components. Figure 5 shows how this is accomplished.

Prior to the modification, the office shown contains a regular PC workplace with a Windows PC and an SGI 1600SW flat panel monitor (Figure 5a). The flat panel stays in place and becomes the focus display of our focus plus context installation. To bring the display planes closer together, the flat panel's protruding front cover is removed, making the LCD completely flat.

Next, the customized projection screen, which consists mainly a 3x4 foot (90x120cm) piece of white foam core, is added to our setup (Figure 5b). A hole large enough to accommodate the entire flat panel display allows the flat panel to be embedded within the projection screen. The surfaces of the flat panel display and the projection screen are aligned in the same plane. The installation shown uses an antique golden frame to hold the canvas, which not only allows the projection screen to stand on the desktop while leaning against the wall, but also gives the installation a stylish look. To cover the gap between the two display areas, a paper mask of appropriate size is used to extend the projection surface across the borders of the focus display, thereby creating a seamless display area (Figure 5d).

The projector (initially a portable Sony VPL-XC50, later a NEC MT 1035) fits conveniently in the space behind the user (Figure 5c), which makes this installation very space-efficient. The 8½ feet (approx. 2½ meters) wide office provides enough projection throw to make the projection fill the entire 12 square foot projection screen. Generally, projectors have to be positioned above the user's head to keep the user from casting a shadow on the projection screen (see also Figure 4). One way of accomplishing that is by mounting the projector on the ceiling. In the shown setup, the projector is placed on a shelf on the opposite side of the office (Figure 5c). To avoid keystoning, the projection surface is tilted slightly.

In the configuration described so far, the projector not only projects on the projection screen, but also on the flat panel monitor. While this overlap is key to achieving the desired zero-gap integration of the two display areas, it results in a double image and reflections on the flat panel display. This effect is avoided by placing a black object over the respective part of the projection. For this purpose, a simple program that creates a resizable window is used. After moving the window to the desired position, all window decorations can be removed by hitting the

window's *freeze* button, which leaves the window entirely black. This completes the display installation setup.

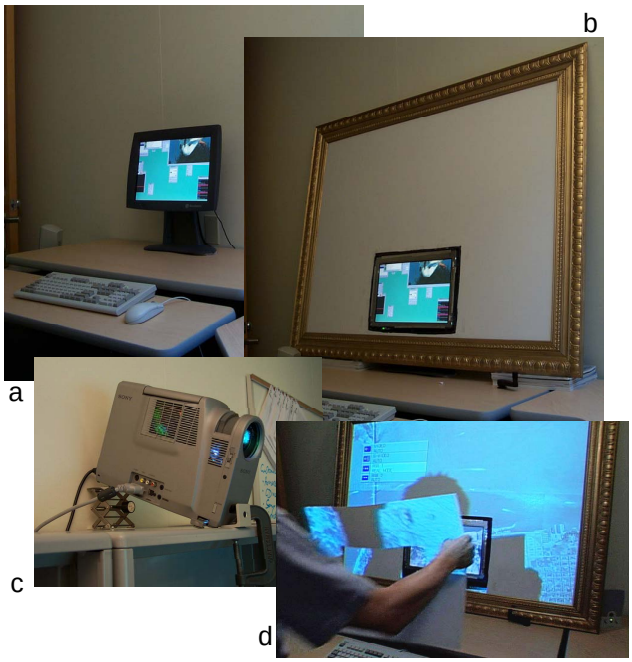

Figure 5: (a) The workplace prior to the modification. (b) The foam core projection screen is placed around the flat panel display. (c) A projector is positioned at the opposite side of the office. (d) A paper mask is added to cover the frame of the flat panel display.

This resulting setup offers a 3x4 feet (90x120cm) display area with a seamlessly integrated hi-res region. The focus display offers 1600x1024 fine pixels, while the context region provides 1024x768 (minus the removed overlap region) coarse pixels. In this particular installation, the resolution in the focus region is 5.1 times bigger than pixels on the focus display, so that each context pixel corresponds to about 26 focus pixels. A large hi-res display with the same surface and the resolution of the focus region throughout the whole display area would have around 20.5 mega pixels.

APPLICATIONS AND TASKS ON F+C SCREENS

Provided with this display, the next step is to adapt applications to it. Before we can understand what applications benefit from a focus plus context screen, we have to understand what the strengths and limitations of this novel display type are. We will analyze f+c screens in comparison to other visualization techniques designed to minimize the need for zooming interactions. Figure 6 shows how f+c screens relate to the visualization techniques mentioned earlier.

Focus plus context screens combine the advantages of overview plus detail *and* fisheyes views. Specifically, f+c screens provide users with the physical continuity of the fisheye *and* the non-distortedness of overview plus detail. The price is that users lose the possibility to zoom and pan the focus independently of the context. The focus region and the context region have become one and the zoom factor between them (e.g. the 1:5.1 ratio in the case of our installation) is now fixed. Besides that, there is the obvious need for additional space for the projection surface and the projector. This limits the applicability of f+c screens to stationary setups and excludes, for example, portable computers and palmtops.

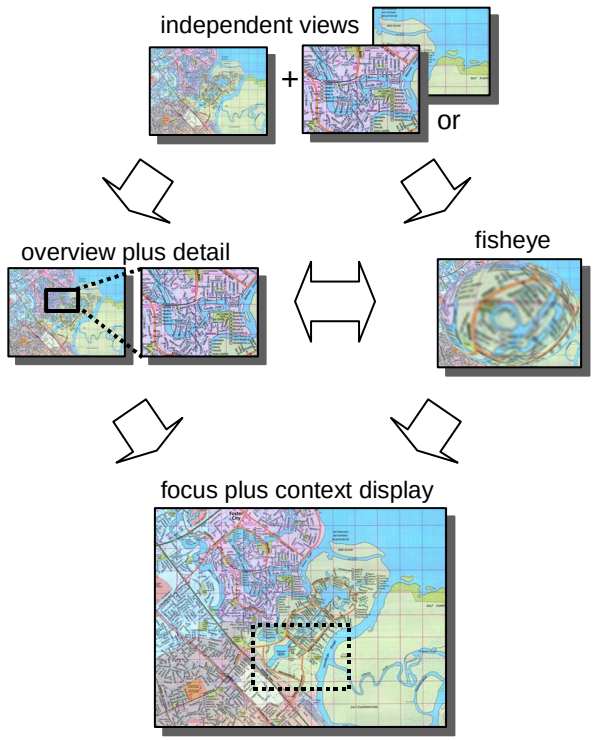

Figure 6: F+c screens compared to other visualization techniques designed for saving zooming interactions.

In order to exploit the benefits of f+c screens, we should apply them to tasks where users switch between the focus region and the context region frequently (because this is where overview plus detail requires users to carry out additional visual navigation) and where the absence of distortion is crucial (because this is where fisheyes do not work properly). How often users switch between focus and context depends on the task. This will be discussed later with the applications we actually implemented. Sensitivity to distortion varies from task to task as well. If the task requires users to compare the lengths of distances or the sizes of surfaces across image regions, distortion makes this task difficult. Additionally, images of real-world objects become difficult to recognize when distorted.

Many types of content fit this model, including representations of cities (street maps and satellite photos), buildings (architectural blueprints, renderings, CAD), integrated circuits and other technical material (drawings or photos), desktops (real and GUI desktops), biological

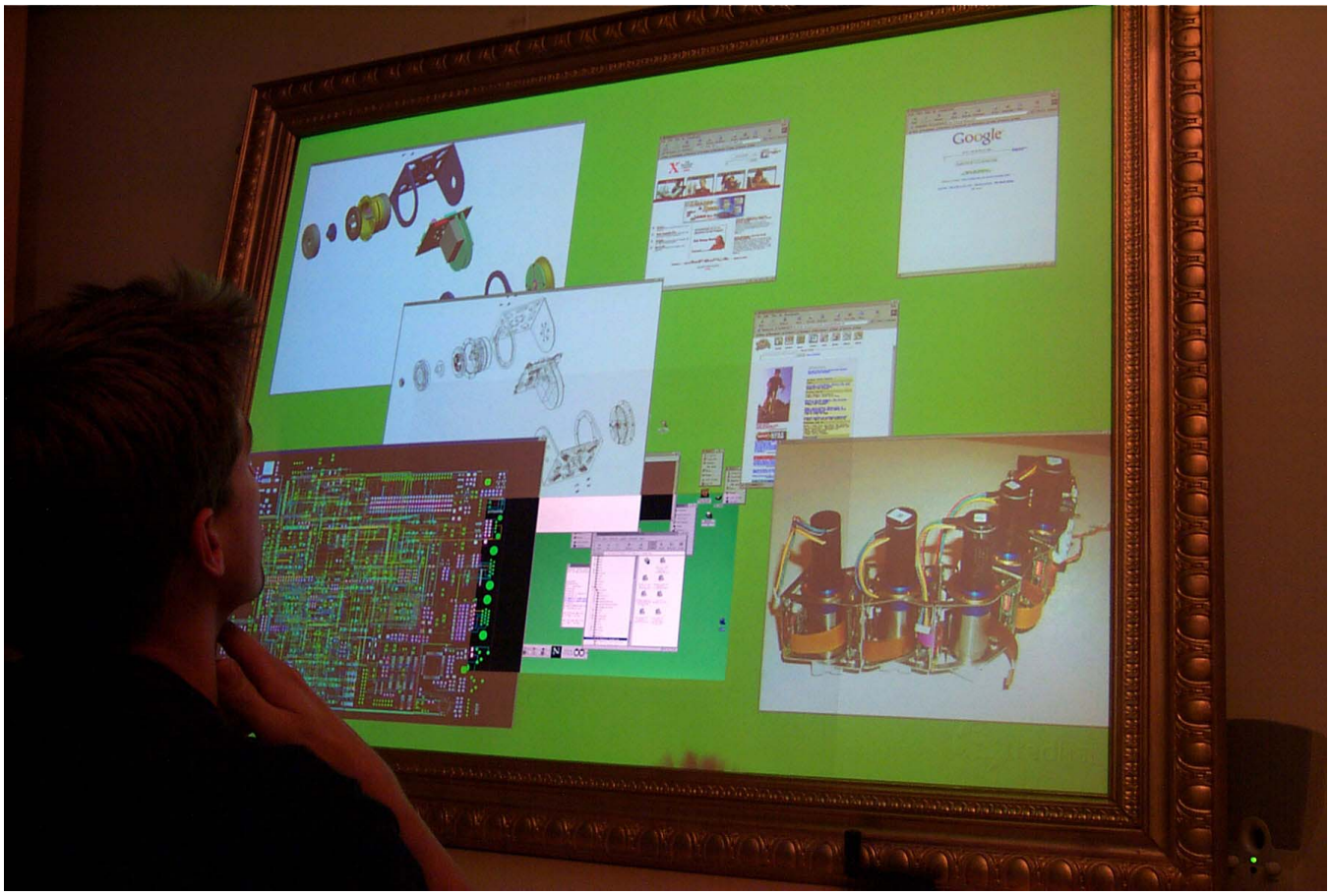

Figure 7: Working with large images and drawings on an f+c screen under Linux.

data (human skin, X-ray images, anatomy, microscopy), star and weather maps, large layouts, designs, and pieces of art (posters, paintings, including zoomable documents [1]).

Note that all these objects can come in very different representations. They can be shown in two dimensions (drawings) or three (renderings), they can be captured optically (photographs) or modeled using a computer (renderings), they can be encoded in a pixel-based (bitmap) or in a vector-based format (drawing), and they can be static (photos) or dynamic (videos, animations). The system displaying them can allow users to browse them in two (drawing program) or three dimensions (VRML editor); they may allow editing (image editor) or not (image viewer).

Applications we implemented

We found the following concrete applications most interesting and therefore implemented f+c screen solutions for them.

Scenario 1 (Editing print products) As mentioned in the introduction, print typically has a much higher resolution than computer displays. To obtain a print product that looks perfect from a distance as well as with a magnifying glass in hand, editors have to work with the print product in many different zoom levels. On an f+c screen (Figure 7), editors of print products can work on details while constantly being aware of each region's context and how detail modifications affect the overall impression. We implemented this scenario on the Linux operating system, using the available image editing and layout applications. As a side effect, the system provides users with large physical space for spreading out information, which Henderson and Card [9] found to be important.

Scenario 2 (Finding the shortest path on a map) When browsing large images, users zoom in and out to inspect objects of different scales, to compare distant objects, and to gain orientation. If the user's task is to find the shortest path from a residential address in one city to a residential address in another city, users need to zoom in to be able to read street addresses, recognize one-way streets, etc. They also need to zoom out to get an overview of the highways connecting the two cities. On an f+c screen (Figure 2), this navigation is simplified because users can constantly see a large region of the map, while simultaneously having access to street-level information in the focus display. Since f+c screens preserve geometry, comparison of distances is straightforward, even across display borders.

Scenario 3 (Videoconferencing and teleteaching) There are many situations where a video presentation simultaneously involves objects of incompatible scales. In our demo scenario shown in Figure 8, a person describes a small robot module she is holding. While it would be difficult to convey the speaker's gestures as well as a detailed view of the robot using the limited resolution of a single TV camera, f+c screens allow all this information to be displayed. On an f+c screen, viewers can simultaneously see the speaker and a detailed view of the object as well as gestures connecting them. An overview plus detail solution involving a separate "document" camera for the object would cause the relation between the speaker's gestures and the presented object to be lost. As a side effect, the large screen of our f+c installation allows the presenter and the objects to be seen at their actual size, which helps the viewer understand the scale of the presented content.

Figure 8: A videoconference partner displayed on an f+c screen. The higher resolution in the focus region allows communicating relevant details.

Scenario 4 (Simulation games) Games that immerse the user in virtual worlds have a single focus of attention. The position of the user's persona in the virtual world determines which objects or game characters are visible, accessible, and potentially dangerous. At the same time, these games often require the user to make decisions that require knowledge of the world around them. In sports games, users have to be aware of the position of other players on the field; in real-time strategy games, users continuously make decisions based on the opponent's activities on a large battlefield. Similar focus plus context effects occur in 3D simulation games, such as the first person shooter Unreal Tournament (http://www.unrealtournament.com), shown in Figure 9. Users can pick up or shoot objects only when they are in the crosshair section in the middle of the screen. The crosshair never moves, so instead of moving their eyes to objects of interest, users continuously pan objects of interest into the crosshair region. This model causes the user to continuously fixate on the screen center. The f+c screen provides a high-resolution picture in the region surrounding the crosshair, while providing a much larger peripheral image in the context area. The fact that the context area is low-resolution does not affect the user's experience, because human vision in the peripheral regions is also limited to low resolution [18].

For all four scenarios, there are systems that employ overview plus detail techniques. However, since these tasks require users to switch between views frequently, we expect f+c screens to be able to boost user's performance. Also, in all four scenarios, geometry preservation is important, which renders distorting techniques such as fisheye views inappropriate.

Figure 9: F+c screens allow players of 3D games (Unreal Tournament) to perceive their surrounding through peripheral vision.

How we implemented these applications

The Gnome Desktop, running on a *Virtual Network Computing* (VNC) XWindows display (http://www.uk.research.att.com/vnc) is shown in Figure 10. For this setup, a VNC server runs on a remote Linux machine to create a 5228x3921 pixel frame buffer, which is the resolution that the display would offer if it were all hi-res. The two physical displays of the f+c screen are connected to a dual-headed Windows PC. This PC runs two instances of VNC viewer that transfer the content from the VNC server over the network. The context display uses the "scale" option (a feature currently only available in the Windows version of VNCviewer) to scale the frame buffer down by 5.1 (= 97/19 ratio), representing the size ratio between focus and context pixels. Since VNC scales by averaging pixel colors (filtering), the image information is preserved as well as is possible. The resulting 1024x768 pixel-sized window holds the entire virtual desktop and is fully interactive. Dragging this VNC window into the projector display increases the size of its pixels, which compensates for the scaling process. The focus display uses a VNC viewer as well, but without the scaling. The virtual frame buffer is bigger than the physi-

cal display, so the VNC viewer provides scrollbars to pan around the image. Using the scrollbars, the focus display is panned until focus and context images line up. Both VNC viewers are now switched to full screen mode, so that the Linux desktop fills the display.

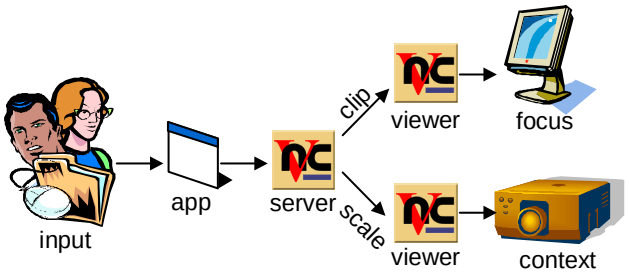

Figure 10: Running Linux on F+c screens via VNC

This setup is fully functional, i.e. it allows running arbitrary XWindows applications. Windows can be dragged around seamlessly across display regions. We successfully ran several applications including Star Office (office productivity application), gimp (image processor), Netscape (web browser), and several Linux tools. Due to several optimization techniques, such as selective refresh, VNC updates window content at a reasonable speed and small windows can be moved in real time. Redrawing a full screen window, however, can require up to a couple of seconds. We are working on improving this by creating a single-machine version. We have also started experimenting with implementations based on a single graphics card that could run both displays. Nonetheless, the fact that the virtual frame buffer is about ten times bigger than a normal PC display limits the achievable speed when applications are stretched to fill most of the screen.

In situations involving panning a full-screen image, a full-resolution bitmap would involve an unwieldy amount of memory. An efficient solution might involve creating different views of the content suited to the individual resolutions and area coverage for each display. If this is supported, each view can be generated directly by a separate application. The navigation on each of the views has to be coordinated in order to preserve geometry of content displayed across display borders. We will use the term *coupled views* to refer to this type of setup. The three f+c implementations shown in Figure 2, 8, and 9 are based on coupled views, which allows them to run at the same speed as they would on a normal PC display.

There are several different ways of obtaining coupled views, e.g. by using applications that allow a single document to be viewed in multiple windows, such as Adobe Photoshop or the Microsoft Office programs. The f+c scenario shown in Figure 2 is based on two image viewers running on different networked machines. Figure 11 shows how this works. The image viewer (ACDsee, http://www.acdsystems.com) uses a "nearest neighbor" approach to zoom images, which would introduce undesired image artifacts. Additionally, the source image for the context display would be larger than necessary. To obtain high-quality output, source images are scaled offline using Adobe Photoshop. A full-resolution focus version and a scaled-down context version of the image are saved to disk. An instance of ACDsee is run on each networked PC; one of them drives the focus display, while the other one drives the context screen. The images are aligned manually and the viewers are switched to full-screen mode.

To allow users to pan within the images, the views must now be coupled. To preserve the image's geometry, both images have to be panned in parallel, but at different speeds. To achieve this, the input from the user is transmitted to both viewer instances and is scaled corresponding to the scaling factor of the bitmaps. In our installation, for example, a 1-pixel pan in the context display accompanies a 5 to 6 pixel pan in the focus image. To accomplish forking and scaling of input events, we wrote the software tool mouseFork. MouseFork receives mouse events from the mouse/trackball device, duplicates them, scales them, and sends them across the net to each display application. MouseFork also replaces the hand tool provided by the individual image viewers with a navigation method that is more convenient for navigating in large images. Panning across large distances with a hand tool requires users to grab the plane (mouse down), pan it (mouse move), release it (mouse up), and to go back (mouse move), to avoid running into the limits of the client screen. MouseFork converts the single stream of mouse move events generated by the trackball into a series of such hand tool operations, so that the user can operate the system by simply rolling the trackball.

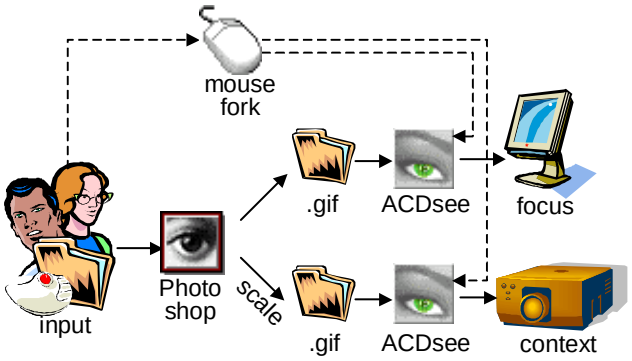

Figure 11: Running an image view at full speed

We used this viewer setup to explore several large images such as a 13,800x12,000 pixel satellite image of San Francisco, a 15,000x15,000 pixel map of London, a 10,000x 30,000 pixel Mandelbrot images, as well as a series of technical drawings, renderings, and circuit layouts. The viewer runs smoothly and there is no perceivable lag between the two display units.

The teleconferencing scenario shown in Figure 8 is still in an experimental stage. It uses forked views as well, which makes it similar to the image viewer setup, but since the user has no means to navigate the shown image, there is no transmission and forking of navigation events. Synchronization between views is achieved by using two piggybacked cameras to perform capture, i.e. the coupling between the two views (panning only, so far) is done mechanically. Note that it is also possible to implement the entire teleconferencing/teleteaching setup entirely with analog technology, i.e. by connecting one analog camera to an analog focus monitor and another one to the analog input of the projector.

The last of our four scenarios, the 3D game shown in Figure 9 is implemented using coupled views, but in this case, the coupling is supported by the game itself. The setup is done in three steps. First, the game allows players to automatically follow another player in a networked game using "spectator mode" and to look through this player's virtual eyes using the command "behindview 0". Using this feature, the view shown on the context display can be synchronized with the view on the focus display. Second, views are calibrated so that their centers (marked with a crosshair) are aligned. This is done by running the game on the projector machine in window mode (instead of fullscreen mode), which allows the window to be moved around until its crosshairs meets the crosshair of the focus display. Third, in order to calibrate scaling, the computer running the context display is given a wider view, by setting its "field of view" variable to a larger value. This completes the setup. While it is possible to run this game on two machines, we used a three-machine setup (one machine for running the game and two "spectators" to generate the views) to better synchronize views. The necessity for this adaptation did not emerge from network lag, but from the fact that Unreal sends update events to spectators only when the player's movements have exceeded a certain tolerance. Using two spectators applies the same lag to both displays and thereby gets them synchronized. The game runs at full speed and is fully playable.

EARLY RESULTS

Our prototype is currently set up in the personal office of one of the authors at Xerox PARC, which allows us to continuously experiment with the system, to demo it frequently, and to let other researchers try it out. Over the past six weeks, we demonstrated the system to about seventy of our colleagues; at least fifteen of them tried it out themselves. We let them experiment with the Linux setup and several applications on top of it (including Star Office, the Gimp, Netscape, etc.), as well as with the image viewer (allowing them to browse a satellite image of San Francisco, a London map, and a fractal). Some of our colleagues also tried out our adaptation of Unreal Tournament. Listed below are some of the impressions of several of our colleagues who used the f+c screen.

The Linux implementation was the first one that we had up and running. It received a lot of feedback and inspired many great suggestions, including the three setups that we later implemented using coupled views. Its support for working with large documents was widely appreciated. Despite the fact that our f+c screen actually displays fewer physical pixels than the two-headed SGI LCD setups that some of our colleagues use, the display was generally perceived as providing "lots of space". The large screen was judged especially useful in combination with the high panning speed of the image viewer, which received a great response especially with the San Francisco satellite image. We adapted the 3D game only very recently, but the few people who tried it described the additional resolution in the focus region as beneficial.

Practically all our testers immediately reflected on how an f+c screen would affect their daily work at PARC. Their feedback varied based on the documents and tasks their daily work involves. The most enthusiastic feedback came from people in our media group (MARS) and in the Research in Experimental Design group (RED). Both groups work with large visual objects, such as posters, design sketches, collections of photographs, etc. Hardware designers also appreciated the display's capability of showing large construction drawings. Two of our colleagues expressed interest in using an f+c screen for managing large websites or network plans. On the other hand, the display generated only limited interest among those users who primarily work with text, especially in program code editing tasks. This feedback is not surprising, when taking into account that regular-sized text becomes unreadable when moved to the context region. Users working with text with an emphasis on layout, such as the people in the media group, however, judged f+c screens as a desirable enhancement of their work environment.

