WidgetLens: Interaction Through The Looking Glass

Bhavna Agarwal * Wolfgang Stuerzlinger †

Department of Computer Science & Engineering, York University, Toronto, Canada

ABSTRACT

Computer and mobile device screens are increasing in size and resolution