FUTURE WORK AND CONCLUSIONS

In the future, we plan to work in three major directions. We are currently setting up an experiment comparing f+c screens with overview plus detail views with respect to a chip design task. We also plan to experiment with f+c screens in multi-user applications, such as walls or tables with multiple embedded focus displays. Furthermore, we planning improvement to the applicability of f+c screens to text-based applications by applying selected visualization techniques, such as Thumbnails [24].

While large hi-res displays match all the usability characteristics of f+c screens plus offering high resolution throughout the entire screen surface, f+c screens are more feasible. They are more than an order of magnitude less expensive than a comparable 20-megapixel display based on tiled projections. Equally important, f+c screens require substantially less space, which allows them to be set up in normal offices settings. Today, two-headed systems are in common use, but as dropping prices currently

bring projectors to the mass market [14], focus plus context screens offer an alternative for users working with visual content, such as designers or architects.

In this paper, we presented a new means for supporting users working with large visual content. By combining visualization techniques with a new type of display setup, f+c screens achieve characteristics expected to outperform existing visualization techniques. While overview plus detail visualizations require users to switch between multiple views, focus plus context screens allow users to *simultaneously* keep track of context information via peripheral vision. Since f+c screens do not distort display content, they are applicable to situations where fisheye views cannot be used.

ACKNOWLEDGMENTS

We'd like to thank the members of RED, especially Mark Chow and Rich Gold, as well as Dale MacDonald who provided us with the wonderful golden frame. We'd also like to acknowledge Brian Tramontana for his efforts in setting up the videoconferencing scenario, Jeff Breidenbach for his Unreal hack, Bill Janssen for the initial idea of using VNC, Mark Stefik and Nola Mae McBain for their valuable comments, and Mark Yim and Dave Duff for providing the modular robot components and the renderings that we used in our demo scenarios.

REFERENCES

1. Bederson, B. B., and Hollan, J.D. Pad++: A zooming graphical interface for exploring alternate interface physics. *Proceedings of UIST' 94*, pp. 17-26. New York: ACM.
2. Butz, A., Hollerer, T., Feiner, S., MacIntyre, B., and Beshers, C. Enveloping users and computers in a collaborative 3D augmented reality, *International Workshop on Augmented Reality 1999*, San Francisco California, October 20-21, 1999.
3. Carpendale, S. A. *Framework for Elastic Presentation Space*, Ph.D. thesis, March 1999, School of Computing Science at Simon Fraser University.
4. Cruz-Neira, C., Sandin, D.J., DeFanti, T.A., Kenyon, R.V., and Hart, J.C. The CAVE: audio visual experience automatic virtual environment. *Commun. ACM* 35(6): 64–72, June 1992.
5. Feiner, S. and Shamash, A. Hybrid user interfaces: breeding virtually bigger interfaces for physically smaller computers. *Proc. of UIST' 91*. pp. 9–17, 1991.
6. Funkhouser, T., and Li, K. On the wall large displays. *IEEE Computer Graphics & Applications*, 20(4), July/August 2000.
7. Furnas, G. Generalized fisheye views. *Proceedings CHI 1986*, pp. 16–23, 1986. New York, ACM, 1986.
8. Grudin, J. Partitioning Digital Worlds: Focal and peripheral awareness in multiple monitor use. *Proceedings CHI 2001*, pp. 458-465.
9. Henderson, D. A., and Card, S. Rooms: The use of multiple virtual workspaces to reduce space of contention in a window-based graphical user interface. *ACM Transactions on Graphics*, 5(3):211-243, 1986.
10. Hereld, M., Judson, I., Stevens, R., Introduction to Building Projection-based Tiled Display Systems *IEEE Computer Graphics & Applications* Vol. 20, No. 4, July/August 2000
11. Hinckley, K., Czerwinski, M., and Sinclair, M. Interaction and modeling techniques for desktop two-handed input. *Proceedings of UIST '98*, pp. 49–58, 1998.
12. Holmquist, L. E. Focus + context visualization with flip zooming and the zoom browser. In *CHI '97 Extended Abstracts* (Atlanta GA, Mar 1997) ACM Press, 263-264
13. Hornbaek, K., Frokjaer, E. Reading of electronic documents: the usability of linear, fisheye, and overview+detail interfaces. *Proceedings of CHI '01*, pp. 293-300, 2001.
14. Lieberman, D. Projectors posed as focal point of networked home. *EE Times* 03/22/01. Available online at http://www.eetimes.com/story/OEG20010321S0055
15. Menendez R. G., and Bernard J.E. Flight simulation in synthetic environments. *Proc. 19th Digital Avionics System Conference*, pp. 2A5/1-6 vol.1, 2000.
16. Olston C., and Woodruff. A. Getting Portals to Behave. *Proc. Information Visualization. 2000*, Salt Lake City, Oct. 2000, pp. 15-25.
17. Plaisant, C., Carr, D., Shneiderman, B., Imagebrowsers: Taxonomy and design guidelines, *IEEE Software*, 12,2 (march 1995), 21-32
18. Rayner K., Pollatsek A. Eye movement and scene perception. *Canadian Journal of Psychology*, 46:3:342-376, 1992.
19. Rekimoto, J. and Saitoh, M. Augmented surfaces: a spatially continuous work space for hybrid computing environments. In *Proceeding of CHI '99*, pp. 378–385, 1999. New York, ACM, 1999.
20. Robertson, G., and Mackinlay, J. The Document Lens. *Proceedings of UIST '93*. 1993, pp. 101 – 108.
21. Shneiderman, B. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Third edition. Reading MA: Addison-Wesley, 1998.
22. Stone, M.C., Fishkin, K., and Bier, E.A. The movable filter as a user interface tool. *Proceedings of CHI '94*, 1994, pp. 306 – 312.
23. Streitz, N.A. Geißler, J., Holmer, T., Konomi, S., Müller-Tomfelde, C., Reischl, W., Rexroth, P, Seitz, P., and Steinmetz, R. I-LAND: An interactive landscape for creativity and innovation, In *Proceedings of CHI '00*, pp. 120–127, New York, ACM, 1999.
24. Woodruff, A., Faulring, A., Rosenholtz, R. Morrsion, J., and Pirolli, P. Using thumbnails to search the Web *Proceedings of CHI '01*, 2001, pp. 198 – 205.

Drag-and-Pop and Drag-and-Pick: techniques for accessing remote screen content on touch- and pen-operated systems

Patrick Baudisch[1], Edward Cutrell[1], Dan Robbins[1], Mary Czerwinski[1], Peter Tandler[2], Benjamin Bederson[3], and Alex Zierlinger[4]

[1]Microsoft Research, Redmond, WA; [2]Fraunhofer IPSI, Darmstadt, Germany; [3]HCIL, University of Maryland, MD; [4]Maila Push, Darmstadt, Germany

{baudisch, cutrell,czerwinski, dcr}@microsoft.com; tandler@ipsi.fhg.de; bederson@cs.umd.edu; alex@zierlinger.de

Abstract: Drag-and-pop and drag-and-pick are interaction techniques designed for users of pen- and touch-operated display systems. They provide users with access to screen content that would otherwise be impossible or hard to reach, e.g., because it is located behind a bezel or far away from the user. *Drag-and-pop* is an extension of traditional drag-and-drop. As the user starts dragging an icon towards some target icon, drag-and-pop responds by temporarily moving potential target icons towards the user's current cursor location, thereby allowing the user to interact with these icons using comparably small hand movements. *Drag-and-Pick* extends the drag-and-pop interaction style such that it allows activating icons, e.g., to open folders or launch applications. In this paper, we report the results of a user study comparing drag-and-pop with traditional drag-and-drop on a 15' (4.50m) wide interactive display wall. Participants where able to file icons up to 3.7 times faster when using the drag-and-pop interface.

Keywords: Drag-and-drop, drag-and-pick, interaction technique, pen input, touchscreen, heterogeneous display.

1 Introduction

With the emergence of pen- and touch-operated personal digital assistants (PDAs), tablet computers, and wall-size displays (e.g., Liveboard, Elrod et al., 1992; Smartboard, http://www.smarttech.com), touch and pen input have gained popularity. Over the past years, more complex display systems have been created by combining multiple such display units. Wall-size touch displays have been combined into display walls, such as the DynaWall (Streitz 2001), or the iRoom Smartboard wall (Johanson, 2002b). Recent PDAs and tablet computers allow connecting additional displays, such as another tablet or a monitor in order to extend the device's internal display space.

Touch/pen-operated screens that consist of multiple display units bring up a new class of input challenges that cannot always be solved with existing techniques, because many of the existing techniques were designed for indirect input devices, such as mice, track pads, or joysticks. Indirect input devices can be used on arbitrary display configurations, because they can simply be mapped to the respective topology (e.g., PointRight, Johanson 2002a). Touch/pen input, however, is based on the immediate correspondence between input space and display space and thus requires users to adapt their input behavior to the physicality of the display system. Here are three examples where this can become problematic.

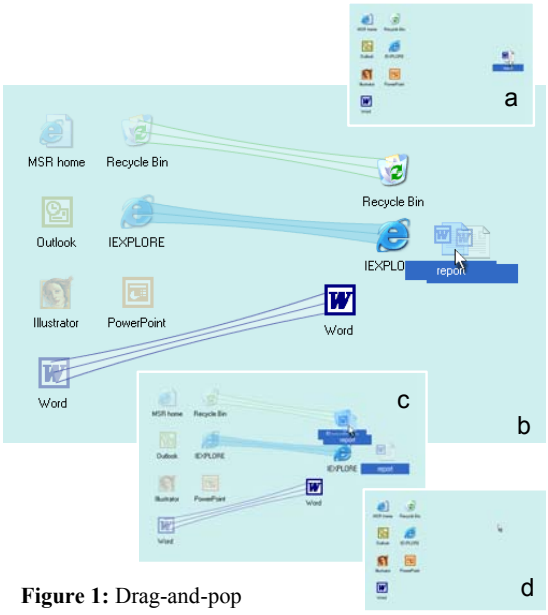

Figure 1: Drag-and-pop

Scenario 1: External monitors. One or more display units within a display system may not be equipped with a touch or pen sensor. Connecting an external monitor to a tablet computer or PDA, for example, allows users to see more material, but requires them to use an indirect input device, such as a mouse, when interacting with content on the external monitor. Since some tablet-specific tasks, such as scribbling, are hard to accomplish with a mouse, users find themselves continuously switching between pen and mouse.

Scenario 2: Interactions across display units. Some interaction techniques, such as drag-and-drop, require users to interact with two or more icons in a single pen-down interaction. If these icons are distributed across physically separate pen/touch input display units, users first have to bring all involved icons to the same display unit, a potentially time-consuming activity (Figure 2a-c).

Scenario 3: Bridging long distances. Accessing icons located far away from the user, e.g., on the opposite side of a 15' DynaWall, requires users to physically walk over, the time for which may in some circumstances increase linearly with distance (Guiard et at, 2001). In addition, *drag* interactions get more error-prone with distance, because users drop objects accidentally when failing to continuously keep the pen tip in contact with the display surface (Rekimoto 1997).

2 Drag-and-pop & drag-and-pick

Drag-and-pop and *drag-and-pick* are interaction techniques that address these issues. We will begin by giving an overview; more detailed descriptions of both techniques can be found in Section 4.

Drag-and-pop extends traditional drag-and-drop as illustrated by Figure 1. (a) The user intends to delete a Word memo by dragging it into the recycle bin. (b) As the user starts dragging the memo's icon towards the recycle bin, icons that are of compatible type and located in the direction of the user's drag motion "pop up". This means that for each of these icons a link icon is created (*tip icon)* that appears in front of the user's cursor. Tip icons are connected to the original icon (*base icon*) using a rubber band. (c) The user drags the memo over the recycle bin and releases the mouse button. The recycle bin accepts the memo. Alternatively, the user could have dropped the memo over the word processor or the web browser icon, which would have launched the respective application with the memo. (d) When the user drops the icon, all tip icons disappear instantly.

Figure 2d shows how drag-and-pop simplifies dropping icons onto targets located at the other side of a bezel that separates display units (scenario 2). Figure 9 shows a user performing a drag-and-pop interaction to drop an icon on a distant target.

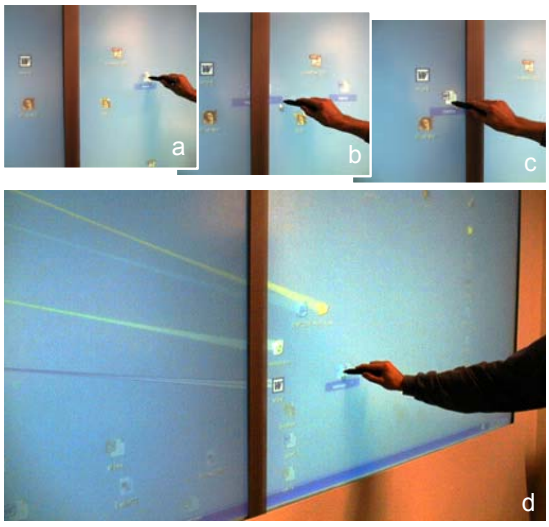

Figure 2: (a-c) Traditional drag-and-drop: Dragging an icon across the bezel requires the user to drop the icon half way across the bezel and pick it up at the other side (d) Drag-and-pop temporarily brings matching target icons to the current pen location, allowing the user to file icons without having to cross the bezel.

Drag-and-pick modifies the drag-and-pop interaction concept such that it allows activating icons, e.g., to open a folder or to launch a program. While drag-and-pop is initiated by the user dragging an icon, drag-and-pick starts with the user performing a drag interaction on empty screen space. The system's response to this drag interaction is similar to drag-and-pop, but with two differences. First, *all* icons located in the direction of the drag motion will pop up, not only those of compatible type (Figure 3). Second, as the user drags the mouse cursor over one of the targets and releases the mouse button, the folder, file, or application associated with the icon is activated as if it had been double clicked.

Figure 4 shows how this allows users to use the pen for launching an application, the icon of which is located on a monitor not supporting pen input.

In principle, drag-and-pick can be applied to any type of widget, e.g., any buttons and menus located on a non-pen accessible monitor. In this paper, however, we will focus on the manipulation of icons.

3 Related work

Drag-and-drop is a well-know interaction technique for transferring or copying information using a pointing device, while avoiding the use of a hidden

clipboard (Wagner, 1995; Beaudouin-Lafon, 2000). Hyperdragging (Rekimoto, 1999), allows extending drag-and-drop across physically separate displays (Scenario 2), but requires an indirect input device, such as a mouse. Most techniques compatible with pen usage are based on point-and-click, e.g., pick-and-drop (Rekimoto, 1997) and take-and-put (Streitz et al., 2001). These techniques, however, cannot be used to access material on a display unit not providing pen support (Scenario 1).

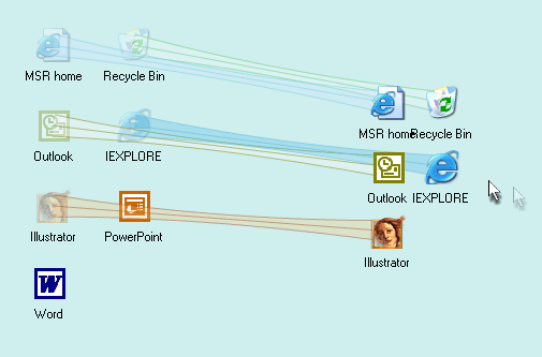

Figure 3: Drag-and-pick makes *all* icons in the direction of the mouse motion come to the cursor.

Figure 4: Drag-and-pick allows users to temporarily move icons from an external monitor to the tablet where the user can interact with them using the pen.

A different set of interaction techniques have been proposed to help users overcome large distances (Scenario 3). Manual And Gaze Input Cascaded (MAGIC) pointing (Zhai et al., 1999) uses eye tracking to move the cursor to the target area, from where the user guides the cursor manually (which requires an indirect input device). Gesture input techniques allow selecting a target and a command in a single interaction and are generally compatible with pen input (Rubine, 1991). 'Throwing' allows users to accelerate an object with a small gesture; the object then continues its trajectory based on its inertia (Geißler, 1998). The imprecision of human motor skills has prevented throwing from being used for reliable target acquisition. Myers et al. (2002) used laser pointers to acquire targets on a Smartboard, but found them to be slower than touch input.

A variety of mouse-based interaction techniques use destination prediction to simplify navigation (e.g., Jul, 2002). Dulberg et al. (1999) proposed a flying click or flick for snapping the mouse to target locations. Swaminathan and Sato (1997) proposed making relevant controls on the screen "sticky".

As an alternative way of launching applications, today's operating systems offer menus containing lists of available application or documents. A 'send to' option (Microsoft Windows) allows sending an icon to a target selected from a predefined list. Compared to 2D desktops, which typically use a larger amount of screen space than pull-down or pop-up menus, menus are limited to a smaller selection of choices unless they use a hierarchical menu organization, which makes their usage less transparent and often less efficient. Furthermore, invoking a content-menu may require hitting a qualifier key, which can be problematic on touch-based systems.

4 Design and algorithms

In this section, we will take a more detailed look at the design and algorithms behind drag-and-pop/pick.

4.1 Selecting candidates

In order to reduce clutter, drag-and-pop creates tip icons only for a subset of the icons on the screen. Drag-and-pop's candidate selection algorithm is initialized with the entire set of icons on the screen; it then successively eliminates candidates using the following four rules.

First, icons of incompatible type are eliminated. If the user drags a text file, the icon of a text processor can create a tip icon; the recycle bin icon can create a tip icon; the icon of another text file, however, cannot, because dragging two text files onto each other is usually not associated with any behavior. Drag-and-pick bypasses this selection step in order to allow users to activate any type of icon.

Second, icons located between the cursor and the location where the tip icons cluster will appear (see following section) are eliminated. This rule avoids creating tip icons that move away from the cursor.

Third, only icons that are located within a certain angle from the initial drag direction (the *target sector*) are considered. The initial drag direction is determined the moment the user drags an icon further

than a given threshold (default 15 pixels). During preliminary testing on a Smartboard, we got good results with first-time users when using sector sizes of ±30 to ±45 degrees. The sector size could be reduced to sector sizes of ±20 degrees as users gained more experience.

Forth, if the number of qualifying icons is above some hard limit, drag-and-pop eliminates tip icon candidates until the hard limit is met. Icons are removed in an order starting at the outside of the target sector moving inwards. This rule assures the scalability of drag-and-pop to densely populated displays, but requires drag-and-pop users working with densely populated screens to aim more precisely. We typically use hard limits between 5 and 10.

4.2 Computing the tip icon layout

Once tip icon candidates have been selected, drag-and-pop determines where on the screen to place the tip icons. In order to avoid interference between tip icons, the location of all tip icons is computed in a centralized fashion.

Our drag-and-pop prototype uses the following algorithm that is illustrated by Figure 5: (1) Snap icons to a grid and store them in a two-dimensional array, with each array element representing one cell of the grid. If two or more icons fall into the same cell, refine the grid. (2) Shrink the icon layout by eliminating all array columns and rows that contain no icons. (3) Translate icon positions back to 2D space by mapping the array onto a regular grid. By default, the output grid is chosen to be slightly tighter than the input grid, which gives extra compression.

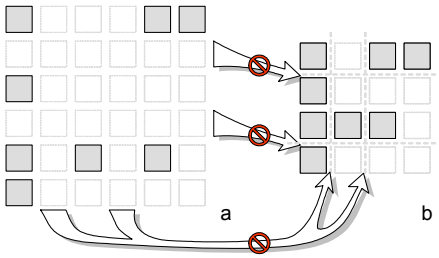

Figure 5: Drag-and-pop computes tip icon layouts (a) by snapping icons to a grid and then (b) removing empty rows and columns.

We chose this algorithm, because it preserves alignment, proximity, and spatial arrangement between icons, which allows users to use their spatial memory when identifying the desired target within the tip icon cluster. This is especially useful when tip icons look alike (e.g., a folder in a cluster of folders). In order to help users distinguish local icon clusters from surrounding icons more easily, the algorithm may be adjusted to *shrink* empty rows and columns during layout computation instead of removing them entirely.

After the tip icon layout has been computed, drag-and-pop positions it on the screen such that the center of the layout's bounding box is located at the direct extension of the user's current mouse motion. The distance of the tip icon cluster to the user's current cursor position is configurable. For inexperienced users, we got best results with distances of around 100 pixels; shorter distances made these users likely to overshoot the cluster. For more experienced users, we were able to reduce the distance to values around 30 pixels, which allowed these users to operate drag-and-pop with less effort, in a more "menu-like" fashion. In order to reduce visual interference between tip icons and icons on the desktop, drag-and-pop diminishes desktop icons while tip icons are visible.

4.3 The rubber band

When the tip icon cluster is displayed, users need to *re*-identify their targets within the tip icon cluster in order to be able to successfully acquire them.

Our first implementation of drag-and-pop created tip icons on top of their bases and used slow-in-slow-out animation (Shneiderman 1998) to move tip icons to their final location. While this approach allowed users to locate the final position of the desired tip icon by visually tracking it on its way from basis to final position, it also required users to either wait for the animation to complete or to acquire a moving target. We therefore chose to abandon the animation and immediately display tip icons at their final destinations.

In lieu of the animation, we provided tip icons with rubber bands. The design prototype of the rubber band is shown in Figure 6. For performance reasons, our prototype, which is shown in all other screenshots, uses rubber bands of a lower level of graphical detail, i.e., a tape and three lines in the color scheme of the corresponding icon.

The purpose of the rubber band is to offer the functionality of the animation, but without the problems alluded to above. The rubber band, decorated with the respective icon's texture, can be thought of as having been created by taking a photograph of the tip icon animation with a very long shutter speed (so-called *motion blur*, e.g., Dachille and Kaufman, 2000). Like the animation, the rubber band allows users to trace the path from base to tip icon. However, users can do this at their own pace and the customized texturing of the rubber band allows users to start tracing it anywhere, not only at the base.

The rubber band is provided with a narrow mid-riff section, suggesting that the rubber band is elastic. This design was chosen to help users understand that tip icons have retracted to their bases when they disappear at the end of the interaction. This feature may also help users find their way to the tip icon faster, because it provides users with a visual cue about how far away the tip icon is located. A thick rubber band section implies that the tip icon (or base) is close; a thin rubber band section indicates that the target is further away.

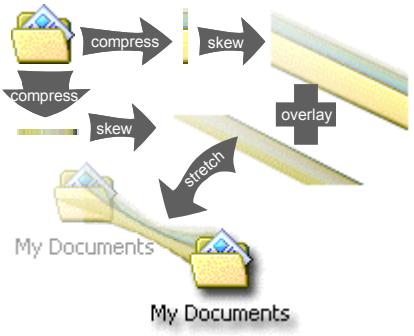

Figure 6: The motion blur textures on the rubber bands that connect tip icons with their bases are made by overlaying skewed copies of that icon.

4.4 Aborting drag-and-pop interactions

As soon as tip icons and rubber bands are shown on the screen, drag-and-pop waits for the user to acquire one of the tip icons to complete the ongoing drag-and-pop or drag-and-pick interaction. There are two cases, however, in which users will want to abort the interaction without acquiring a tip icon.

The first case is when the user dragged the mouse at a wrong angle so that the desired target icon did not pop up. In this case, the user may either drop the icon and try again or complete the interaction as a regular drag-and-drop interaction, i.e., by dropping the icon onto the target icon's base instead.

The other case occurs if the user is intending to perform a regular mouse drag operation, for example to rearrange icons on the desktop or to capture a set of icons using a lasso operation. For these cases, drag-and-pop allows users to terminate tip icons on-the-fly and to complete the interaction without drag-and-pop/pick. To abort, users have to move the mouse cursor away from the tip icon cluster while still keeping the mouse depressed. This can be done by overshooting the cluster or by changing mouse direction. In particular, this allows users to access the underlying drag-and-drop and lasso-select functionality by introducing a simple zigzag gesture into their cursor path. The zigzag contains at least one motion segment moving away from the tip icons, thus terminating tip icons as soon as they appear.

The algorithm: the tip icon cluster is kept alive as long as at least one of the following three rules is successful. The first rule checks whether the mouse cursor has moved closer to the center of at least one of the icons in the tip icon cluster. This rule makes sure that the cluster does not disappear while users approach their targets. The second rule checks if the cursor is in the direct vicinity of an icon. This rule provides tolerance against users overshooting a tip icon while acquiring it. The third and last rule keeps the cluster alive if the cursor is stationary or if it is moving backwards very slowly (up to 5 pxl/frame). This rule makes drag-and-pop insensitive to jitter. Figure 7 illustrates the resulting behavior.

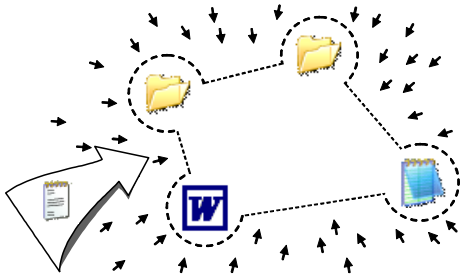

Figure 7: The tip icon cluster is kept alive as long as the user moves towards the cluster (arrows) or inside the convex hull surrounding the cluster (dashed).

5 User study

In this section, we report the results of a user study comparing drag-and-pop with the traditional drag-and-drop technique. To examine the effects of bezel-crossing as well as distance, as described in Scenarios 2 and 3, we chose to run the study on a tiled wall-size display. During the study, in which participants filed icons into folders or dragged them onto the icons of matching applications, we recorded the time and accuracy of these movements. Our main hypothesis was that participants would perform faster when using the drag-and-pop interface, primarily because it would avoid the need for crossing the bezels, but also because it would bridge the space to very distant icons more efficiently.

5.1 Desktop layout

To obtain a representative set of icon arrangements for the study, we gathered desktop screenshots from 25 coworkers who volunteered their participation (15 single, 6 dual, and 4 triple monitor users). Overall resolutions ranged from 800,000 pixels to 3,900,000 pixels (66% more than the display wall used in the experiment).

We clustered the obtained desktops by number of icons and arrangement pattern. Then we chose representatives from each of the three resulting main clusters for the study (Figure 8). The "sparse" desktop reflected the desktops of roughly two thirds of the participants. It contained only 11 icons, most of which were lined up in the top left corner of the screen. The "frame" desktop reflected the desktops of three of the participants. It contained 28 icons arranged around the top, left, and right edge of the screen. The "cluttered" desktop, finally, contained 35 icons that were spread primarily across the top and left half of the screen. Five participants had chosen this style of arranging their icons.

Icon layouts were stretched to fit the aspect ratio of the display wall used in the experiment. An area at the bottom right of the screen was reserved for the starting locations of the icons to be filed during the study (dashed shape in Figure 8).

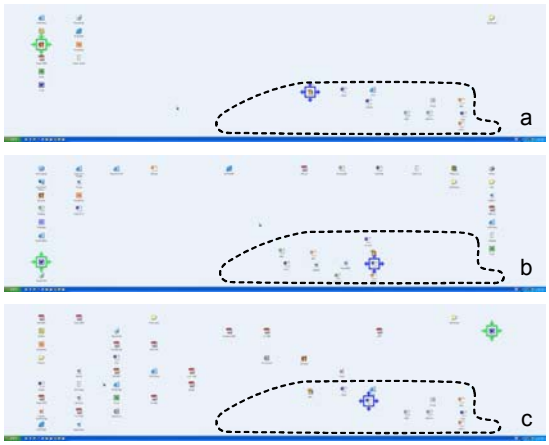

Figure 8: The (a) sparse, (b) frame, and (c) cluttered desktop layouts used in the study. The dashed line indicates the space reserved for the icons users had to file. Boxes around icons indicate icon to be filed and target.

5.2 Participants

Eight colleagues with no experience using drag-and-pop were recruited for this experiment. Due to technical problems, the data from one of these participants had to be dropped leaving us with 7. There were 2 female and 5 male participants ranging in age between 18 and 35. All were right handed with normal or corrected-to-normal vision.

5.3 Method

The test was run on the DynaWall (Streitz, 2001), a display wall consisting of three Smartboard units (Figure 9). Each Smartboard consisted of a back-projected 72"display with resistive touch input, so that the entire display was 15' (4.50m) long and 45" (1.12m) high. Display units could be operated by touching the display, but for easier handling participants were provided with color-free felt pens. Each of the three display units ran at a resolution of 1024x768 pixels, offering an overall resolution of 3072x768 pixels. The three display units were connected to a single PC equipped with two Matrox Millennium graphics cards and running WindowsXP. During the experiment, the DynaWall ran a simulated Windows desktop. We compared drag-and-pop to a control condition of drag-and-drop. Since our preliminary Windows-based version of drag-and-pop did not support the full functionality required for the study, we implemented a simulation using Macromedia Flash (www.macromedia.com). The drag-and-pop interface used in the experiment was configured to a ±30 degree target sector, 35 pixel target distance, and a maximum number of 5 tip icons.

Figure 9: DynaWall setup used in user study

To each desktop layout we added 10 document icons in the lower right quadrant of the screen. These appeared in six different arrangements (Figure 8 shows 2 of them). The participants' task was to drag these icons into a given target folder or application. Icons of image files, for example, were to be filed in a folder labeled "My Pictures" and all Word documents should be dropped onto the Word application. To counterbalance for order effects, we required participants to file the documents in a randomized order. That is, for each movement, the item to be filed was highlighted along with the target icon. All other document icons were frozen, so that participants could only move the highlighted icon. As soon as participants began moving an item, all highlighting was removed, forcing participants to remember the destination item. We did this to assure that participants would have to re-identify tip icons when using the drag-and-pop interface, just as they would in a real-world task.

Participants were allowed several minutes to practice moving and filing icons in the prototype to get them accustomed to both the DynaWall display and the drag-and-pop interface. Once it was clear that users understood how to use the display and the interfaces, they were allowed to go on to the study. Participants filed 2 sets of icons for each interface (drag-and-pop and control), for each of the three desktops. Thus participants filed 2 x 10 icons x 2 interface x 3 desktops for a total of 120 movements.

To mitigate learning effects associated with new desktop arrangements or a new interface, we omitted the first 5 trials for any desktop-interface combination from our analyses, yielding ~15 correct trials per cell or 90 movements per participant.

5.4 Results

5.4.1 Task performance

Task performance was evaluated through speed and accuracy measurements. Error rates were considerably larger for drag-and-pop than for the control (6.7% vs. 1%). We observed two things that made this type of error more likely in the drag-and-pop condition. First, in the drag-and-pop condition candidate targets were brought closer together, making it easier to accidentally drop an item on the wrong target. Second, because drag-and-pop targets had been translated away from their "home" location, participants would sometimes forget which item was in fact the target, especially if visually similar icons (e.g., other folders) had created tip icons as well.

All data analyses for movement times were performed on the median movement times for each participant in each condition to normalize the typical skewing associated with response time data. Summary statistics report the means of these times.

Target icons could be located in the same display unit as the icon to be filed, in a neighbor display unit, or in the display unit at the other end of the display wall, requiring users to cross 0, 1, or 2 bezels in order to file the icon. To test the effect of bezel crossing on performance, we ran a 2 (Condition) x 3 (Bezels Crossed) within subjects ANOVA on the median movement data. This revealed a significant main effect for condition, $F(1,6) = 18.2$, $p<0.01$. Collapsed across all distances, drag-and-pop was significantly faster than the control. There was also a significant main effect of bezels crossed, $F(2,12) = 19.5$, $p<0.01$; movement time increased as the number of bezels participants had to cross to get to the target icon increased. As hypothesized, we also saw a significant interaction between condition and number of bezels crossed, $F(2,12) = 15.2$, $p<0.01$. As seen in Figure 10, an increase in the number of crossed bezels resulted in only a small increase in movement time for drag-and-pop, whereas it had a huge effect for the control interface.

When no bezels had to be crossed, drag-and-pop appeared to be slightly slower than control, although follow-up t-tests showed that this difference was not significant, $t(6)=1.73$, ns. When 1 or 2 bezels had to be crossed, drag-and-pop was significantly faster than drag-and-drop ($t(6)=4.02$, $p<0.01$ & $t(6)=4.12$, $p<0.01$, respectively). With 1 bezel crossed, drag-and-pop was twice as fast as the control and with 2 bezels it was 3.7 times as fast.

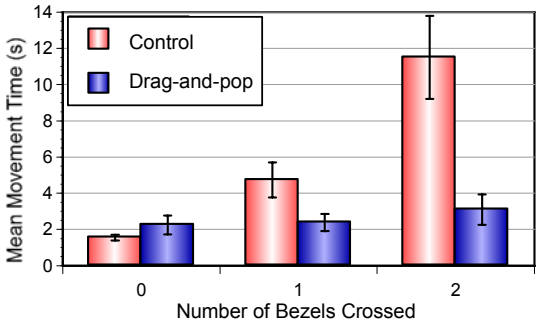

Figure 10: Mean movement time for control and drag-and-pop interfaces (± SEM).

Figure 11 shows a scatter plot of movement time versus target distance for both conditions. The best linear fit for drag-and-drop was $f(x)=0.007x-1.76$, $r^2=0.23$. The linear fit for drag-and-pop was $f(x)=4.19$, $r^2<0.0001$. This reinforces what can be seen in Figure 10—movement time increases with distance for the control interface, but stays relatively constant for the drag-and-pop interface.

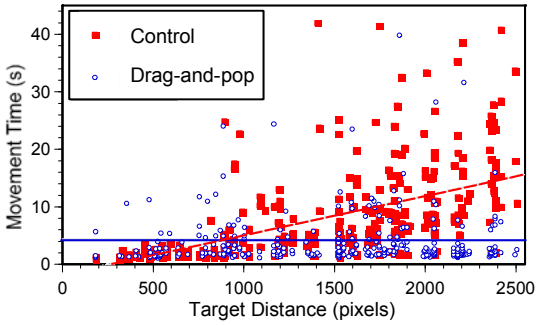

Figure 11: Movement time vs. target distance.

5.4.2 Questionnaire and subjective feedback

At the end of the study, participants answered a short questionnaire about their experience using the DynaWall and drag-and-pop. Participants were very enthusiastic about drag-and-pop. On a 7 point Likert scale (where 7=strongly agree and 1=strongly disagree), there was a mean > 6 for questions such as, "I liked using drag-and-pop", "I always understood what was happening when drag-and-pop was on,"

and "I would use drag-and-pop for large displays." There was a mean of less than 3 for "It took a long time to get used to drag-and-pop" and "It was hard to control what the targets did when drag-and-pop was on." Participants reported the drag-and-pop interface to cause less manual stress and fatigue than the control interface.

The most common problem with drag-and-pop was in getting the right group of targets to pop up, and several participants requested a wider angle for destination targets. This relates to an observation we made about how people interact with touch-sensitive wall-displays. On the wall display, participants had to employ their whole arm to make a movement, resulting in targeting motions in the shape of arcs. This means that the initial direction of the movement was *not* in the direction of the target. To accommodate such arcs in the future, we have adapted the target selection algorithm of drag-and-pop by giving the target sector extra tolerance for movements towards the top of the screen.

6 Conclusions and future work

The substantial time-savings found in the user study confirm our expectations. Although when used within a single screen unit drag-and-pop does not seem to by faster than traditional drag and drop (first pair of bars in Figure 10; drag-and-pop's capability of bridging distance to the target seems to be nullified by the need for re-orientation), its advantages on very large screens and its capability of bridging across display units are apparent. On the usability side, we were glad to see that participants had no trouble learning how to use the technique and that they described the technique as understandable and predictable. The single biggest shortcoming, the target selection, is the subjects of current work. In addition to the changes described above, we consider dropping the notion of a fixed target sector size and replace it with a mechanism that adjusts the sector size dynamically based on the number of matching targets.

Given the recent advent of commercially available tablet computers, our next step will be to explore how drag-and-pop and especially drag-and-pick can help tablet computer users work with external monitors. While this paper focused on icons, we plan to explore ways of operating menus, sliders, and entire applications using the techniques described in this article.

Acknowledgments

Thanks to Diane Kelly, Dieter Böcker, Lance Good, Amanda Williams, and LDUX. Work done at IPSI was supported by a grant from Microsoft Research.

References

Beaudouin-Lafon, M. (2000) Instrumental Interaction: An Interaction Model for Designing Post-WIMP Interfaces. In *Proc. CHI '00*. pp. 446–453.

Dachille, F. and Kaufman, A. (2000) High-Degree Temporal Antialiasing. In *Proc. Computer Animation* '00, pp. 49-54.

Dulberg, M.S., St. Amant, R., and Zettlemoyer, L.S. (1999), An Imprecise Mouse Gesture for the Fast Activation of Controls. In *Proc. Interact '99*, pp. 375–382.

Elrod, S., et. al. (1992) Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration. In *Proc. CHI'92*, pp. 599-607.

Geißler, J. (1998) Shuffle, Throw or Take It! Working Efficiently with an Interactive Wall. In *CHI '98 Late–Breaking Results*, pp. 265–266.

Guiard, Y., Bourgeois, F., Mottet, D. & Beaudouin-Lafon, M. (2001) Beyond the 10-Bit Barrier: Fitts' Law in Multiscale Electronic Worlds. In *Proc IHM-HCI 2001*, pp. 573-587.

Johanson B., Hutchins, G., Winograd, T., and Stone, M. (2002a) PointRight: Experience with Flexible Input Redirection in Interactive Workspaces. In *Proc. UIST '02*, pp. 227–234.

Johanson, B., Fox, A., and Winograd, T. (2002b) The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms, *IEEE Pervasive Computing,* 1 (2), 67-74.

Jul, S. (2002) Predictive targeted movement in electronic spaces. In *CHI'02 Extended Abstracts,* pp.626-627.

Myers, B., Bhatnagar, R., Nichols, J., Peck, C., Kong, D., Miller, R., and Long, C. (2002) Interacting At a Distance: Measuring the Performance of Laser Pointers and Other Devices. In *Proc CHI'02*, pp 33-40.

Rekimoto, J. and Saitoh, M. (1999) Augmented Surfaces: A Spatially Continuous Work Space for Hybrid Computing Environments. In *CHI'99*, pp. 378-385.

Rekimoto, J. Pick–and–Drop: A Direct Manipulation Technique for Multiple Computer Environments. In *Proc. UIST'97*, pp. 31–39, 1997.

Rubine, D. (1991) Specifying Gestures by Example. In *Proc. Siggraph'91*, pp. 329-337.

Shneiderman, B. (1998) *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Third edition. Reading MA: Addison-Wesley.

Streitz, N.A., Tandler, P. Müller-Tomfelde, C., Konomi, S. (2001) Roomware. In: Carroll, J.A. (Ed.), *Human-Computer Interaction in the New Millennium,* Addison Wesley, pp. 553-578.

Swaminathan, K. and Sato, S. (1997) Interaction design for large displays. In *Interactions* 4(1):15 – 24.

Wagner, A., Curran, P., O'Brien, R. (1995) Drag me, drop me, treat me like an object. In *Proc CHI '95*. pp. 525–530.

Zhai, S., Morimoto, C. Ihde, S. (1999) Manual And Gaze Input Cascaded (MAGIC) Pointing. In *Proc. CHI '99*, pp. 246-253.

Halo: a Technique for Visualizing Off-Screen Locations

Patrick Baudisch
Microsoft Research[1]
One Microsoft Way
Redmond, WA 98052, USA
+1 (425) 703 4114
baudisch@microsoft.com

Ruth Rosenholtz
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304, USA
+1 (650) 812 4390
rruth@parc.com

ABSTRACT
As users pan and zoom, display content can disappear into off-screen space, particularly on small-screen devices. The clipping of locations, such as relevant places on a map, can make spatial cognition tasks harder. Halo is a visualization technique that supports spatial cognition by showing users the location of off-screen objects. Halo accomplishes this by surrounding off-screen objects with rings that are just large enough to reach into the border region of the display window. From the portion of the ring that is visible on-screen, users can infer the off-screen location of the object at the center of the ring. We report the results of a user study comparing Halo with an arrow-based visualization technique with respect to four types of map-based route planning tasks. When using the Halo interface, users completed tasks 16-33% faster, while there were no significant differences in error rate for three out of four tasks in our study.

Keywords
Halo, visualization, peripheral awareness, off-screen locations, hand-held devices, spatial cognition, maps.

INTRODUCTION
People use maps in a number of tasks, including finding the nearest relevant location, such as a gas station, or for hand-optimizing a route. Using a map, users can easily compare alternative locations, such as the selection of restaurants shown in Figure 1a (as indicated by the barn-shaped symbols). Users can see how far away a restaurant is from the user's current location, and whether it lies close to other locations the user considers visiting. When users are required to use a zoomed-in view, however, for example to follow driving directions (Figure 1b), relevant locations disappear into off-screen space, making the comparison task difficult[2]. Comparing alternatives then requires users to zoom in and out repeatedly—a time-consuming process that can hardly be accomplished on-the-fly. Especially on small-screen devices, such as car navigation systems or personal navigation devices, this can severely limit a user's capability with respect to spatial cognition tasks.

CHI 2003, April 5-10, 2003, Fort Lauderdale, Florida, USA.

HALO
Halo addresses this issue by virtually extending screen space through the visualization of the locations of off-screen objects. Figure 2a shows a map navigation system that is enhanced with Halo. The figure shows the same detail map as Figure 1b, but in addition the display also contains the location information contained in Figure 1a. The latter is encoded by overlaying the display window with translucent arcs, each indicating the location of one of the restaurants located off screen. Figure 2b shows how this works. Each arc is part of a circular ring that surrounds one of the off-screen locations. Although the arc is only a small fragment of the ring, its curvature contains all the information required for locating the ring center, which is where the off-screen object is located. Although the display window shown in Figure 2a by itself contains no restaurant, the display informs users that there are five of them in the periphery and that the one located southwest is closest.

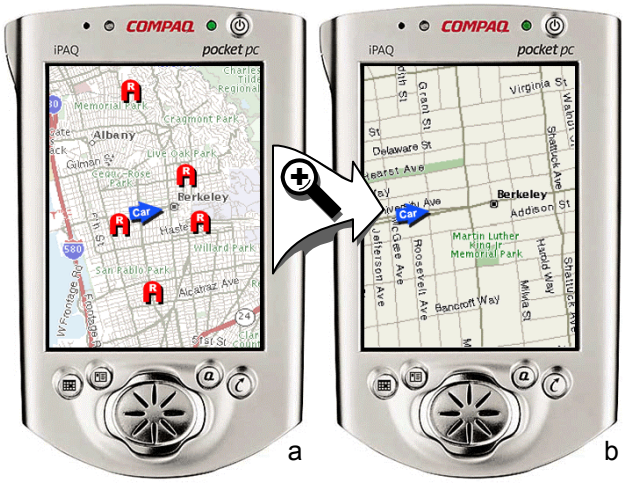

Figure 1: The problem: In order to make route decisions, users need to see the alternatives (a), but when drilling down to street information, relevant locations disappear into off-screen space (b).

Figure 3 shows how ring sizes are governed. As the map is panned, the restaurant moves from on-screen to off-

[1] The work reported in this paper was carried out during the first author's affiliation with Xerox PARC, now PARC Inc.

[2] See also the concept of *desert fog* in zoomable interfaces [13].

screen. As the restaurant icon reaches the border region of the display window, a ring grows under the icon. As the restaurant moves further off-screen, ring radiuses are recomputed dynamically, so that the ring is always just big enough to reach into the border region of the display window while never occluding the display's central region.

Figure 2: (a) Enhancing the map from Figure 1 with *Halo* shows where in off-screen space the five restaurants are located. (b) How it works: each off-screen location is located in the center of a ring that reaches into the display window.

Figure 3: As this location is panned out of the display window, a ring emerges from its center. The ring grows as the location is panned further away.

In the remainder of this paper, we discuss related work, present the concept and the design choices behind Halo, present our findings resulting from interviews with users of personal navigation devices, and present a user study comparing Halo with a more traditional arrow-based visualization style. We conclude with a discussion of the benefits and limitations of our visualization technique.

RELATED WORK IN VISUALIZATION TECHNIQUES

A substantial amount of research has been done on navigation aids, such as techniques for displaying driving [2] or walking directions [7]. While for following driving directions essentially any interface with an arrow was found to be sufficient [9], the contextual information required for route planning is more often supported using maps [14], e.g. for museum guides [1].

Several visualization techniques have been proposed for viewing large documents such as maps with limited screen resources. Multi-window arrangements, such as overview-plus-detail visualizations [16, 8], simultaneously display multiple views of the same map. However, the different scales of the individual views make it more difficult for users to integrate map information into a single consistent spatial mental model and require users to spend additional time reorienting when switching between views [3].

Focus-plus-context visualization techniques, e.g. fisheye views [11, 6], use only a single view onto the document, so that split attention is avoided. However, these techniques introduce distortion, which interferes with any task that requires precise judgments about scale or distance.

Another track of work has evolved around visualization techniques *pointing* into off-screen space. Figure 4 shows two everyday-life examples that use arrows to point to an off-screen highway and to off-screen game characters. Similar examples can be found in Pad++ [4] and in collaborative virtual environments, where lines emerging from a user's face help others see what the user is looking at [10]. By visualizing only selected off-screen content and by overlaying the visualization onto other display content, these "arrow-based" visualizations are very compact (see [12, 8] for additional research on semitransparent overlays). Their main limitation is that arrows convey only direction information, so that map navigation tasks would require arrows to be annotated with distances.

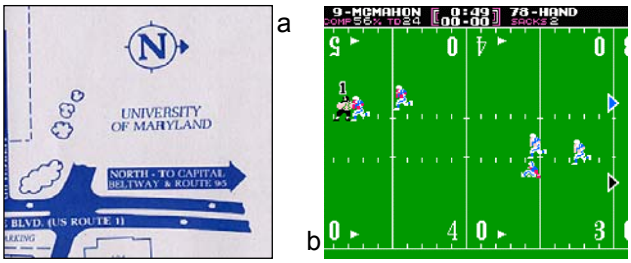

Figure 4: Related work: (a) The arrow on this map points to an unseen highway. (b) The arrows on the right point to football players off screen (© Nintendo '89).

Halo combines many of the advantages of the approaches listed above. It offers a single non-distorted view that allows users to inspect detail information without losing context. Unlike arrow-based visualizations, Halo does not require additional distance annotation; arcs provide full information about the location of off-screen objects, not only their direction. This eliminates the need for a scale indicator; the distance information encoded in the arcs always refers to the scale of the current scene. This allows users to carry out distance computations visually, which, as we show in the evaluation section of this paper, can improve user performance significantly.

CONCEPT AND DESIGN DECISIONS BEHIND HALO

The concept behind Halo derives from techniques well known in cinematography and theatre [5]. In cinematog-

raphy, conventions used to imply off-screen space include the use of *exit and entry points* (character exiting or entering through one of these points), *point-of-view* (character on-screen looking somewhere off-screen), and *partially out of the frame* (part of an on-screen prop protrudes into off-screen space) [15]. In *partially out of the frame,* viewers recognize the prop as being only a portion of the whole object, which implies that the rest of the object has to be in off-screen space.

The main difference between Halo and arrow-based techniques can be explained using this classification. Arrows-based techniques implement a *point-of-view* technique, which can convey only directional information. Halo uses the *partially out of the frame* technique, and by "attaching" the off-screen location to the prop, the prop conveys the full off-screen location information.

The *prop* has to fulfill two requirements. First, to allow viewers to mentally fill-in the missing off-screen parts, it has to be an object that viewers know and recognize. Second, the object has to display features that allow viewers to understand its position in space well enough to know the location of the attached target. The ring shape used by Halo fulfills both requirements. A ring is a familiar shape, and furthermore it fulfills the second requirement in an extraordinary way, since a ring can be reconstructed from any fragment. This tremendous redundancy makes rings robust against various types of mutilation, such as cropping at the window border or partial occlusion by other rings.

Furthermore, humans are efficient at searching for lines of higher curvature among lines of lesser curvature [18]. Thus the rings provide an advantage in searching for closer off-screen locations. This can be expected to have a positive impact on task completion time for many tasks striving for path length minimization, such as the search for the closest gas station on a map.

Halo implements a modified streetlamp metaphor

Our original concept for Halo was to represent off-screen locations as abstract "streetlamps" that cast their light onto the ground/map. This metaphor allowed us to derive four important properties for Halo. First, a streetlamp creates an aura, a large artifact which allows observers to infer the lamp's existence even if it is not in view. Second, the aura created on the map is round, resulting in the benefits discussed above. Third, light overlays itself onto objects without occluding them; overlapping auras from multiple lamps aggregate nicely by adding up light intensities. Forth, the fading of the aura with distance provides an additional visual cue about the distance of the streetlamp. An intense aura indicates a lamp located nearby; a weaker aura indicates a more distant lamp.

Our first prototype implemented this metaphor literally by using light auras on a dark background. The final design, (Figure 2) has undergone three modifications. First, in order to make it easier to perceive the halo curvature, we replaced the smooth transition at aura edges with a sharp edge. Second, to minimize occlusion of window content and overlap between auras, we replaced the disks with rings. Third, we inverted the color scheme resulting in dark halos on a light background in order to better accommodate typical map material, which used a light background.

The concept of fading arcs representing more distant locations was implemented by using translucency. Halo renders the short arcs that represent nearby locations as nearly opaque. Longer arcs representing more distant location are rendered with increasing translucency, which also compensates for the additional visual weight that their additional length would otherwise cause.

Within the framework set by the streetlamp metaphor, we made a series of additional design decisions with the goal of maximizing the visualization of location, particularly the indication of distance, which is a central theme in Halo. The design described in the following subsections introduces a third visual cue for distance, arc length.

Intrusion border and arc length

In order to limit the interference of arcs with display content, Halo restricts arcs to the periphery of the display. Only the space outside the *intrusion boundary* (Figure 5) is shared between arcs and content; the space *inside* the intrusion boundary is reserved exclusively for content.

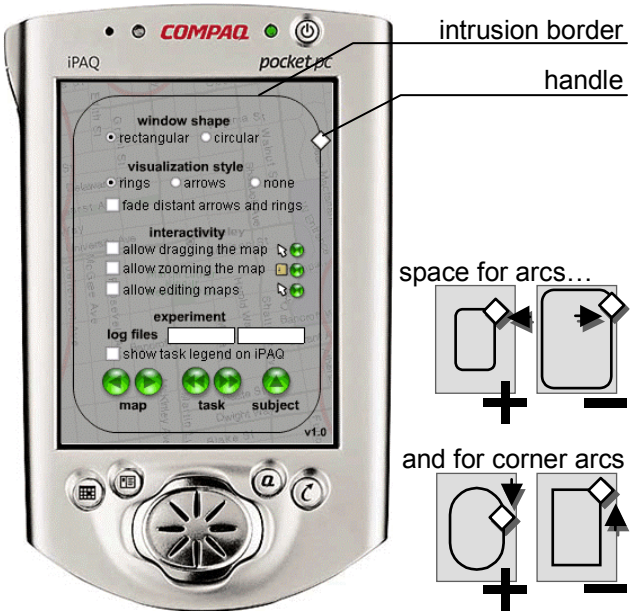

Figure 5: Halo preference dialog. By scaling the intrusion border (horizontal drag), users assigns space to arcs. Rounding corners (vertical drag) gives extra space to corner arcs.

The shape of the intrusion boundary was designed such that arc length would serve as another indicator for distance, in addition to curvature and opacity. Ideally, a longer arc would indicate that the represented object is further away than an object represented by a shorter arc. On a circular screen, as, for example, on a watch-type device, this is easily accomplished by using a circular

intrusion border. Here, arc length depends only on distance to the location, and, as Figure 6a illustrates, two arcs representing the same distance on such a device have the same arc length.

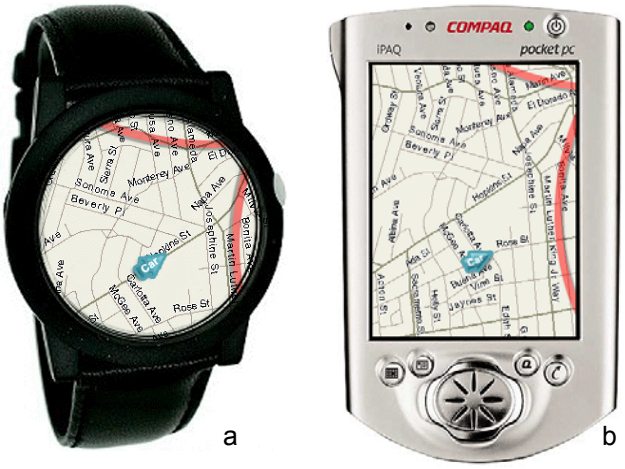

Figure 6: (a) On a circular display, arcs representing the same distance have the same length. (b) On a rectangular display, that is not always the case, because arcs in corners may be cropped.

On a non-circular display window, achieving correspondence between arc length and distance to the represented location requires additional attention. With a rectangular intrusion boundary, arcs cropped at a corner of the display window are shorter than arcs of comparable intrusion depth along an edge (Figure 6b). The accurate solution, i.e. computing intrusion depth on a per-arc basis as a function of the desired arc length, can make arcs intrude deeply into the display window, which conflicts with the notion of a space reserved for content. Halo therefore maintains the concept of an intrusion border limiting arc intrusion, but uses a rounded boundary (see Figure 5) to give extra intrusion depth and thus length to corner arcs.

Making Halo scale to large numbers of locations

Arcs mapping to similar positions on the intrusion border may overlap. In general, arcs are highly robust against overlap, but there are two cases where it can become an issue.

First, arcs of strongly collocated locations will yield arcs with large amounts of overlap along the entire length of the arc. Halo handles this by merging strongly overlapping arcs into a single *multi-arc* (Figure 7). Multi-arcs are created by rendering 2-3 thinner, concentric arcs, centered at their average location. Groups of four or more locations are indicated by a thick double ring. As the user pans towards a cluster, arc overlap will decrease, so that targets that are not exactly collocated will become individually accessible.

Second, scenarios involving a large number of off-screen locations can get cluttered, since the number of intersections between arcs grows quadratically with the number of arcs. For tasks where locations represent *alternatives*, Halo allows suppressing the rendering of locations that fall below a certain rank-specific relevance threshold. For tasks that require users to visit *all* targets, Halo allows showing *all* targets by merging arcs into multi-arcs using bottom-up clustering.

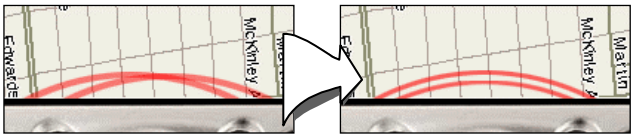

Figure 7: Overlapping arcs merge into double arc.

Design variables available for content visualization

Halo uses arc shape, arc length, and opacity for conveying location information. This means that a wide range of design variables, such as color, texture, and arc thickness, remain available for communicating additional properties of the respective off-screen locations, such as a restaurant's Zagat's rating. Applications designers may, for example, choose to overload such a relevance value to arc opacity (with the notion that relevance may compensate for distance), map it to arc thickness, or map it to color properties, such as hue.

In the next two sections, we move on to a preliminary field study and an experimental evaluation of Halo.

INTERVIEWS WITH NAVIGATION DEVICE USERS

In order to define realistic tasks for our user study, we conducted a preliminary field study. We interviewed 8 users who used five different personal navigation devices: 6 users of GPS devices and 2 users of personal digital assistants (PDAs) running map software. Participants were male researchers from three research labs who volunteered their participation. Each interview lasted between 10 and 40 minutes. We used an informal interview procedure covering the device, the application subjects used, and the subjects' tasks. In four cases, we obtained demonstrations of actual usage of the device. We also asked about specific problems with existing technology and suggestions for improvement. A summary of our results follows:

Driving directions: Two participants use *Garmin eMap* personal GPS navigation devices for driving directions (www.garmin.com/manuals/etrex_vis.pdf). They plan routes using their desktop computer, e.g. using *Microsoft Streets & Trips*, upload the results into the *eMap* device, and then follow the turn-by-turn directions. **Car compass**: One participant uses his *Magellan* GPS device as a compass, because, as he explains, compasses do not work in cars. **Finding home**: One participant uses his *Garmin eTrex Summit* GPS device to find his way back to the car when cross-country skiing or hiking. The device tells him how far he is away from his car, allowing him to return on time. It also shows him which direction to go. **Data collection**: Two participants use their *eMap* and *eTrex* GPS devices to generate location data for their research project, but do not interact with the devices directly. **Map planning**: Two participants use their PDAs (no GPS support) to find locations while in the city. The *iPAQ* Pocket

PC user runs a pocket version of *Microsoft MapPoint*. The *Palm Pilot* user runs *Vindigo*, a subscription service that features restaurants as well as up-to-date content, such as movie theaters schedules. *Vindigo* allows visualizing locations on a map.

Only the PDA users used their devices for making route decisions on the fly. The GPS device users found the screens too small (160x120 b/w pixels on the *eMap*) and screen redraw too slow (up to several seconds). Applying on-the-fly changes to routes on the GPS devices would be possible but would require a copilot. When deriving tasks for our experimental comparison, this gave extra weight to the two PDA users, although tasks and experiences of all eight users were considered.

Deriving tasks for the experimental comparison

Based on the interviews, we devised four experimental tasks that involved spatial cognition. Inspired by the hiker using his GPS device for returning to his car, we included a task where users would estimate the location and distance of off-screen locations. The second task was modeled after the iPAQ user who used his device for finding nearby restaurants. The iPAQ user also inspired the third task, namely organizing multiple locations into a single traversal. The forth and last task was modeled after the desire of the Palm user to see traffic conditions integrated into the route planning process. The two PDA users and one of the driving direction users mentioned the need to zoom frequently, so we included maps of variable scales in the experiment. We did not include a task involving users following directions, since it did not involve a significant amount of spatial cognition. We will describe all four tasks in detail in the following section.

Based on the results of our field interviews, we now had realistic tasks that would support a fair experimental comparison between different approaches to displaying contextual location information on a handheld device.

USER STUDY: HALO VS. ARROWS

In our user study, we compared Halo with an interface using an arrow-based visualization. Users had to complete four tasks. The main goal of this study was to determine which interface would allow users to complete their task fastest.

Interfaces/apparatus

Figure 8 shows the Arrow interface and the Halo interface used in the study. Both interfaces were designed for a Compaq iPAQ Pocket PC, which was emulated on the screen of a desktop computer. Emulation was necessary because for one task subjects were required to select locations outside of the iPAQ. For the study, we re-implemented an earlier Java version of Halo in Macromedia Flash™, extended it with features required for the study, and inserted functions logging the user's selections, task completion times, and error rates. The Flash version was also used to create the screenshots in this paper and the video figure. The emulated iPAQ screen measured 3" x 4", roughly 33% bigger than its real-life size. The laptop computer screen was a 12" screen run at 1024 x 768 pixels, 105 dpi resolution. Users made selections required by the tasks using an external mouse.

The Halo and the Arrow interfaces differed with respect to their way of indicating the location of off-screen locations. The Halo interfaces used red arcs for that purpose, as described in this paper. Instead of the arcs, the Arrow interface visualized off-screen locations by using arrows pointing along a line from the center of the screen to the off-screen locations and lined up with the border of the display window (see Figure 8a). Arrows were of the same color and opacity as the arcs of the Halo interface. Unlike the arcs, arrows were annotated with a three-digit number indicating the distance of the off-screen location from the display border. In order to allow users to interpret the number, there was a scale indicator at the bottom right inside the display window.

The Halo interface differed in two ways from that described in previous sections. First, to provide a clearer comparison of the arc and arrow cues to off-screen location, the fading of arcs was disabled, so that all arcs were of the same opacity. Second, in order to prevent users from obtaining the requested information through navigation, zooming and panning were disabled. Individual maps used scales ranging from 110m to 300m per cm on the screen. In order to provide users with a visual cue for the current zoom factor, a map was used as the backdrop, which scaled with the zoom. No other task information was available from the map. During the study, off-screen locations were never close enough to each other to require the use of the multi-arcs described earlier.

Figure 8: (a) The Arrow interface and (b) the Halo interface, both showing the same map. Which of the 5 off-screen restaurants is "closest" to the car?

Tasks

Users had to complete four tasks. Figure 9 shows example maps for each task. The users were instructed, "Complete each map as quickly as possible while maintaining reasonable accuracy." Distances in the task were 'as the crow flies', not distances along streets depicted in the map.

The "Locate" task: The user's task was to click in the off-screen space at the expected location of the off-screen targets indicated by each of the five red arrows/arcs (Figure 9a). Users were allowed to locate targets in any order; the system automatically picked the closest match.

The "Closest" task: Each map contained a blue car icon and five red arrows/arcs representing restaurants (Figure 9b). The user's task was to click on the arrow/arc corresponding to the off-screen location closest to the car.

The "Traverse" task: Each map contained a blue car icon and five target indicators. Targets could be either off-screen, indicated by red arrows/arcs, or on-screen (Figure 9c). The user's task was to select all five targets in order, so as to form the shortest delivery path, beginning at the car.

The "Avoid" task: The user's task, as "ambulance dispatcher," was to select the hospital farthest from traffic jams, thus most likely to be accessible to an incoming ambulance. Each map contained indicators of five on- or off-screen traffic jams, and three blue cross-shaped icons representing hospitals (Figure 9d).

Figure 9: Examples of maps used in the four tasks

Procedure

12 users participated in the study, including the second author of this paper, unpracticed with the use of the interface and tasks. There was no significant or observable difference between the performance of the second author and other users in the study and the author is excluded from any discussion of user preferences. We used a within-subject experimental design, i.e., each subject carried out all four tasks on both interfaces. In order to avoid sequence effects, task order, and interface order on a particular task, were counterbalanced between subjects.

Users received verbal instruction and four training maps for each interface, followed by eight timed maps. Upon completing each task, they answered questions about their interface preference for that task, and their impression of how confusing/clear the interfaces were. Upon concluding all tasks, users were asked to rate difficulty for each task, and to specify their overall interface preference. Users were interviewed upon completion of the tasks. The overall session took around 30 minutes.

Hypotheses

Our first hypothesis was that subjects would complete each task faster with the Halo interface than with the arrow-based interface. This hypothesis was based on the assumption that Halo arcs would allow for a faster perception of the represented locations than the textual annotation used by the arrow-based interface, and in particular that the gestalt of Halo arcs would help subjects perceive multiple locations at a glance. This, we expected, would help subjects form a spatial model, which would enable easier distance comparisons. Our second hypothesis was that subjects would experience an increase in task speed without an increase in error rate. Our third hypothesis was that higher efficiency would also result in higher subjective satisfaction with the Halo interface.

Results

Task completion time: Table 1 summarizes the average time subjects required to complete a map, for each task and interface. Confirming our first hypothesis, subjects achieved better task completion times in all four tasks when using the Halo interface. In the Locate task, task completion was 16% faster when subjects used the Halo interface. In the Closest task the difference was 33%, in the Traverse task 18%, and in the Avoidance task 16%. These results were significant, as discussed in more detail below.

Task	Arrow interface	Halo interface
Locate	20.1 (7.3)	16.8 (6.7)
Closest	9.9 (10.1)	6.6 (5.3)
Traverse	20.6 (14.1)	16.8 (8.7)
Avoid	9.2 (4.7)	7.7 (5.8)

Table 1: Average task completion times in seconds (and standard deviations)

We evaluated these differences in completion time using a repeated-measures ANOVA for each task. In each case, our model included factors of interface style (arcs/arrows), subject, map, order (arrows-first, arcs-first), and interaction effects between interface style and each of the other main factors. We used a conservative criterion for significance due to the large number of tests involved. Unless otherwise stated, all significant effects are significant at the $p<.001$ level. Due to space constraints, we present here only effects of interest, i.e. those involving interface type. Where appropriate, ANOVA's were performed on log response time. We also assumed binomially distributed data for percent correct data, and Gamma distributed data where appropriate for distance error data.

As mentioned above, the main effects of interface were significant for the Locate ($F(1,141=21.50)$), Closest ($F(1,140=54.85)$), and Avoid ($F(1,140=18.18)$) tasks, and marginally significant for the Traverse task

(F(1,140)=10.28, p=0.0017). We did find significant subject x interface interactions for the Closest (F(9,140)= 4.01) and Traverse (F(9,140=3.75) tasks. Closer examination revealed that in both cases, the interactions were the result of 2 out of 12 subjects (a different 2 for the 2 tasks) performing faster with the arrows than with the arcs, while all other subjects showed the opposite pattern. This interaction does not greatly affect our interpretation of the main effects.

From subject response, the higher cognitive load for localizing arrow targets seemed to be the major factor influencing the Halo interface performance advantage over the arrow-based interface, with 7/11 subjects volunteering that arrows "required too much math." Furthermore, two subjects volunteered that while some work was required for both interfaces to get a sense of the configuration of all targets, with the Halo interface this configuration would *persist*, which made tasks like the Traverse and Avoid tasks, where subjects had to acquire a mental model of the space, much easier.

Error rate: Table 2 shows the error for each of the four tasks. Due to the different nature of the four tasks, error was computed differently for each task. For the Closest and the Avoid tasks, which required subjects to choose among a small number of choices, we analyzed their percent correct performance. For the Locate task, we measured error as the Euclidian distance between the subject's location estimate and the actual location of the off-screen location. For the Traverse task, we used the difference in length between the traversal subjects chose and the optimal traversal for the respective map. The average total distance in the Locate task was 98 pixels, and the average optimal distance in the Traverse task was 1156 pixels.

For the Locate task, the ANOVA did find a significant main effect of interface on error (F(1,1039)=14.34), although the difference in error, the accuracy of the Halo interface was 5 pixels worse in average, was comparably small. For the Closest, the Traverse, and the Avoid tasks, Table 2 shows a reduction in error with the Halo interface, but none of these differences were significant (Traverse: F(1,166)=0.55, p=.54; Closest: F(1,154) = 0.05, p=.18; Avoid: F(1,154)=0.12, p=.27). This confirms our second hypothesis that faster task completion with the Halo interface would not come at the expense of error, for all tasks except the Locate task.

Task	Arrow interface	Halo interface
Locate	23.5 pixels (21.6)	28.4 pixels (33.8)
Closest	22% (42%)	21% (41%)
Traverse	97.4 pixels (94.7)	81.0 pixels (96.7)
Avoid	15% (35%)	14% (34%)

Table 2: Error rate (and standard deviations).

Dependence of error on distance: For the Locate task, we found, for both interfaces, a clear linear correspondence between distance and error, as well as a significant interaction between interface and distance (F(1,1039)= 114.58). Regression analysis yielded the relationships: Error(pixels) = 6.6 + 0.17*dist for arrows; and Error(pixels) = -6.4 + 0.37*dist for arcs. Since for Halo the incremental change in curvature gets smaller with growing radius, the distance awareness provided decreases with distance, and the error for arcs increases faster with distance than the error for arrows (p<.001).

Dependence of error on direction: We analyzed whether error depended on whether arcs were cropped at the corner of the display window. We found subjects to have twice as much error, on average, when the arc crossed a corner (M=52.3 pixels, SD=44.4) than when the arc lay along one side of the display (M=23.3 pixels, SD=28.7) (F(1,511)=41.6).

Distance error vs. direction error: To better understand the strengths and weaknesses of the two interface styles, we separated radial *distance* errors from *direction* errors, with *direction* error measured perpendicular to radial *distance*. This showed that the higher error in the locate task came mainly from the distance error. Subjects had significantly more bias towards underestimating distance with the arcs (M= -19.0 pixels, SD=38) than with the arrows (M= -0.6 pixels, SD=30) (F(1,1074)=81.80). There was no significant difference between arcs and arrows in direction errors (F(1,1022)=1.70, p=.81, M(arcs)=5.9, SD=9.4, M(arrows)=6.6, SD=7.5). These results are in line with our expectations, given our knowledge of the interface.

Subjective preference: For all four tasks, the majority of subjects who expressed a clear preference (see Table 3) preferred the Halo interface, which confirms our third hypothesis that improved efficiency would translate into preference. Overall, 6/11 subjects preferred the Halo interface, compared to 3/11 for the Arrow interface, with 2/11 having no overall preference.

Task	Arrow interface	Halo interface
Locate	2	8
Closest	3	6
Traverse	1	7
Avoid	2	4

Table 3: Number of subjects who expressed a clear preference for the respective interface. Remaining of 11 subjects expressed no preference.

Two subjects, one of whom preferred arrows and the other of whom had no preference, reported that they liked arrows because they could "just read the numbers… they just tell me what to do—I don't need to guess."

Discussion

Overall, our user study confirmed our hypotheses and provided evidence for Halo's usefulness in tasks involving spatial reasoning. Only our hypothesis regarding the error rate for the locate task was not borne out—the Halo interface did result in lower location accuracy than the

Arrow interface. As our analysis showed, the difference in accuracy was caused almost exclusively by subjects underestimating distances when using the Halo interface. Looking for an explanation, the comment of one subject seems relevant, who mentioned that the arc *felt* more like being part of an *oval*, rather than as part of a ring—which would be a valid explanation for the biased perception of distance. While this effect requires more detailed examination, we plan to address the issue by adapting Halo's geometric model to the mental model of the users. This means replacing the rings used in the current version of Halo with *ovals*, the parameters of which will be determined by the biases measured in the user study.

The other issue that needs to be addressed is subjective satisfaction. Despite the superiority with respect to task completion time, not all subjects preferred the Halo interface. Based on subjects' comments during the experiment, it seems that the *perceived* accuracy of the Halo interface may have been the reason for this. 6 subjects reported either that they felt they were more accurate with arrows, or they were uncertain of their accuracy with the arcs. We feel that this effect may partially be explained by the fact that interface panning and zooming was disabled in the experiment, so that subjects never got a chance to verify their model of off-screen geometry by panning the involved locations onto the screen. We expect some of this insecurity to go away with practice, particularly with the visual feedback that users get through panning and zooming.

CONCLUSIONS

In this paper, we presented Halo, a visualization technique providing users with location awareness of off-screen objects. Halo provides a single non-distorted view of a document, overlaid with location information for the off-screen locations. Unlike arrow-based visualizations, Halo does not require explicit distance annotation; the distance information is encoded in the arcs themselves and directly incorporates the scale of the scene.

We have presented a user study evaluating Halo in comparison to an arrow-based visualization technique. Tasks were picked based on the results of a field study, also briefly presented in this paper. Halo led to significant timesaving (16% to 33%) in the four experimental tasks, as well as higher subjective satisfaction.

In future work, we plan to explore the application of Halo in the area of real-time tasks, such as simulations or highly interactive games where Halo arcs will be used to continuously update users about the location of moving objects in the user's periphery.

Acknowledgments

Halo is an outgrowth of our work with Polle Zellweger, Jock Mackinlay, Lance Good, and Mark Stefik on City Lights techniques for providing awareness of and support for navigation to off-screen objects. Thanks also to Scott Minneman and Allison Woodruff for their comments on earlier versions of Halo.

REFERENCES

1. Abowd, G.A., Atkeson, C.G., Hong, J., Long, S., Kooper, R., Pinkerton, M. Cyberguide: a mobile context-aware tour guide. *ACM Wireless Networks* 3:421–433, 1997.
2. Agrawala, M.and Stolte, C. Rendering effective route maps: improving usability through generalization. In *Proc. Siggraph'01*, pp. 241–249.
3. Baudisch, P., Good, N., Bellotti, V., Schraedley, P. Keeping Things in Context: A Comparative Evaluation of Focus Plus Context Screens, Overviews, and Zooming, In *Proc. CHI 2002*, pp. 259–266.
4. Bederson, B. B., and Hollan, J.D. Pad++: A zooming graphical interface for exploring alternate interface physics. In *Proc. UIST' 94*, pp. 17–26.
5. Burch, N. 1983. *Theory of Film Practice*, London: Praeger Publishers Inc.
6. Carpendale, M.S.T., Montagnese, C. A. Framework for Unifying Presentation Space. In *Proc. UIST'01*, pp 61–70.
7. Chewar, C. M., McCrickard, D.S. Dynamic route descriptions: tradeoffs by usage goals and user characteristics. In *Proc. Smart Graphics '02*, pp 71-78.
8. Cox, D., Chugh, J., Gutwin, C., and Greenberg, S. (1998). The Usability of Transparent Overview Layers. In *Proc. CHI '98*, pp. 301–302.
9. Ekman, I., Lankoski, P. What should it do? Key issues in navigation interface design for small screen devices. *CHI'02 Extended Abstracts*, pp. 622–623.
10. Fraser, M., Benford, S., Hindmarsh, J., and Heath, C. Supporting awareness and interaction through collaborative virtual interfaces. In *Proc. UIST'99*, pp. 27–36.
11. Furnas, G. Generalized fisheye views, in *Proc. CHI '86*, pp. 16–23.
12. Harrison, B.L., Kurtenbach, G., Vicente, K.J. An Experimental Evaluation of Transparent User Interface Tools and Information Content Evaluation. In *Proc. UIST '95*, pp.81–90.
13. Jul, S. and Furnas, G. Critical zones in desert fog. Aids to Multiscale Navigation In *Proc. UIST 99*, pp.97–106.
14. Levine, M.; Marchon, I. and Hanley, G. The Placement and Misplacement of You-Are-Here Maps. *Environment and Behavior* 16(2):139–157, 1984.
15. Marsh T. and Wright, P. Using cinematography conventions to inform guidelines for the design and evaluation of virtual off-screen space. In *AAAI 2000 Spring Symp. Ser. Smart Graphics*, pp. 123-127, 2000.
16. Plaisant, C., Carr, D., & Shneiderman, B. Image-Browser Taxonomy and Guidelines for Designers, *IEEE Software*, 12(2):21–32, 1995.
17. Rosenholtz, R. Search asymmetries? What search asymmetries? *Perception & Psychophysics* 63(3): 476–489.
18. Treisman, A. and Gormican, S. Feature analysis in early vision: Evidence from search asymmetries. *Psychological Review*, 95(1):15–48.

Summary Thumbnails: *Readable* Overviews for Small Screen Web Browsers

Heidi Lam
University of British Columbia
201-2366 Main Mall
Vancouver, BC
hllam@cs.ubc.ca

Patrick Baudisch
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
baudisch@microsoft.com

ABSTRACT
In order to display web pages designed for desktop-sized monitors, some small-screen web browsers provide single-column or thumbnail views. Both have limitations. Single-column views affect page layouts and require users to scroll significantly more. Thumbnail views tend to reduce contained text beyond readability, so differentiating visually similar areas requires users to zoom. In this paper, we present *Summary Thumbnails*—thumbnail views enhanced with readable text fragments. Summary Thumbnails help users identify viewed material and distinguish between visually similar areas. In our user study, participants located content in web pages about 41% faster and with 71% lower error rates when using the Summary Thumbnail interface than when using the Single-Column interface, and zoomed 59% less than when using the Thumbnail interface. Nine of the eleven participants preferred Summary Thumbnails over both the Thumbnail and Single-Column interfaces.

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

General Terms: Human Factors, Design.

Keywords: Web browsing, small screen device, PDA, thumbnail view, overview, semantic zooming.

INTRODUCTION
Web pages are typically designed with the desktop screen in mind, and therefore often use multi-column layouts and preformatted page widths. Such pages can be hard to read on small screens. If rendered as is, the resulting page is typically much larger than the web browser screen and users need to scroll both horizontally and vertically to view it [27].

To avoid the need for horizontal scrolling, the majority of commercially available small-screen web browsers provide a single-column viewing mode that reformats the page by concatenating all its columns, thus displaying it as a single, very long column. Figure 1 shows an example. While this approach tends to work well for helping users read pages, it is of limited use while browsing. Since this approach affects the layout of pages so significantly, users may find it hard to recognize pages familiar from desktop viewing. (See the left side of Figure 3 for a better idea of what this page looked like before the conversion). This display style also significantly increases the required amount of vertical scrolling. As the scrollbar position in Figure 1 indicates, accessing the news story that used to be at a prime location at the top of the page now requires scrolling 8 screens down, past what used to be the menu column of that page.

Figure 1: Viewing the first headline in this single-column browser, requires scrolling 8 screens down.

To reduce the need for horizontal and vertical scrolling and to give users an overview of the page, researchers have proposed displaying web pages as a thumbnail view, i.e., a version of the page that is scaled down to fit the width of the small screen (e.g., [7]). Figure 2a shows an example. In the intended use, users start by viewing a web page in the thumbnail mode, rapidly identify the area of interest, and then zoom into that area for reading. The problem with this approach, however, is that the required size reduction typically renders text on thumbnails unreadable [2], as illus-

CHI 2005, April 2–7, 2005, Portland, Oregon, USA.

trated by the callout in Figure 2a. Due to the lack of readable text, users are often unable to tell similar looking areas apart and thus have to resort to exploring these locations using a tedious zoom-in-and-out or panning strategy.

Figure 2: (a) Traditional thumbnail views render text in this news page unreadable. (b) The same page as a Summary Thumbnail contains enough readable text to allow users to identify the area containing the sought content.

SUMMARY THUMBNAILS

To address this issue, we propose *Summary Thumbnails*. Summary Thumbnails are thumbnail views enhanced with fragments of readable text as shown in Figure 2b. Similar to traditional thumbnails, Summary Thumbnails preserve the original page layout which allows users to identify the overall page structure, and can help users recognize previously viewed pages. In Figure 2, for example, users may recognize the typical appearance of the homepage of the CNN news site and may find that the page consists of a thin menu column at the left, a news headline column in the middle, and a column of links and auxiliary materials at the right. Unlike traditional thumbnail views, however, the readable text fragments provided by Summary Thumbnails allow users to disambiguate the desired news story from similar looking areas. For example, the image caption shown in this Summary Thumbnail allows users to verify that this really is the story of interest concerning the Iraqi Governing Council. The readable text thereby eliminates the need for any further zooming or panning activity.

In order to fit the space requirements, mainly to fit the page to the browser width, Summary Thumbnails contain less text than the original web page (we will describe our text reduction algorithm in detail in the "Implementation" section). When zoomed in, however, Summary Thumbnails show the original, unabbreviated version of the page as shown in Figure 3. Zooming interactions involving a change in representation have also been referred to as *semantic zooming* [3]. Note that despite the change of representation during this zoom operation, the Summary Thumbnail and the detail views look similar enough for users to maintain a sense of which areas in the thumbnail correspond to which areas in the detail view.

Figure 3: The zooming interaction provided by Summary Thumbnails is a semantic zoom: zooming in replaces the abbreviated text with complete text.

Summary Thumbnails can be scaled arbitrarily, allowing them to fit the screen size of any target device. Font size can be adjusted independently, which allows adapting Summary Thumbnails to the requirements of the target scenario. In the case of Personal Digital Assistants or Smartphones with a crisp display, setting the font size to the smallest readable font size (e.g., 7 pixels, as done in Figure 2) maximizes the amount of viewable screen content. In other scenarios, reading text of that size would be hard. For example, the display may use very high resolution resulting in very small font, the display may be blurry (e.g., a TV set), or the user may be vision-impaired. In theses case, setting font size to a higher value can help address the problem. Figure 4 illustrates this at the example of a page being scaled to fit a variety of target screen sizes.

In the remainder of this paper, we give a brief summary of the related work and discuss the algorithm and implementation behind Summary Thumbnails in more detail. We then present two user studies in which Summary Thumbnails outperformed the competing Single-Column interface in terms of task time and accuracy, and the Thumbnail interface in terms of navigation effort. We conclude with a summary of our findings and some ideas for future work.

Figure 4: Summary Thumbnails can be adapted to arbitrary screen and font sizes. Here the same web page is shown on mockups of different devices.

RELATED WORK

Summary Thumbnails are associated with two fields of related work, i.e., small-screen browsing techniques and semantic zooming.

There are four general approaches to displaying web pages on small screen devices: device-specific authoring, multi-device authoring, automatic re-authoring, and client-side navigation [4]. The first two approaches obtain high-quality results by authoring for device specifics (e.g., [13]). This requires the effort and cooperation of the individual page authors, and cannot be applied to existing pages.

Automatic re-authoring and client-side navigation do not require the collaboration of page authors and are therefore more widely applicable. Research prototypes that use automatic re-authoring fall into two main categories: page reformatting and page scaling. An example of page reformatting is the aforementioned single-column views (e.g., *Small-Screen Rendering*™, opera.com). Other examples of techniques based on page reformatting include the *Power Browser* [6], where images and white space are removed, and the *WEST* browser [5], which uses flip zooming, a visualization technique that breaks pages into screen-sized tiles, and presents them as a stack. Difficulties with recognizing layout and leveraging the desktop browsing experience, as we have described them for single-column browsing, are common to all these approaches, since they all impact the page layout more or less significantly.

To avoid the drawbacks faced by page reformatting, researchers proposed approaches that preserve the appearance of the page by scaling the page, resulting in a web page thumbnail [7]. To help overcome the user effort involved in zooming thumbnails, researches have proposed extending thumbnail browsers by adding overview plus detail solutions ([18], also, in the *Thunderhawk* browser, thunderhawk.com), callouts for selected content areas (*WebThumb* [23]), user-selected column zoom (*SmartView* [16]), text summaries [11], rapid serial visual presentation display (*Streaming Thumbnails* [8]), and removal of irrelevant tiles (*collapse-to-zoom* [2]). Fisheye-based solutions such as fishnet [1] were shown to be useful for reducing the length of a web page. For reducing the width of a page, however, fisheye-based approaches can force users to scroll horizontally for each line of read text.

The second field of related work is semantic zooming, or techniques that change the representation of content during scaling or zooming [3, 12]. In the context of web browsing on the desktop, semantic zooming has been used to present query results [24, 9], to support keyword searching inside web pages [21, 1], and to improve accessibility [26]. For smaller screens, Gomes et al. [11] developed a system that displays text on PDAs at different semantic zoom levels: from displaying only the title at the lowest level, to displaying the complete original text. Lee & Grice [15] extracted text from XML-based files and displayed them in a customizable viewing style on PDAs. While these systems allow PDA users to view a larger selection of web pages, neither of them preserves the original layout of the viewed pages.

Researchers have found that displaying both the thumbnail and a text summary of a web page better supports page identification among query results [9]. These two elements can be presented separately [9] or integrated, e.g., in enhanced thumbnails where search words "popout" of otherwise unreadable thumbnails [24]. The concept of search term popouts was used to help users find keywords in web pages more efficiently by combining it with an overview-plus-detail approach (*popout prism* [21]) or a fisheye approach (*fishnet* [1]). While search term highlighting/popouts were proven to be effective [1], their applicability is limited to those cases where users can, and are willing to express their information need in terms of query terms.

Summary Thumbnails combine many benefits of the approaches listed above. As thumbnails, Summary Thumbnails preserve page layouts, and allow users to leverage their prior browsing experience. Readable text fragments allow users to disambiguate page content and to identity relevant areas. Further, since these text fragments are offered across the entire page, it takes less effort to skim the page than when using techniques that require users to explore the page using isolated focus areas [18, 23].

IMPLEMENTATION

A standard way of processing web pages for viewing on small screen devices is through a proxy server that transforms pages on-the-fly (e.g., *Thunderhawk* browser, thunderhawk.com, but also [26]). A proxy server is a program that receives web page requests (here from mobile devices), loads the respective pages, converts them, and serves them to the devices that requested them. Running the proxy on a powerful machine, such as a PC or server,

eliminates the need for processing on computationally weak mobile devices. Also, this approach makes it easier to serve different platforms, such as the ones mentioned above.

Our current implementation of Summary Thumbnails implements such a converter program, a standalone executable that runs on a desktop PC. However, since our immediate focus was to enable the user study, our converter still lacks the communication capabilities of a proxy and requires users to load and save pages manually.

Our converter program supports two output formats. First, it can output a converted HTML page. This page has the structure of a Summary Thumbnail, i.e., it contains abbreviated but enlarged text, but it s still as big as the original page. The final web page size reduction is performed by displaying the page on a web browser with scaling capabilities. We used the CSS zoom feature supported by MS Internet Explorer 5.5 and later [2] (msdn.microsoft.com).

Second, to implement our study conditions, we configured our converter to automatically render pages using a Microsoft WebBrowser control (http://msdn.microsoft.com), capture its output, and scale down the captured bitmap images to the desired output size using bi-cubic filtering. We created a simple viewer program for viewing and scrolling the resulting bitmaps. We used this viewer to administer all interface conditions in our user studies. Also all screenshots shown in this paper are screenshots of this viewer.

Our converter prototype is based on the MSHTML library, which prevents it from handling pages that contain frames. Besides that, conversion so far has worked for all web pages we sampled. Note however that since our converter program modifies only text, text encoded as bitmaps remains unchanged.

Text reduction

Removing common words: Words that otherwise occur less frequently tend to carry more of the meaning contained in a segment of text [19]. When reducing text, our converter program therefore removes common words first, as defined in a standard word frequency list [25]. The preservation of rare words also helps preserve keywords that a user might be visually scanning for. Alternatively, our converter program can be configured to crop paragraphs, which can be preferable for cases where text is already highly summarized, e.g., news headline. The Summary Thumbnail shown in Figure 2 was generated using the second method.

Preservation of line count: Our initial strategy was to make Summary Thumbnails preserve the overall length of the page. Since larger font in Summary Thumbnails is also taller, however, length preservation would have required removing entire lines, and this resulted in pages that appeared to be incomplete. Our current prototype therefore preserves the total number of lines instead of the absolute length of the page. The resulting Summary Thumbnails are typically longer than the corresponding thumbnails. The actual length increase depends on the amount of text in the original page. For the 44 pages used in our user studies, the increase in length ranged from 0 to 83% (median 33%). In cases where neighboring columns contain different amounts of text, text growth can affect the vertical alignment at the bottoms of the columns. For web pages we looked at so far, the misalignment seemed acceptable.

Omission of ellipses: It is common practice to replace removed content with placeholders [2] or ellipses to indicating the omission, yet we decided against that option. Due to the high degree of text reduction in Summary Thumbnails this would have add visual noise, be spatially expensive, and would render ellipses mostly meaningless given their ubiquity.

Detailed description of our algorithm

Here is a schematic overview of our algorithm. First, the page is loaded. Then all text that is smaller than a user-defined threshold is enlarged. The result is a page that is still a wide as the original page, but now contains large text, e.g., about 19 pt for a page to be displayed on a PDA. Enlarged text typically occupies more lines than at its original font size. To preserve the line count, our program removes words until the total numbers of lines in the paragraphs are preserved. The resulting page is then saved in HTML format for devices with scaling capabilities, or rendered, scaled, and saved as a bitmap for all other devices. The following two paragraphs describe this process in additional technical detail.

The page is loaded and partitioned into *elements* (`IHTMLElements`, as defined `in` the Microsoft MSHTML library, msdn.microsoft.com) by recursively traversing the page's Document Object Model (w3.org). Elements can be paragraphs of text, input boxes, option boxes, or other elements specified in a style sheet.

To reduce the text, our prototype iterates through all elements of the page. For each element it performs the following steps: (1) Width, height, and font attributes are extracted. The number of lines of text is not directly available, so it is estimated as cell height/font height. (2) The text is extracted from the element and stored in a string. (3) The font of the string is enlarged. (4) *First reduction:* The width of the string is compared with a first estimate of the available space, i.e., width of the element multiplied by the number of text lines in the element. Lowest ranked words are removed until this space requirement is met. (5) *Second reduction:* If the element contains multiple lines of text, the string is broken to the width of the element. More words are removed until meeting these more stringent space requirements. (6) If the element has not enough room to even accommodate a single word, one word is selected and cropped to fit the space constraints. (7) The `innerText` property of the element is updated. For text that is separated by tags, the original element's `innerHTML` property is traversed. Rather than replacing this text as a whole, all those words that had been removed from the string earlier are now removed from `innerHTML`, to preserve the original

formatting. (8) Appropriate <font> tags are inserted to increase the font size of the text in the element. (9) The resulting page is exported as described above.

QUALITATIVE USER STUDY

Our first user study was a qualitative study. Its purpose was to solicit initial user responses to our design and obtain a sense of appropriate tasks for use in the following quantitative study.

Study Design

Participants: We recruited 9 participants internally (7 male). All were mobile device users.

Interfaces: The participants used three different interfaces: a Thumbnail interface (Figure 2a), a Single-Column interface (Figure 1), and a Summary Thumbnail interface (Figure 3). All interfaces were displayed on a laptop computer using a viewer program that offered a net display area of 240x320 pixels for actual page content. All three interfaces allowed participants to scroll vertically through the web pages using the scrollbar. All interfaces fitted web pages to the screen width to remove the need for horizontal scrolling. Thumbnail and Summary Thumbnail interfaces were early design prototypes; while we explained the zooming capabilities to the participants, these prototypes did not yet support zooming; zooming support was not added until the quantitative study.

Procedure: Participants were presented with the same news page (http://news.bbc.co.uk) displayed on the interfaces in random order. They were told to scroll through the page and pick a news story they deemed interesting and elaborate on how it was represented on the respective interface. We encouraged participants to "think-aloud" during this process. For each interface, we conducted a brief interview and asked the participants to list pros and cons of the individual interfaces, and what page types they expected each interface to be most and least suitable for. We asked additional questions about application scenarios using a questionnaire. Overall, the study took 45 minutes per participant. Participants received a minor gratuity for their participation.

Results

Thumbnail interface: According to participants, the major advantages of the Thumbnail interface were that it preserved page layout (4 reports), that it provided an overview (8 reports), and that it provided a sense of branding (2 reports). However, six participants said the text—and for some pages images as well—could end up being too small to be useful. Two participants expressed concerns about the need for zooming before the text information on the page could be read.

Six participants judged the Thumbnail interface as useful in situations where the layout of the pages aided navigation, e.g., in maps and previously visited sites, or where the images conveyed enough information for decision-making, e.g., shopping sites that made heavy use of images. Seven participants judged the interface as inappropriate for pages that required users to actually read the page, as opposed to just scanning it (e.g., portal pages, lists of search results).

Single-Column interface: Six participants rated the legible text offered by the Single-Column interface favorably, and one said he felt more confident before clicking a hyperlink using this interface than when using any of the other two study interfaces. Four participants liked that this interface avoided the need for horizontal scrolling. Eight complained about the altered layout and three participants described the transformed web page as being "ugly", or "unrecognizable". Four participants expressed dislike for the large amount of vertical scrolling this interface required; one participant liked the fact that all information could be viewed by vertical scrolling only.

Seven participants judged this interface as useful for linear reading (e.g., news, search results), and six said they found it inappropriate for tasks that relied heavily on the original layout of the page (e.g., map, bus schedules, and carefully designed pages). One participant expressed concerns that the Single-Column interface would be inappropriate for tasks that required users to compare different parts within the page (e.g., performing price comparisons while making a purchase decision).

Summary Thumbnail interface: Eight participants found and liked that the Summary Thumbnail interface preserved page layout better than the other two interfaces. Seven mentioned the text in the Summary Thumbnail interface to be legible, and judged this superior to the thumbnail interface. However, one participant was concerned that this interface would show too little text information for him to be able to select areas for further investigation and was concerned that the abbreviated text content offered by this interface might be misleading.

All nine participants judged the interface as suitable for tasks that relied on the layout of the page (e.g., maps, tables, and professionally designed pages), but not for linear reading (e.g., reading of news articles or search lists).

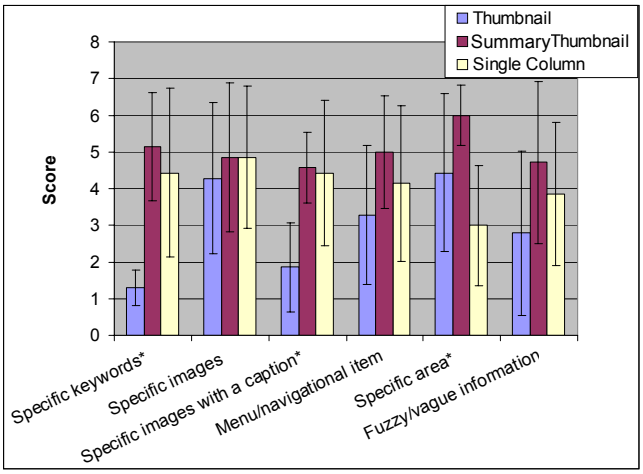

Figure 5: Questionnaire results. Scores are averages of 9 results (1=useless, 7=extremely useful, * one or more differences significant).

Questionnaire results and overall preference: Figure 5 shows the results of the questionnaire in which participants rated how appropriate they judged the individual browsers for six given types of web browsing tasks. We performed a Kruskal-Wallis test on the satisfaction ratings for the three browsers across the six task types, and found three significant differences. The majority of participants judged the Summary Thumbnail interface as more useful than Thumbnail interface for keyword search (χ^2=11.4, p=0.003) and captioned image searches (χ^2=9.0, p=0.01), and more useful than the Single-Column interface when looking for a specific area on a previously visited page (χ^2=7.8, p=0.02).

In the final ranking, 8 of 9 participants ranked the Summary Thumbnail interface first; one preferred the Single-Column interface.

QUANTITATIVE USER STUDY

In this study, we evaluated the performance of the same three small screen interfaces (Summary Thumbnail, Thumbnail, and the Single-Column interfaces). In addition, we also included a web browser using a desktop-sized display window ("Desktop interface") as a common reference. The participants' task was to locate information in web pages originally designed for the desktop. Our main hypotheses were that the Summary Thumbnail interface would require less scrolling than the Single-Column interface and less zooming than the Thumbnail interface and that this would lead to measurable task time and accuracy differences. The experiment was a within-subjects design with a single manipulation (Interface). The dependent variables were task time and error rate. In addition, we recorded zooming and scrolling events.

Participants

11 participants (7 males) from the local community were recruited for the study. Ages ranged from 23 to 52 years, median 42 years. All had prior experience with desktop web browsers, and 4 out of 11 of the participants had previously used a mobile device for web browsing. Two of them indicated that they preferred using a 2D spatial view with horizontal and vertical scrolling for web browsing on their PDAs while the other two preferred the single-column mode on their devices.

Interfaces

In the experiment, participants viewed web pages using the following four interfaces: a Thumbnail interface, a Summary Thumbnail interface, a Single-Column interface, and a Desktop interface. The first three were similar to the prototypes used in the qualitative study reported earlier, but with additional interaction functionality. In addition to using the scrollbar, participants could now vertically scroll using keyboard or mouse wheel. The Thumbnail and Summary Thumbnail interfaces also allowed participants to zoom into a 1:1 scaled view of the original page by clicking the respective area in the thumbnail view as shown in Figure 3. While zoomed in, participants could scroll the page in all four directions by pressing the arrow keys on their keyboard or by using the scrollbars. The Desktop interface displayed the original web pages without reformatting or scaling, and only vertical scrolling was provided via scrollbar and keyboard. Single-column views were generated using an Opera browser supporting Small-Screen Rendering™ (opera.com). All four interfaces were able to display all pages used in the study without requiring horizontal scrolling—horizontal scrolling occurred only in the *zoomed-in* views of the two thumbnail-based interfaces.

All interfaces were presented on an 18" LCD screen running at 1280x1024 pixel resolution. As shown in Figure 6, the top area of the 818x827 pixel study environment contained a textual description of the task, while the remaining display area was used to display the respective interface. For the thumbnail-based interfaces and the Single-Column interface, the window contained a simulated PocketPC device with a display area of 240x320 pixels. Since we included the Desktop condition as an approximate upper bound on performance on these tasks, we did not intentionally limit the browsing space used by these interfaces. In fact, our Desktop offered a viewing space of 800x600 pixels (Figure 6b).

Figure 6: Program used to administer the quantitative experiment running the (a) Summary Thumbnail, (b) Desktop, and (c) Single-Column interface.

Task

Each trial started with a written task description displayed at the top of the study environment (Figure 6). When ready, the participant clicked a button to reveal the web page described and to start the timer. The participant's task was to locate the information described in the textual description and place a marker on that location. Markers were placed by clicking into that area or by dragging an existing marker with the mouse. Size and shape of the area considered a correct answer varied across pages, but all were large enough to allow easy placement of the marker. When satisfied, the participant pressed a button to complete the trial.

Example: The page in Figure 6 shows an imdb.com review for the movie Shrek 2. The task description at the top of the window reads *"You are thinking about renting a movie for the night, and remember your friend once recommended "Shrek 2". You want to see what the rating is before heading to the video store. You have used this movie review/database site in the past, so you went to the home page, searched for "Shrek 2" by name, and navigated to the following page. You expect to see the rating information somewhere in the main section of the page, and you know they will be next to those bright yellow stars. <Click "Ready To Start" to look for the rating of the movie "Shrek 2">.*

How these tasks were created

To obtain a balanced set of web pages and a description of an actual information need we went through the following three-step procedure instead:

First, we collected web pages and task descriptions by interviewing 12 university student volunteers. These volunteers did not participate in the actual study. During the interviews, the volunteers randomly selected three to five web pages from their recent browser history. For each of these pages, they told us why they visited the page, how they got to the page, and where they expected target information to appear on the page before it was displayed. We gathered a total of 45 pages. Figure 7 shows Summary Thumbnails of some of these pages.

Next, we manually aggregated the gathered information into task descriptions, each consisting of a web page and a brief task and background description. Figure 6 shows one example.

Finally, we had two internally recruited pilot participants perform all 45 tasks using the desktop interface. We removed a page because both pilot participants found its description ambiguous. Based on the pilot participants' task times, we divided the remaining 44 pages into the four sets of 2 (training) + 9 (timed) trials, such that the sets balanced both average task time and page types (e.g., image, text box, main section of the page etc.). During the study participants performed all four trial sets—each one on a different interface. Presentation order and the assignment of trial sets to interfaces were counterbalanced.

Figure 7: Summary Thumbnails of seven of the 44 pages used in the quantitative user study.

Procedure

At the beginning of the study, participants filled in a questionnaire with demographic information. Participants then completed 2 training and 9 timed trials using the first interface, after which they filled out a short questionnaire about that interface. Then they repeated the procedure with a different sets of pages on the remaining three interfaces. The presentation order of pages and their assignment to interfaces was counterbalanced. Finally, participants were asked about their overall preference.

Hypotheses

Our hypotheses were:

1. *Displaying web pages in a way that preserves their original layout allows users to locate information faster and more accurately.* For tasks where participants were able to find desired information based on page layout, we expected the desktop interface to perform best, followed by the Summary Thumbnail interface, and the Thumbnail interface. The Single-Column

interface should affect page layout more than the other interfaces and should therefore perform worse on such pages.

2. *The presence of readable text reduces the need for zooming navigation.* We expected the Summary Thumbnail interface to require less zooming than the Thumbnail interface. The Desktop and the Single-Column interfaces obviously required no zooming.
3. *Shorter pages require less scrolling*. We consequently expected the Desktop interface to require the least amount of scrolling. Since Summary Thumbnails were slightly longer than the corresponding thumbnails, the Thumbnail interface should require less scrolling then the Summary Thumbnail interface. We expected the Single-Column interface to require by far the highest amount of scrolling.
4. *Effect of incomplete text fragments are tolerable*. While incomplete text fragments on the Summary Thumbnails could potentially lead to misinterpretation and a higher error rate in those trials, we expected these effects to be minor.

Results

Unless otherwise stated, we used single-factor, Analysis of Variance to analyze the data. We performed post-hoc analyses with Bonferroni corrections for multiple tests to further explore significant main effects. We used an alpha value of p=.05 across all these tests.

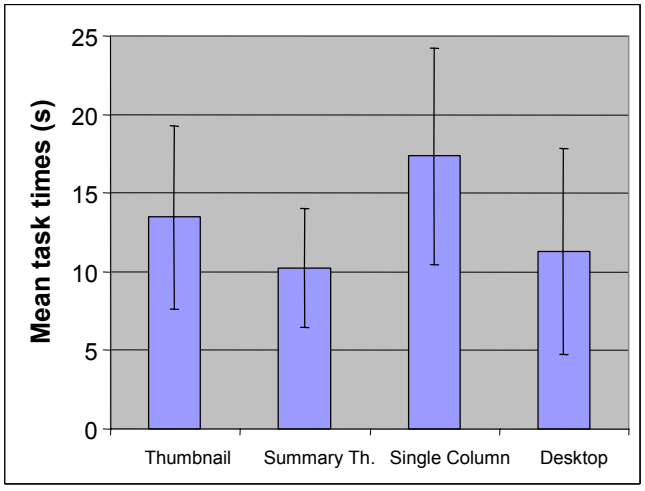

Figure 8: Mean task times for each interface (+/- standard error of the mean).

Task completion time

Task time results are shown in Figure 8. ANOVA indicated a significant main effect of interface ($F(3,40)=3.12$, $p=0.04$). Post-hoc analyses showed that Summary Thumbnail trials were significantly faster than Single-column trials. The difference was 41%.

Task Accuracy

Figure 9 shows the error rates, i.e., cases where participants had incorrectly placed the marker, thus failed to locate the intended target.

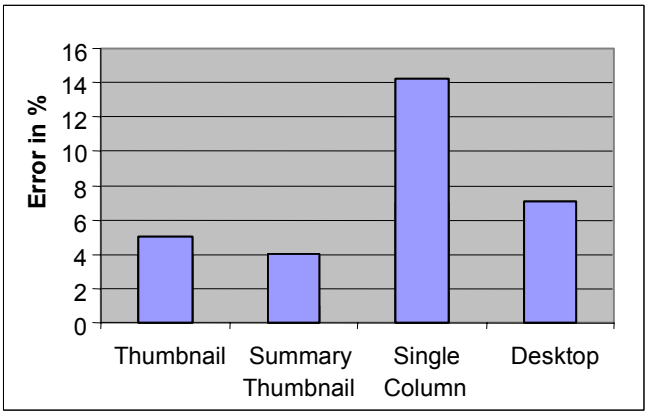

Figure 9: Error rates: percentage of erroneous trials across all 9 participants.

Zooming

Since there were only two interfaces that allowed zooming, an unpaired, two-tailed t-test was used to analyze the results. Participants zoomed 59% less often when using the Summary Thumbnail interface than when using the Thumbnail interface ($t(20)=2.1$, $p<0.001$, Figure 10). Three participants did not zoom in at all in the Summary Thumbnail trials.

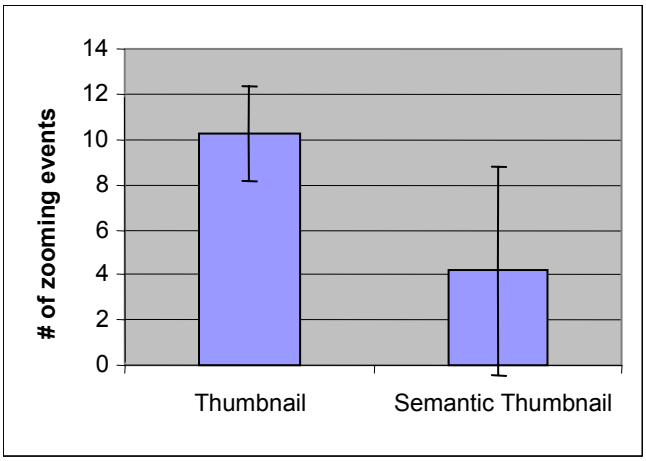

Figure 10: Zooming event counts (+/- standard error of the mean).

Zooming accuracy: Next, we looked at what percentage of participants' zoom-in interactions were successful, i.e., the resulting detail view did contain the target area. The difference (62% with the Thumbnail and 51% with the Summary Thumbnail interface) was not statistically significant.

Scrolling

The vertical scrolling results are shown in Figure 11a. A single-factor ANOVA revealed a main effect of Interface ($F(3,40)=12.5$, $p<0.001$). Post-hoc analyses showed that the Single-Column interface required significantly more vertical scrolling than any of the other three interfaces (e.g., 6.9 times more than the Summary Thumbnails). When zoomed in, horizontal scrolling results are shown in Figure 11b. Participants scrolled 88% less horizontally when using the Summary Thumbnail interface than when using the Thumbnail interface ($F(1,20)=15.3$, $p<0.001$).

We observed that participants scrolled back and forth when they had trouble orienting themselves. Scroll direction change results may therefore give a sense of participants' level of confusion while searching for the targets (Figure 11c and d). A single factor ANOVA of the scrolling data for the four interfaces revealed a main effect ($F(3,40)=4.3$, $p<0.01$) for the vertical direction change. Post-hoc analyses showed that this parameter is 4 times lower in the Summary Thumbnail than the Single-column trials. The Summary Thumbnail trials contained horizontal scroll direction change 82% less than the Thumbnail trials ($t(20)=3.9$, $p<0.001$).

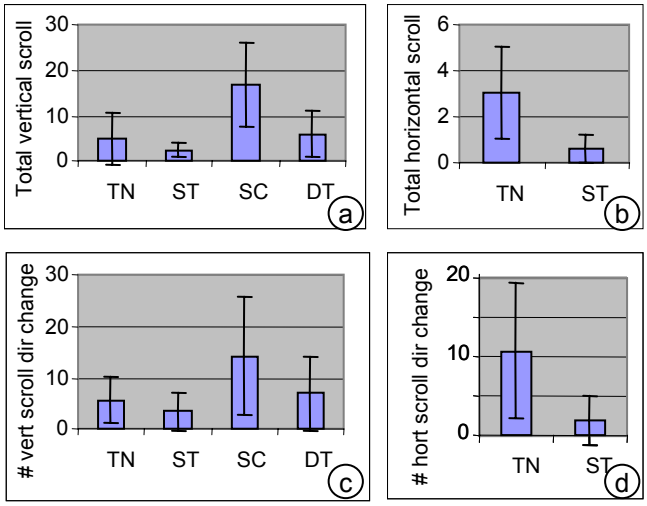

Figure 11: Scrolling results. (a) total amount of vertical scrolling for the interfaces (unit=1000 pixels); (b) total amount of horizontal scrolling while zoomed in (unit=1000 pixels); (c) total number of vertical scroll direction change events; (d) total number of horizontal scroll direction change events while zoomed in (TN=Thumbnail; ST=Summary Thumbnail; SC=Single-column; DT=Desktop).

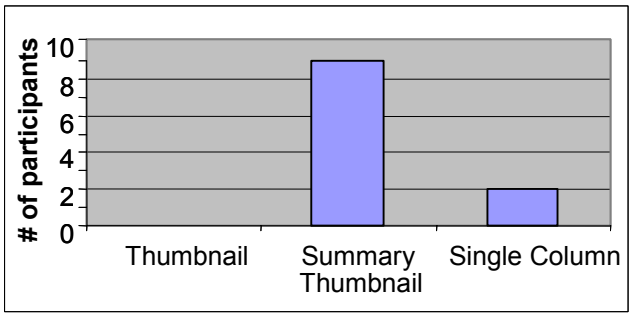

Figure 12: Participant's preferences for use on their personal browsing device

Overall preference

At the end of the study, we asked participants "Which of the three web browser: Thumbnail, Summary Thumbnail, and Single-column would you install on your mobile device?" As shown in Figure 12, nine out of 11 participants preferred the Summary Thumbnail interface ($\chi^2(2)=12.2$, $p=0.002$), while the remaining two selected the Single-Column interface.

DISCUSSION

The results of the study indicated a strong participant preference for the Summary Thumbnail interface over the Single-column and the Thumbnail interfaces on small screens when browsing web pages originally designed for the desktop.

Our results suggest that better preservation of page layout can help participants browse. Trials using the Summary Thumbnail interface were 41% faster than those using the Single-Column interface where layout was considerably altered. Participants also made more mistakes when using the single-column interface than when using any of the other interfaces. Our observations during the study indicate that the layout conversion of the Single-Column interface may have affected participants' ability to recognize some page elements, such as horizontal tabs or horizontal menus, as well as overall page structures. In fact, the Single-Column trials contained more back-and-forth scrolling than all other trials, indicating that participants had difficulties in orientation. This confirmed our first hypothesis, wherein we postulated that layout information would help visual search.

Our study results also supported our second hypothesis: layout information in itself is not always sufficient for locating content—readable text is required as well. By offering fragments of readable text, the Summary Thumbnail interface was able to reduce the amount of zooming by 59% when compared to the Thumbnail interface. Three participants even located all trial targets without ever zooming into the detail view. Another indicator of direct access to target information was scrolling. Summary Thumbnail trials contained less horizontal scrolling and scrolling direction changed less often than in the Thumbnail trials. These results may indicate that participants could disambiguate the page content better using the Summary Thumbnails than with "plain" thumbnails.

Interestingly, participants scrolled 51% less when using the Summary Thumbnail interface than when using the Desktop interface. While this result may seem surprising at first, it is easily explained: Since the Summary Thumbnail interface (as well as the Thumbnail interface) was running in the "portrait" aspect ratio typical of *handheld devices*, it allowed participants to see more of the length of the page than the Desktop interface, which used a landscape aspect ratio. The differences in scrolling did not lead to any significant effects in task time or error rate. Still, we are surprised to see that the Desktop interface did not clearly outperform the Summary Thumbnail interface. One possible interpretation of this finding is that the reduced amount of scrolling and the reduced amount of text participants were confronted with on the Summary Thumbnail compensated for the obvious benefits of the Desktop interface.

Our concern that text cropping in the Summary Thumbnails interface would lead to a higher error rate was not confirmed. Instead, participants made fewer errors with the Summary Thumbnail interface compared to the Single-Column interface, where all of the original text was avail-

able. Also, participants seemed to find it easier to orient themselves with the Summary Thumbnail interface than the Single-Column interface, as indicated by the scroll direction change results.

CONCLUSIONS AND FUTURE WORK

In this paper, we presented Summary Thumbnails, a technique for displaying web pages on small screen devices that combines the benefits of thumbnail-based web browsers with the benefits of readable text. Our user study results indicate that Summary Thumbnails were found to be more effective in supporting web browsing than the single-column browsing technique that currently dominates the commercial market of small screen devices. Also, the vast majority of participants preferred Summary Thumbnails over traditional thumbnail views as well as single-column browsing.

As future work, we plan to combine Summary Thumbnails with a more powerful zooming interaction (e.g., *collapse-to-zoom* [2]) and the auto-cropping of photos [22]. In addition, we plan to explore the applicability of Summary Thumbnails for illustrating search results and as an alternative to the Thumbnail view in the file system viewer.

Acknowledgments

We would like to thank Mary Czerwinski, Ed Cutrell, Sue Dumais, Maneesh Agarwala, Dane Howard, Dan Vogel, Xing Xie, and Chong Wang for their support.

REFERENCES

1. Baudisch, P., Lee, B., and Hanna, L. Fishnet, a fisheye web browser with search term popouts: a comparative evaluation with overview and linear view. In *Proc. AVI 2004, pp 133-140.*
2. 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. To appear in *Proc. UIST 2004*.
3. Bederson, B.B., Hollan, J.D., Perlin, K., Meyer, J., Bacon, D., and Furnas, G., Pad++: a zoomable graphical sketchpad for exploring alternate interface physics, *Journal of Visual Languages and Computation* , 7, 1996, p3-31
4. Bickmore, T., and Schilit, B, Digestor: Device-Independent Access to the World Wide Web. In *Proc. Seventh Intl. WWW Conference 1997,* pp. 655–663.
5. Bjork, S., Bretan, I., Danielsson, R., and Karlgren, J. WEST: A Web Browser for Small Terminals. In *Proc. UIST'99,* pp. 187-196.
6. Buyukkokten, O., Gracia-Molina, H., Paepcke, and Winograd, T. Power Browser: Efficient Web Browsing for PDAs. In *Proc. CHI 2000*, pp. 430-437.
7. Chen, L., Xie, X., Ma, Y. W., Zhang, H-J., Zhou, H. Feng, H. DRESS: A Slicing Tree Based Web Representation for Various Display Sizes. Technical Report MSR-TR-2002-126, December 2002.
8. Czerwinski, M. P., van Dantzich, M., Robertson, G., and Hoffman, H. The Contribution of Thumbnail Image, Mouse-over Text and Spatial Location Memory to Web Page Retrieval in 3D. In *Proc INTERACT'99*, pp. 163-170.
9. Dziadosz, S., and Chandrasekar, R. Do Thumbnail Previews Help Users Make Better Relevance Decisions about Web Search Results? *Proc SIGIR'02*, pp. 365-366.
10. Gajos, K. and Weld, D. SUPPLE: Automatically Generating User Interfaces. In *Proc. IUI'04*, pp. 93-101.
11. Gomes, P., Tostao, S., Goncalives, D., and Jorge, J. Web Clipping: Compression Heuristics for Displaying Text on a PDA. *Mobile HCI '01* (poster paper), 2001.
12. Good, L., Bederson, B., Stefik, M., and Baudisch, P. Automatic Text Reduction For Changing Size Constraints. In *CHI'02 Extended Abstracts*, pp. 798-799.
13. Jacobs, C., Li, W., Schrier, E., Bargeron, D., and Salesin, D. Adaptive Grid-Based Document Layout. In *Proc. SIGGRAPH'03,* pp. 838-847
14. Lawton, D. T. and Feigin, E. J. Streaming Thumbnails: Combining Low Resolution Navigation and RSVP Displays. In *Proc. CHI'00*, pp. 159-160.
15. Lee, K. B., Grice, R. A. Getting and Giving Information: An Adaptive Viewing Application for the Web on Personal Digital Assistants. In *Proc. 21st annual international conference on Documentation*, pp. 125-132.
16. Milic-Frayling, N. and Sommerer, R. Smartview: Enhanced document viewer for mobile devices. *Technical Report MSR-TR-2002-114*, Microsoft Research, Cambridge, UK, November 2002.
17. Nygren, E., Lind, M., Johnson M., and Sandblad, B. The Art of the Obvious. In *Proc. CHI '92*, pp. 235-239.
18. O'Hara, K., Sellen, A. and Bentley, R. Supporting Memory for Spatial Location while Reading from Small Displays. In *Proc CHI'99*, pp. 220-221.
19. Salton, G. and McGill, M.J. *Introduction to Modern Information Retrieval.* New York: McGraw-Hill, 1983.
20. Stone, M. C., Fishkin, K., and Bier, E. A. The Movable Filter as a User Interface Tool. In *Proc. CHI'94*, pp. 306-312.
21. Suh, B., and Woodruff, A. Popout Prism: Adding Perceptual Principles to Overview + Detail Document Interfaces. In *Proc. CHI'02*, pp. 251-258.
22. Suh, B., Ling, H., Bederson, B. B., and Jacobs, D. W. Automatic Thumbnail Cropping and Its Effectiveness, In *Proc. UIST'03,* pp. 95-104.
23. Wobbrock, J., Forlizzi, J., Hudson, S., Myers, B. WebThumb: Interaction Techniques for Small-Screen Browsers. In *Proc. UIST'02,* pp. 205-208.
24. Woodruff, A., Faulring, A., Rosenholtz, R., Morrison, J., and Pirolli, P. Using Thumbnails to Search the Web. In *Proc. CHI'01*, pp. 198-205.
25. Word frequency list from http://www.comp.lancs.ac.uk/ucrel/bncfreq/lists/6_1_1_all_tag.txt
26. Hanson, V. L. and Richards, J. T. A Web Accessibility Service: Update and Findings. In *Proc. SIGACCESS '04,* pp.169-176.
27. Jones, M. Marsden, G., Mohd-Nasir, N., Boone, K. Buchanon, G. Improving Web Interaction on Small Displays. In *Proc. WWW8 conference*, pp. 1129-1137.

Snap-and-go: Helping Users Align Objects Without the Modality of Traditional Snapping

Patrick Baudisch, Edward Cutrell, Ken Hinckley, and Adam Eversole
Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
{baudisch, cutrell, kenh, adame}@microsoft.com

ABSTRACT
Snapping is a widely used technique that helps users position graphical objects precisely, e.g., to align them with a grid or other graphical objects. Unfortunately, whenever users want to position a dragged object *close* to such an aligned location, they first need to deactivate snapping. We propose *snap-and-go*, a snapping technique that overcomes this limitation. By merely stopping dragged objects at aligned positions, rather than "warping" them there, snap-and-go helps users align objects, yet still allows placing dragged objects anywhere else. While this approach of inserting additional motor space renders snap-and-go slightly slower than traditional snapping, snap-and-go simplifies the user interface by eliminating the need for a deactivation option and thereby allows introducing snapping to application scenarios where traditional snapping is inapplicable. In our user studies, participants were able to align objects up to 138% (1D) and 231% (2D) faster with snap-and-go than without and snap-and-go proved robust against the presence of distracting snap targets.

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

Keywords: Alignment, snapping, snap-dragging, mouse input, pseudo haptics.

INTRODUCTION
Sometimes users need to perform *precise* graphical manipulations. As shown in Figure 2, users increase visual clarity by aligning graphical objects with each other or by scaling table columns to the same width. They align audio and video segments to assure synchronicity and they make precise selections in bitmap images to make sure subsequent filters get applied to the right areas.

Obtaining precise results by dragging or scaling an object with the mouse requires considerable motor skill. Therefore, many computer applications help users align objects. *Snapping* (e.g., *snap-dragging* [5]) provides aligned positions with an attraction behavior sometimes described as "magnetism" [3] or "gravity" [5]. Figure 1a illustrates this with an example of a slider with a single snap location.

CHI 2005, April 2–7, 2005, Portland, Oregon, USA.

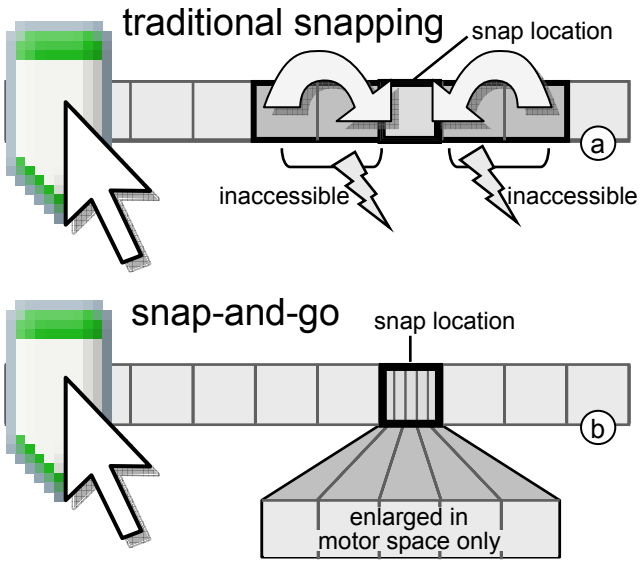

Figure 1: (a) The problem: Traditional snapping warps the knob of this slider to the target whenever close, making it impossible to place the knob in the areas marked inaccessible. (b) The proposed solution: Snap-and-go inserts additional motor space at the snap location, thereby keeping all slider positions accessible.

Whenever the user drags the knob into the area surrounding the snap location the knob is automatically "warped" to the snap location. Given that this attraction area is larger than the snap location itself, here five pixels instead of one, the alignment task is simplified significantly.

The downside of snapping, however, is that this magnetic behavior can get in the user's way. While users are often likely to use the recommended snap locations, there are cases where users want to place a dragged object elsewhere. Continuing the examples from Figure 2: (a) To align the *baselines* of these text fragments, a user may need to move the right one down *a little*, but the grid keeps holding it back. (b) To fit a slightly longer word into this table cell, the user may need to widen that column just a bit, but it scales in steps causing the next column to overflow. (c) The user tries to make space to allow the following voice over clip start *a little before* the matching video segment, but scaling either snaps back or scales too much. (d) To remove the frame around this picture, the user tries to create a selection around it, but the lasso either snaps to the entire image or leaves at least two pixels out. In these and other scenarios, traditional snapping prevents users from accomplishing their task, every time the user tries.

In the following sections, we take a closer look at why this is problematic; present the *snap-and-go* alignment technique and explain how it avoids this problem; go over the related work; and report three user studies in which snap-and-go was found to improve participants' speed when aligning objects on the screen. We conclude with a summary of our findings and an outlook on future work.

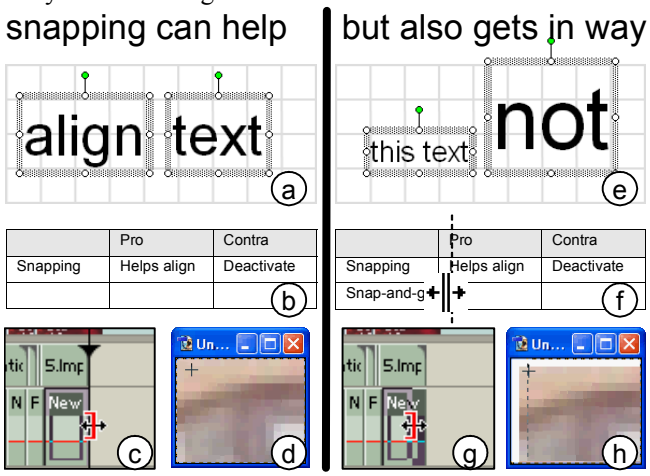

Figure 2: Traditional snapping helps (a) align objects in a graphics editor, (b) created equally sized columns, (c) align audio clips with video clips, and (d) assure that the entire bitmaps was selected, (e-h) but gets in the way when users editing goals are not foreseen by the application designer.

Traditional snapping requires deactivation

In order to enable the *non*-alignment tasks above, traditional snapping requires application designers to provide snapping objects with an additional user interface that allows users to *disable* snapping. This, however, turns out to be more complex than expected.

A common approach is to allow users to de/activate snapping by holding down a modifier key. However, the modifier key approach is inapplicable if (1) the application is already using all modifier keys for other purposes, such as for switching tool options (e.g., Adobe Premiere), (2) there are more snapping constraints than modifier keys (e.g., Microsoft Visio), or (3) the target audience are non-experts and cannot be expected to discover modifier keys.

In these cases, application designers are forced to revert to offering on-screen controls, such as a checkbox in a context menu (Figure 3a). To improve discoverability, some application designers place checkboxes right into the visible user interface, even though this adds to the complexity of the interface (Figure 3b). In case there are too many snapping options, deactivation ends up in an options dialog (Figure 3c), again with low discoverability.

As the participants of our user study confirmed, snapping is a useful and highly appreciated function. It saves users time with every use and enables users with limited motor skills to perform tasks they otherwise could not perform at all. The price, however, is additional user interface complexity—potentially an additional checkbox per snapping function and time spent using it. *Snap-and-go* is designed to overcome this limitation.

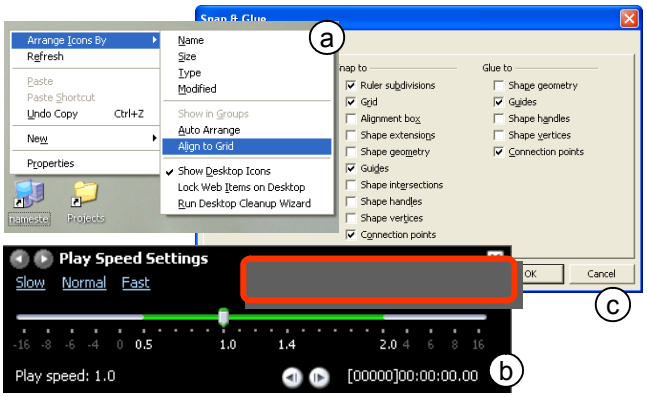

Figure 3: Checkboxes for deactivating snapping in (a) Windows XP, (b) Media Player (c) and Visio.

SNAP-AND-GO

Snap-and-go is a snapping method that does not require deactivation. Figure 1b illustrates this with the example of a slider. Instead of reassigning motor space to snap locations, snap-and-go inserts *additional* motor space at the snap locations. This is done by reducing the input gain of the mouse while over the target. Rather than seeing the knob jump to the target, users feel that the mouse temporarily stops at the target—thus the name of the technique, derived from the expression *stop-and-go*. Visual feedback (e.g., Figure 14b) informs users about successful alignment.

Unlike traditional snapping, snap-and-go does not require deactivation to allow users to avoid alignment. It achieves this by managing motor space differently. Traditional snapping requires a deactivation option because its approach of *redistributing* motor space to the snap location leaves other locations with no representation in motor space, which renders these locations inaccessible. Snap-and-go's approach of *inserting* additional motor space at the snap location leaves the motor space of all other locations intact.

Replacing traditional snapping with snap-and-go allows users to enjoy snapping functionality without the need to learn about modifier keys or to sacrifice screen space for checkbox interfaces (Figure 3). In addition, snap-and-go can be applied to applications that do not offer snapping today, such as to help users center audio balance or to drag the time slider in a DVD player to the beginning of a chapter. These scenarios have no space for a deactivation interface, which is why traditional snapping has never been applied to them. In its current version, snap-and-go is limited to platforms using an *indirect* input device—*direct* input devices, such as pens or touch input do not allow creating extra motor space.

SNAP-AND-GO IN 2D

The slider example given in the previous section is limited to one-dimensional drag interactions. Snap-and-go in 2D is similar to the 1D case in that they both insert additional motor space to hold dragged objects at aligned locations. In addition, however, alignment in 2D requires *guiding* dragged objects to aligned positions. While dragging an object in 1D will inevitably cross all locations between

start and end position, dragging an object in 2D traverses only *one* path and therefore cannot guarantee that a dragged object will ever get to the aligned position in the first place. Since warping was the reason why traditional snapping required deactivation, we need an alternative mechanism for bringing the dragged object to the target.

Snap-and-go in 2D is based on two basic widgets that application designers can overlay over screen objects to help users align objects with them. Similar to the 1D case, these widgets manipulate motor space, but they contain additions to guide dragged objects to snap locations. To provide a basis for explaining the actual snap-and-go widgets (Figure 5), we start by looking at the underlying concepts and evolution history. These are illustrated by Figure 4.

(a) The problem: The user's task is to align the partially visible gray square at the bottom right with the two fully visible gray squares. This requires placing the square at the location marked with a dashed outline. The user's drag direction (shown as a black arrow) would miss the aligned position.

To make the following steps easier to illustrate, we switch to the visuals of a target acquisition task. We paint a knob labeled onto the dragged object and a matching socket labeled onto the position where the knob has to go in order to align the objects. Both are only for the purpose of illustration and are not visible to the user.

(b) We add an invisible "funnel" over the socket. The "funnel" consists of two invisible guides. As the user drags the square, the knob now hits the funnel and slides along that funnel until the knob gets trapped at the socket in the funnel's center and the user is provided with visual feedback confirming alignment (e.g., Figure 18).

The funnel widget simplifies 2D alignment for two reasons. First, instead of having to simultaneously steer the mouse to precise x and y coordinates, the funnel widget requires users to only hit the entrance of the funnel and to follow through. The funnel thereby turns the 2D acquisition task the user would normally have to perform into a 1D acquisition task, also known as "crossing task" [1]. Second, the new target of the user's task, i.e., the entrance of the funnel is significantly bigger than the single-pixel aligned position that the user would otherwise have to acquire. Application designers can configure this width by choosing a funnel of appropriate size.

(c) Aligning the funnel and extending it into a cross. Dragged objects reach the funnel center fastest when making contact with the funnel at a more obtuse angle. While an obtuse angle can be guaranteed by rotating the funnels towards the mouse pointer, we choose a stationary funnel aligned with the coordinate system of the screen. The benefit of this is that it helps users align the dragged object in x or y or both, which is often useful. Duplicating the funnel for all four directions forms a *cross widget* (made of a horizontal and a vertical *guide*), which allows users to align objects dragged when coming from different directions.

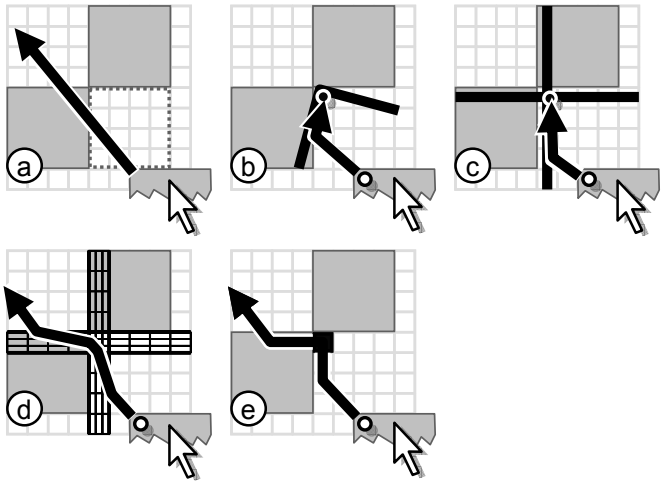

Figure 4: Evolution of snap-and-go in 2D

(d) Minimizing side-effects by using 'frixels': When trying to drag an object to a *different* location on the screen, users may accidentally hit a cross widget on the way. To minimize disturbance, we replace the solid guides of the cross widget with permeable ones. These are created using pixels that slow drag motion down—we call this *friction*. To obtain the desired guidance effect we need pixels with *directional friction* (or "*frixels*"). The vertical guide in this example consists of frixels with a horizontal friction of 3 and a vertical friction of 1, shown as pixels subdivided into three thinner subpixels. Crossing such a frixel with the mouse requires the user to move the mouse horizontally three times as far than a normal pixel would. This causes the path of the dragged object to be *bent*, guiding it through the snap location at the funnel center. At the funnel center, guides overlap and their friction values add up, here resulting in a center frixel with 3x3 friction. This frixel holds the dragged object for a moment, which helps users release it.

(e) Frixel-based widgets create visual effects similar to solid widgets. To obtain a precise trajectory, snap-and-go tracks the position of the dragged object in terms of subpixels. This extra information, however, is not presented to the user; a user dragging an object through a frixel widget sees the dragged object progress in terms of complete pixels. Figure 4e shows how the trajectory from Figure 4d appears to the user as a series of discreet events: the dragged object latches onto the vertical guide, it slides up two pixels where it reaches the aligned position, gets held there for a moment, and then slides along the horizontal guide for two pixels until it breaks free.

We are now ready to explain the two widgets that form the basis of snap-and-go in 2D. The *plus widget* shown in Figure 5a is the basic widget for simultaneous alignment in x and y. It is based on the cross widget from Figure 4d, but it is of finite size and its friction is faded out towards the periphery to minimize the impact on trajectories not aiming for this target. The *bar widget* shown in Figure 5b is designed for alignment in only one axis. It is made of traditional, non-directional friction, which avoids the here unnecessary sideway-motion that directional frixels introduce.

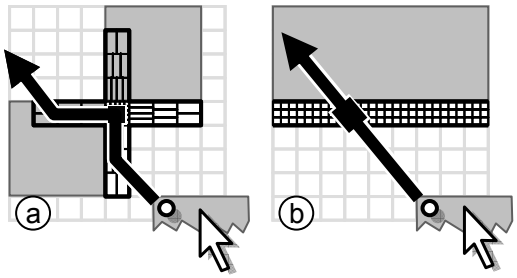

Figure 5: The basic widgets of snap-and-go in 2D are (a) plus widget and (b) bar widget.

More complex widgets can be created by combining basic widgets. The example shown in Figure 6a helps users snap a connector line to a rectangle on screen. While the corresponding object based on traditional snapping (Figure 6b) allows users to connect *only* to edge centers (which can lead to undesired "elbows"), the snap-and-go widget allows users to *also* connect anywhere along the edge.

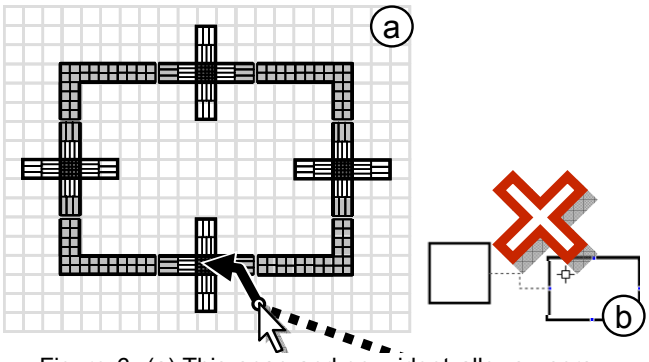

Figure 6: (a) This snap-and-go widget allows users to connect a connector line to edge centers, but also anywhere else on its edges. (b) The same widget based on traditional supports only edge centers.

The example widgets shown in this section feature friction values between 2 and 5. We chose these values to assure the readability of the diagrams. While friction values that low are possible (and we found significant effects for friction values as low as two), we found most users to prefer higher friction values (20-30, see the user study sections).

RELATED WORK

Snap-and-go builds on three areas of related work, i.e., alignment techniques, constraints and snapping, and manipulation of motor space/mouse gain.

Alignment techniques in related work fall into two main categories. The first category contains techniques that are applied post-hoc, similar to traditional menu or toolbar-based alignment functions: users pick two or more objects and then choose a function, such as "stack vertically" from a toolbar. The Alignment Stick [23] allows aligning objects by pushing a ruler against both objects—the moment the second object starts moving both objects are aligned. This approach can reduce the need for selecting alignment functions repeatedly and thus save user effort.

The second category consists of techniques that are applied while dragging the object to be aligned. The original snap-dragging technique by Bier [5] allows users to create and place alignment objects; subsequently placed graphical objects then automatically snap to these alignment objects. By avoiding the need for an extra interaction step, snapping eliminates the overhead faced by explicit alignment functions. Various researchers have added to the concept of snap-dragging by extending it to 3D [6, 2], adding anti-gravity feedback to inform users when attempting to create an illegal connection [13], or changing snapping grids while dragging objects (*HyperSnapping* [20]). Snapping has been applied to a wide range of applications, including snapping and zipping windows together [4]. A particularly simple and thus widely used alignment object is the grid. The *CAGE* [2] extends grids such that they allow aligning graphical objects with *each other*.

Another way of aligning objects is to describe the desired goal state using constraints [8]. Constraints are supported by a variety of toolkits, such as *Juno* [22] and their use reaches back as far as to Sutherland's *Sketchpad* [24]. While initially created with text interfaces, some systems allow users to established constraints using snap-dragging (*augmented snapping* [11]). Similarly, [3] allows users to manipulate aligned groups without giving up alignment. Other researchers suggested creating alignment behavior *by demonstration* [18] or by generating several aligned versions and letting users pick (*suggestive interfaces* [14]).

Snapping and constraints restrict the space where objects can be placed. Since this leads to the aforementioned problem of inaccessible space, snap-and-go inserts additional motor space instead. Manipulation of motor space has been studied in the field of *pseudo haptics* [17] and can be applied to any indirect pointing device. Lécuyer et al. showed that changing the coupling between mouse motion and mouse pointer motion can be used to simulate the haptic sensation of texture [16].

Changes in the mouse-to-pointer gain have also been used to help users overcome long distances and to acquire small targets, e.g., *object pointing* [12], and *lay lines* [15]. Expanding targets (in screen space and motor space) [21] was found to help users acquire small targets. In combination with an *area cursor*, making targets "sticky" was found to help users with motor disabilities acquire small targets with the mouse (*sticky icons* [27], *semantic pointing* [7], also suggested by [25]). Unlike these methods, snap-and-go offers a method for guiding the user to very small targets, as we will discuss in more detail at the end of the following section.

SNAP-AND-GO FOR TARGET ACQUISITION

Target acquisition techniques are relevant to the topic of alignment, because an alignment operation can be reduced to a target acquisition task, (e.g., the "Snap-and-go in 2D" section above). Based on this similarity, we created an adapted version of snap-and-go that serves as a target acquisition aid. As shown in Figure 7a, the plus widget remains the same, only the visuals change: Rather than guiding a *dragged object* to an *aligned position*, the cross wid-

get now guides the *mouse pointer* to the *target*. Note that plus widgets always guide the pointer to the target center, thus work across target sizes (Figure 7b).

The snap-and-go target acquisition technique offers benefits similar to the snap-and-go alignment technique: it allows users to distinguish between multiple targets in close proximity, where other techniques, such as snap-to-target or area cursor [27] fail to distinguish between them.

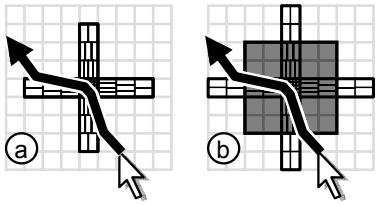

Figure 7: Snap-and-go as a target acquisition aid.

Note that the inverse is not true: a technique originally designed to be a target acquisition technique *cannot* necessarily serve as an alignment technique. Enhancing an alignment position with a sticky icon as shown in Figure 8a turns the acquisition task into a crossing task, which can simplify target acquisition [1]. Crossing that pixel, however, is still hard. Note that sticky icons cannot be used to make a cross widget (Figure 8b); their *non*-directional friction will *not* guide the dragged object to the target. Figure 8c and d illustrate the difference: A sticky icon measuring only one pixel is easily missed. Snap-and-go guides the pointer or dragged object into the target, thereby creating a fisheye effect in motor space.

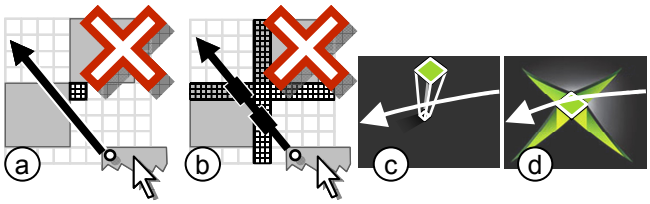

Figure 8: (a, b) Attempt to use sticky icons to align objects in 2D. (c) Sticky icons vs. (d) snap-and-go

IMPLEMENTATION

We created two implementations of snap-and-go, i.e., a complete implementation in C# and a reduced prototype in Macromedia Flash that we use for running user studies. The Flash version supports 1D snap-and-go and the simplified version of 2D snap-and-go shown in Figure 4d. Figure 9 illustrates how this prototype implements snap-and-go by subtracting friction at snap locations from traversed distances. A simple 2D cross widget can be obtained by running this code on *x* and *y* coordinates. (Note that this code is abbreviated for space reasons; it misses code for updating the mouse pointer to keep knob and pointer together, etc.).

The C# version supports the more advanced versions of 2D snap-and-go described in this paper. The code is based on rectangular friction objects, each of which defines a friction gradient of configurable direction and strength. By combining multiple friction objects application designers can create arbitrary friction widgets, including the plus and the bar widgets and the example shown in Figure 6. By integrating friction along the interpolated pointer path, the program assures all traversed friction widgets will take effect, even in cases where the stepwise nature of mouse motion causes the pointer to jump over a widget without actually touching it. Pointer position is tracked in subpixels. This assures that users can traverse friction widgets even very slowly and at flat angles. Rounding errors would otherwise cause progress across the widget to continuously be rounded to zero, causing frixel widgets to appear solid.

```
snapTo(x, w, snapX) {
    if (snapAndGoActive) {    // snap-and-go
        if (x >= snapX + w)
            return x - w + 1;
        else if (x > snapX)
            return snapX;
        else return x;
    } else {                  // traditional snapping
        if (x > snapX - w/2 && x < snapX + w/2)
                return snapX;
        else return x;
    }
}
```

Figure 9: Code fragment for 1D snap-and-go with a single snap location of width w located at snapX (top) in comparison to traditional snapping (bottom). The function returns the location of the dragged knob in dependence of the pointer position.

USER STUDIES

To objectively evaluate performance using snap-and-go, we performed a series of three user studies. The participants' task in all three studies was to align a dragged object with a target location with pixel-accuracy. The studies differed in whether there was a single attractor at the target or multiple attractors, and whether alignment took place in one or two dimensions (Table 1).

	Snap-and-go compared to traditional snapping…	Snap-and-go with distractors…
…in 1D	Study 1	Study 2
…in 2D	Study 3	

Table 1: Scope of the 3 studies reported below

USER STUDY 1: SNAP-AND-GO VS. SNAPPING IN 1D

The purpose of this first study was to verify that snap-and-go indeed helps users align objects, to explore the impact of attractor strength on task time, and to compare snap-and-go with traditional snapping.

Task

The participants' task was to drag the knob of a slider to a highlighted target location as quickly as possible. Figure 10 shows the apparatus, which consisted of a horizontal slider with a single highlighted target location, which in some conditions was complemented with an *attractor* (see below). For each trial the slider was reinitialized to the shown state; target distance and attractor, however, were varied.

Task time was counted from the moment the knob was picked up until the moment the knob was successfully aligned and the participant had released the mouse button. Each trial required successful alignment, so in cases where participants released the knob anywhere but over the target, they needed to pick it up again to complete the trial.

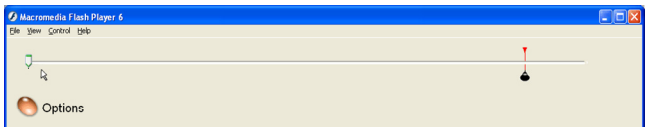

Figure 10: The apparatus. The user's task was to align the slider knob located at the left with the target located at the right.

Alignment required pixel precision. To make that possible the knob was provided with the visuals shown in Figure 11a. To prevent the mouse pointer from occluding the target, participants were encouraged to drag the mouse slightly downwards while dragging the knob (Figure 11c).

Interfaces

There were three main interface conditions, namely *traditional snapping* and *snap-and-go*, implementing the two snapping functionalities illustrated by Figure 1, as well as no snapping.

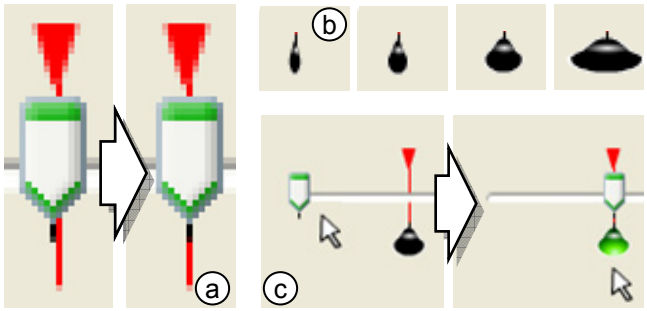

Figure 11: Close-up of the knob reaching the target: A black dash at the bottom of the knob helped visually verify alignment. (b) Attractors used "light bulb" visuals and came in four sizes. (c) Dragging the knob into an attractor caused it to light up.

In the no snapping condition, the target consisted only of the vertical red line shown in Figure 11a. In the snapping conditions, the target was complemented with an attractor, turning the target into a snap location. Attractors behaved differently depending on the snapping condition, but offered the same visuals, a "light bulb" located below the slider (Figure 11b and c). In their inactive state light bulbs were black, but turned to bright green when the knob was captured. To inform participants during the study about the current attractor strengths, the width of the bulb on screen reflected the width of the attractor in motor space (Figure 11a). Interface conditions thus differed in interactive behavior and visuals.

Experimental design

The study design was within subjects 2 x 4 x 4 (Snapping Technique x Attractor Width x Target Distance) with 8 repetitions for each cell. Distances were 100, 200, 400, and 800 pixels, and Widths 5, 10, 18, and 34 pixels. In addition, participants performed 2 blocks of trials with snapping off at each distance. For each trial, we recorded task completion time and error, i.e., number of times the participant dropped the knob before aligning it properly. Interface order, distances, and sizes were counterbalanced.

Participants received training upfront and at the beginning of each block. The study took about 35 min per participant.

Apparatus

The experiment was run on a PC running WindowsXP with an 18" LCD monitor, at a resolution of 1280x1024 pixels and 60Hz refresh rate, and driven by an nVidia graphics card. The interface used in this study was implemented in Macromedia Flash; its functioning was briefly described in the Implementation section of this paper. The optical Microsoft IntelliMouse was set to a medium mouse speed and participants were allowed to adjust it prior to the beginning of the study.

Participants

Nine volunteers, (7 male) between the ages of 25 and 50 were recruited from our institution. Each received a lunch coupon for our cafeteria as a gratuity for their time. All had experience with graphical user interfaces and mice; three were trackball users. All were right-handed.

Hypotheses

We had three hypotheses: (1) Participants would perform faster with snap-and-go than with no snapping. (2) Stronger attractors and shorter distances would reduce task time. (3) Due to the additional distance in motor space, participants should be slightly slower when using snap-and-go then when using traditional snapping. However, we expected the difference to be small.

Results

To correct for the skewing common to human response time data we based our analyses on the median response time across repetitions for each participant for each cell.

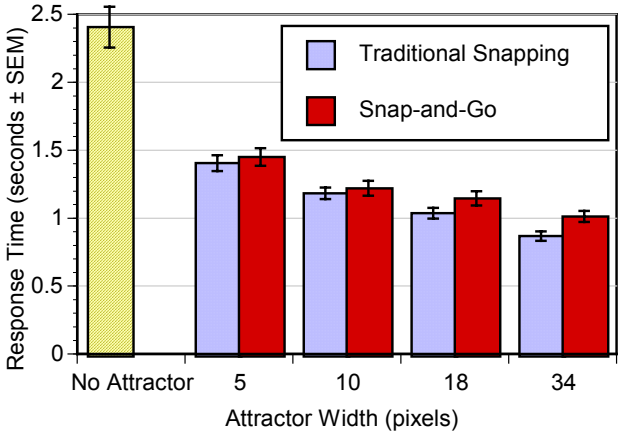

Figure 12: Task time by snapping technique and attractor width across all distances (+/- standard error of the mean).

Snapping vs. no snapping: We compared the most conservative case for the two snapping conditions (attractor size = 5) against no snapping for each distance. We performed a 3 x 4 (Snapping Technique x Target Distance) within subjects analysis of variance. There were significant main effects for both Snapping Technique, $F(2,16)=66.3$, $p<<0.01$ and for Target Distance, $F(3,24)=20.19$, $p<<0.01$. Planned comparisons of no snapping vs. traditional snapping and vs. snap-and-go were also significant, $F(1,8)=76.7$, $p<<0.001$ and $F(1,8)=61.5$, $p<<0.01$ respectively.

Snap-and-go vs. traditional snapping: We performed a 2 x 4 x 4 (Snapping Technique x Attractor Width x Target Distance) within subjects analysis of variance. We found significant main effects for each factor. As expected, traditional snapping was faster than snap-and-go $F(1,8)=24.0$, $p<0.01$. Also as expected, differences were fairly small, ranging from 3% for attractor widths 5 and 10 to 14% for attractor width 34 (Figure 12)

Impact of attractor width and distance on task time: Not surprisingly, there were significant effects for Attractor Width, $F(3,24)=97.6$, $p<<0.01$ and for Target Distance, $F(3,24)=224.4$, $p<<0.01$; performance improved as attractor width increased and as target distance decreased. There were no significant interactions. Given the similarity to Fitts' Law experiments, we compared user performance against the Fitts Index of Difficulty (ID), a metric that combines target width and movement distance into one measure of acquisition difficulty [19, 10]. Figure 13 plots mean movement time for each ID for the two snapping techniques. The regression of movement time against ID for each snapping technique was:

Snapping: $MT=0.265 + 0.19*ID$, $r^2=0.75$

Snap-and-go: $MT=0.487 + 0.159*ID$, $r^2=0.59$

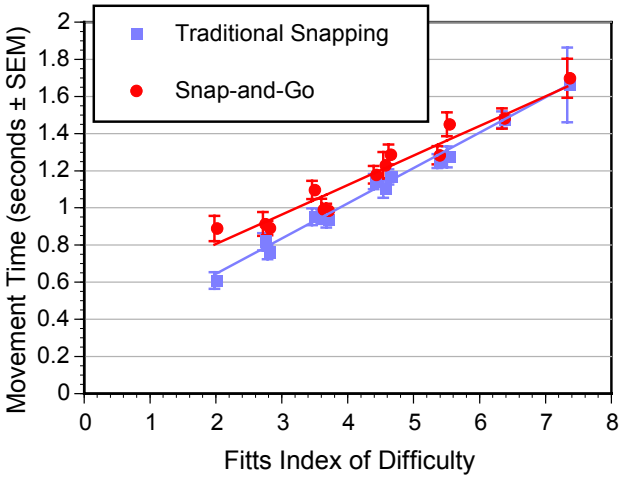

Figure 13: Fitts analysis of task times.

Note that the main divergence is at very low indices of difficulty. Once the task gets harder (e.g., longer movements, smaller attractor sizes) performance in the two techniques begins to converge.

Error rates were generally low, indicating that the target and knob visuals did allow participants to visually validate alignment sufficiently well. Differences in the error rates for the three different snapping conditions were non significant (No Snapping: 6.1%, Traditional Snapping: 3.7%, Snap-and-Go: 2.6%).

Across snapping methods, eight of nine participants indicated a preference for the stronger 34 and 18 width attractors; one participant preferred the weakest attractor strength included in the study (5).

USER STUDY 2: IMPACT OF DISTRACTORS

Experiment 2 was designed to investigate two questions. First, in the case of multiple potential targets (e.g., chapters along a DVD player time slider) should an application designer provide each chapter with an attractor or would the additional attractors (*distractors*) get in the user's way? Second, even with multiple attractors being present, users might at least occasionally want to target a non-enhanced location. How will distractors affect that?

Task and interface were the same as in User Study 1, except for the following two differences. First, in half of the trials there was an attractor over the target, while in the other half there was not. Second, there were up to four "distractors" located in front of and behind of the target as shown in Figure 14 (distances 64, 32, and 16 pixels in front, and 16 pixels behind target, see Figure 14).

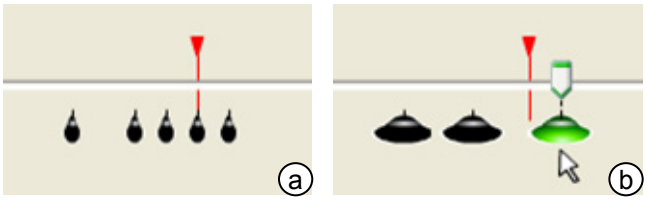

Figure 14: Attractor/distractors: (a) Attractor at target and all four distractors (width 10). (b) No target attractor, but three of the four distractors.

In this study, we only included snap-and-go but *not* traditional snapping. The reason is that for some distractor combinations traditional snapping would have required deactivation (e.g., Figure 14b) and deactivation interfaces were outside the scope of this study.

To keep the overall study time manageable, we limited the design to a single distance and two attractor widths. The resulting design was a (2 x 2 x 2 x 2 x 2 x 2) (Target Attractor on x Distractor 1 x Distractor 2 x Distractor 3 x Distractor 4 x Attractor Width) with 4 repetitions for each cell.

Nine participants hired from the community (6 male) between the ages of 18 and 60 participated in the study. All were right-handed.

Results

The full factorial analysis was very difficult to interpret because of interaction effects. Initial results showed no significant effect for distractor position, so we simplified the analysis by combining the four binary Distractor vari-

ables into a single variable, Number of Distractors (0, 1, 2, 3, or 4). We then did two analyses: one for when the Target Attractor was on, and one for when it was off.

Target with attractor: We performed a 5 x 2 (Number of Distractors x Attractor Width) within subjects analysis of variance. As expected, there was a significant effect for the Number of Distractors, $F(4,32)=5.1$, $p<0.01$ (Figure 15). Interestingly, in this condition, there was no main effect for the width of the attractor or an interaction. This might indicate that the additional *attraction* caused by a larger attractor was at least partially compensated for by the equally larger *distraction* caused by larger distractors.

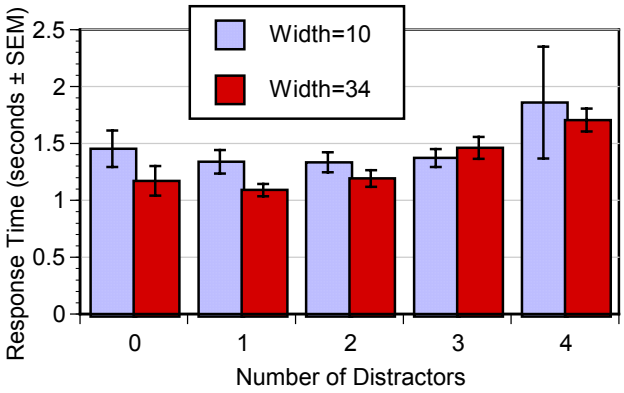

Figure 15: Task times *with* attractor at target and 0-4 distractors.

Target without attractor: We performed the same analysis for trials in which the target had no attractor, but there were still up to four distractors. This time the only significant effect was for Attractor Width (of the distractors). Unlike the case with an attractor at the target, there is no tradeoff here, so larger distractors impacted task time more. There was no main effect for the Number of Distractors or a significant interaction (Figure 16).

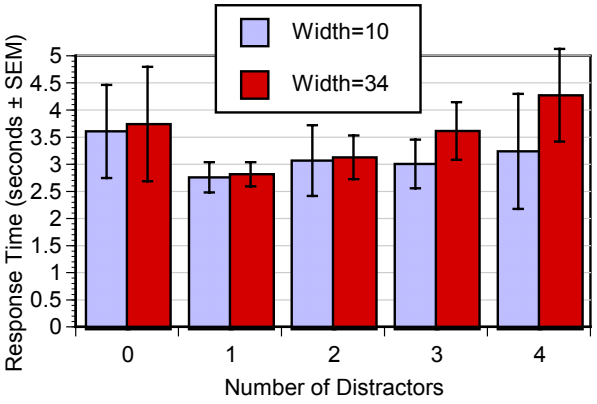

Figure 16: Task times *without* attractor at target and 0-5 distractors

These results provide some indication for deciding how many locations to provide with an attractor: successfully providing a target with an attractor saves users an average of 1.9 seconds per alignment interaction; the addition of four snapping locations not targeted by the user causes a penalty of less than 0.5 seconds.

USER STUDY 3: SNAP-AND-GO IN 2D

In our third study, we replicated the first two experiments for an alignment task in two dimensions. In this experiment each participant performed two tasks, the first being the 2D equivalent of the first user study, and the second being the 2D equivalent of the second user study. The study took participants about an hour to complete.

Task and interfaces

In both tasks, the participants' task was to align a square with two other squares as illustrated by Figure 17.

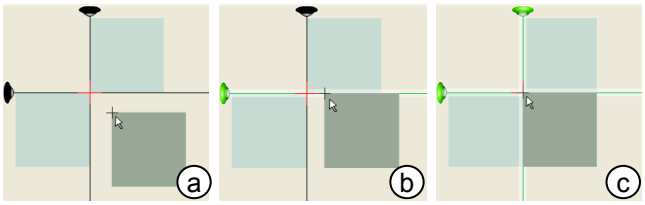

Figure 17: Task 1 of the 2D study. Participants aligned the dark square with the two lighter squares

Figure 18a illustrates the individual elements on the screen. Depending on condition, there were up to six line-shaped attractors. Three vertical attractors at the target location and 10 and 60 pixels right of it, three horizontal attractors at and below the target location. In the snap-and-go conditions attractors implemented the simplified snap-and-go behavior described in Figure 4d. Upon contact with the edge of the dragged rectangle attractor lines changed colors from black to green and displayed a white, 10-pixel wide halo. A light bulb at the end of each line informed participants about the current attractor strengths. The top left corner of the dragged rectangle was provided with a cross of 1-pixel lines to help visually verify alignment.

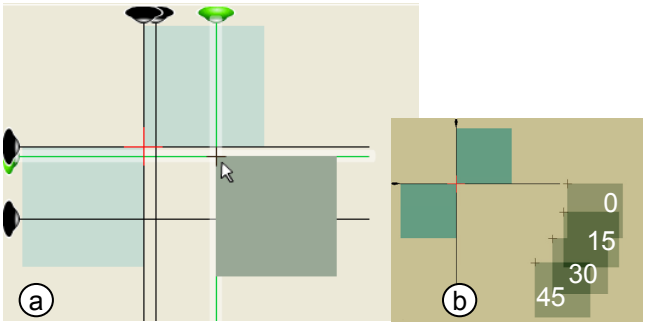

Figure 18: (a) Apparatus used in the 2D study. (b) The four start locations of the square.

Task 1: In correspondence to the first user study, only the two attractors aligned with the target were present (Figure 17). The design was within subjects 2 x 4 x 4 (Snapping Technique x Attractor Width x Approach Angle) with 8 repetitions for each cell. Distance to the target was 200 pixels and approach angles were 0, 15, 30, and 45 degrees, as shown in Figure 18b. Participants also performed 2 blocks of trials with snapping off at each distance.

Task 2: The design was within subjects 2 x 2 x 2 x 2 x 2 x 2 (Target Attractor Vertical x Target Attractor Horizontal x Distractor Vertical10 x Distractor Horizontal10 x Distrac-

tor Vertical60 x Distractor Horizontal60) with 4 repetitions per cell. This means that the target was enhanced with only the horizontal attractor, only the vertical attractor, both, or none. In addition, up to four distractors were located at the locations described above. Drag distance was always 200 pixels and all trials started at the 30 degree approach angle. To prevent sequence effects on Task 1 caused by unbalanced training, all participants performed Task 1 first.

Participants: Eleven volunteers (11 male) recruited internally participated. The average age was 31 years (std dev 8.0); all were right-handed.

Hypotheses corresponded to the 1D case. However, the cross widget used in the study causes more sideways drift than a combination of plus and bar widgets. We therefore expected distractors to have a slightly bigger impact.

Results

Task 1: snap-and-go vs. traditional snapping

As in previous experiments, we based our analyses on the median response time across repetitions for each participant for each cell. We performed a 2 x 4 x 4 (Snapping Technique x Attractor Width x Approach Angle) within subjects analysis of variance.

Traditional snapping was significantly faster than snap-and-go, $F(1,10)=13.1$, $p<0.01$ (Figure 19), and there was a significant effect for Attractor Width, $F(3,24)=97.6$, $p<<0.01$; performance improved as attractor width increased. There were no significant interactions or main effect for Approach Angle.

Performance vs. no snapping

We compared the most conservative case for the two snapping conditions (attractor size = 5) against no snapping for each distance. We performed a 3 x 4 (Snapping Technique x Target Angle) within subjects analysis of variance. There was a significant main effect for Snapping Technique, $F(2,20)=67.7$, $p<<0.01$. Planned comparisons of no snapping vs. traditional snapping and vs. snap-and-go were also significant, $F(1,10)=85.4$, $p<<0.001$ and $F(1,10)=60.0$, $p<<0.01$ respectively. See Figure 19. Again, there was no significant effect for Target Angle.

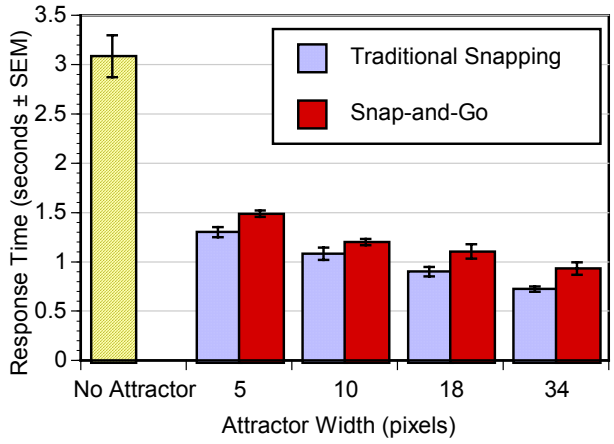

Figure 19: Task times in 2D by snapping method and attractor width.

Again, error rates were generally low. Differences in the error rates for the three different snapping conditions were not significant (No Snapping: 5.1%, Traditional Snapping: 3.7%, Snap-and-Go: 4.5%).

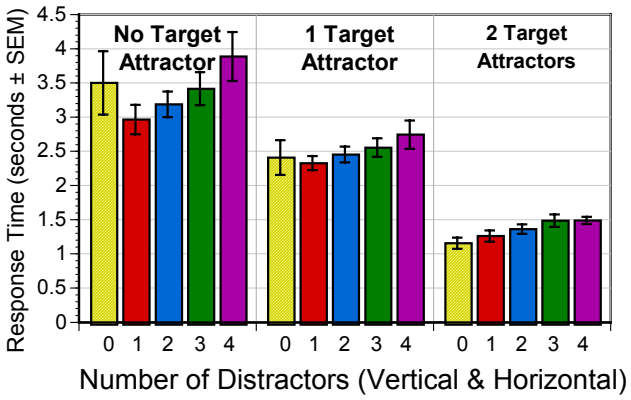

Figure 20: Response times for 2D distractor task with 0-4 distractors for each level of target attractor.

Task 2: 2D Distractor Task

As in experiment 2, the full factorial analysis was very difficult to interpret because of interaction effects. Initial results showed no significant effect for distractor position, so we simplified the analysis by combining the four binary Distractor variables into a single variable, Number of Distractors (0, 1, 2, 3, or 4). In addition, the two Target Attractor variables were reduced to 1, Number of Attractors (0 to 2).

We performed a 3 x 5 (Number of Target Attractors x Number of Distractors) within subjects analysis of variance. As expected, there was a significant effect for the Number of Distractors, $F(4,40)=3.7$, $p<0.01$, with movement time steadily increasing with the number of distractors (Figure 20).There was also a large main effect for Number of Target Attractors, $F(2,20)=69.8$, $p<<0.01$. As the number of target attractors move from 0 to 1 and then up to 2, performance improved markedly. There were no significant interactions.

DISCUSSION

In the three studies presented above, we covered snap-and-go in one and two dimensions. Across conditions, participants were significantly faster with snap-and-go than without snapping support. For the largest attractor width speed-ups were 138% in 1D and 231% in 2D. As expected, the additional motor space snap-and-go caused it to be slightly slower then traditional snapping, with differences of 3% in 1D and 14% in 2D. Snap-and-go turned out to be fairly robust against the presence of distractors.

Resulting design improvements

With traditional snapping, dragged objects are visibly warped when latching-on. During the first 1D study, some participants expressed how this visual cue helped them verify alignment. We therefore created a version of snap-and-go visuals that emulates warping using *anticipation behavior* [26, 9]: when latching on, the knob briefly over-

shoots and returns to the snap position; similarly, the knob first moves backwards when breaking free. We also created a version were the knob behaves as if dragged over little vertical barriers left and right of the snap location.

We also observed that switching from a traditional snapping condition to a snap-and-go condition caused some participants to converge particularly slowly towards the snap location—waiting for it to latch on. Since snap-and-go requires users to drag the knob *past* the apparent snap location, hesitant dragging continued all the way to the actual snap location, which affected task time. Figure 21 shows a redesign with corrected visual affordance. Here attractors are displayed *behind* the snap location. At the expense of introducing additional motion, this design also updates users about their location in motor space by moving the attractor against the mouse motion as the user passes it.

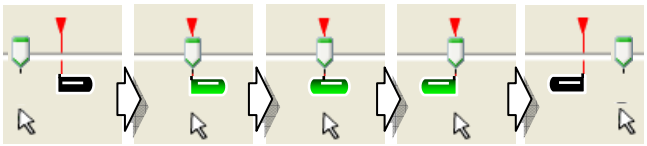

Figure 21: Redesigned attractor visual that always appears *behind* the target.

CONCLUSIONS

In this paper, we presented snap-and-go, an alignment technique that—unlike traditional snapping—does not require deactivation. While slightly slower than traditional snapping, the ability to omit the deactivation interface allows deploying snap-and-go in application areas where additional interface complexity would be prohibitive.

We made three main contributions. First, we demonstrated how manipulations of mouse gain can help users align objects. Second, we extended our technique to 2D by introducing the plus and the bar widgets that guide dragged objects to snap locations. And third, we presented three user studies evaluating snap-and-go in 1D and 2D in comparison with traditional snapping and no snapping.

As future work we plan to extend the snap-and-go concept to indirect pointing devices, such as pen and touch input.

ACKNOWLEDGMENTS

We would like to thank Cameron Etezadi, Gavin Jancke, Jordan Schwarz, and Phil Fawcett for their support of the snap-and-go project, as well as George Robertson, Maneesh Agrawala, Andy Wilson, and Noah Snavely for their comments on a draft of this paper.

REFERENCES

1. Accot, J., and Zhai, S. More than dotting the i's Foundations for crossing-based interfaces. In *Proc. CHI'02.* pp. 73–80.
2. Baudisch, P. The Cage: Efficient construction in 3D using a cubic adaptive grid. In *Proc. UIST'96,* pp. 171–172.
3. Beaudouin-Lafon, M. & Mackay, W. Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces. In *Proc. AVI'00,* p.102–109.
4. Beaudouin-Lafon, M. Novel Interaction Techniques for Overlapping Windows. In *Proc. UIST'02.* pp 153-154.
5. Bier, E. and Stone, M. Snap dragging. In *Proc. SIGGRAPH'86,* pp. 233–240.
6. Bier, E. Snap-dragging in three dimensions. In *Proc. 1990 Symposium on Interactive 3D Graphics,* pp. 193–204.
7. Blanch, R. Guiard, Y., Beaudouin-Lafon, M. Semantic Pointing: Improving Target Acquisition with Control-Display Ratio Adaptation. In *Proc. CHI'04*, pp. 519–526.
8. Borning, A. Defining constraints graphically. In *Proc. CHI 86*, pp. 137–143.
9. Chang, B.-W. and Ungar, D. Animation: From Cartoons to the user interface. In *Proc. UIST'93*, pp. 45–55.
10. Fitts, P., The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement, *Journal of Experimental Psychology,* v 47, June 1954, pp. 381–391.
11. Gleicher, M. and Witkin, A. Drawing with constraints. *The Visual Computer*, 11(1):39–51, 1994.
12. Guiard, Y., Blanch, R., and Beaudouin-Lafon, M. Object pointing: A complement to bitmap pointing in GUIs. In *Proc GI 2004,* pp. 9-16.
13. Hudson, S. Adaptive semantic snapping—a technique for semantic feedback at the lexical level. *Proc CHI'90,* pp. 65-70.
14. Igarashi, T., and Hughes, J.F. A Suggestive Interface for 3D Drawing . In *Proc. UIST'01*, pp.173-181.
15. Jul, S. This is a lot easier! Constrained movement speeds navigation. In *CHI'03 extended abstracts,* pp. 776 - 777.
16. Lécuyer, A., Burkhardt, J.-M., Etienne, L. Feeling Bumps and Holes without a Haptic Interface: the Perception of Pseudo-Haptic Textures. In *Proc. CHI 2004.* pp 239–247.
17. Lécuyer, A., Coquillart, S., and Kheddar, A. Pseudo-Haptic Feedback: Can Isometric Input Devices Simulate Force Feedback? In *Proc. IEEE VR2000*, pp.18–22.
18. Lieberman, H. editor. *Your Wish is My Command–Programming by Example*. Morgan Kaufmann Publishers, 2001.
19. MacKenzie, I.S. Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction 1992.* 7:91–139.
20. Masui, T. HyperSnapping. In *Proc. Symposia on Human-Centric Comp.—Languages and Environ. 2001*, pp. 188–194.
21. Michael McGuffin, Ravin Balakrishnan. Acquisition of Expanding Targets. In *Proc. CHI'02*, pp. 57-64.
22. Nelson. G. Juno, a constraint-based graphics system. *Computer Graphics*, 19(3):235–243, *Proc. SIGGRAPH'85.*
23. Raisamo, R. and Räihä, K.-J. A new direct manipulation technique for aligning objects in drawing programs. In *Proc. UIST'96,* pp. 157–164.
24. Sutherland, I. *Sketchpad: A Man Machine Graphical Communication System.* PhD thesis, MIT, 1963.
25. Swaminathan, K. and Sato, S. (1997) Interaction design for large displays. In *Interactions* 4(1):15 – 24.
26. Thomas, B.H. and P. Calder. Applying cartoon animation techniques to graphical user interfaces. *TOCHI* 8(3):198–222, September 2001.
27. Worden, A., Walker, N., Bharat, K and Hudson, S. Making Computers Easier for Older Adults to Use: Area Cursors and Sticky Icons. In *Proc. CHI '97*, pp. 266–271.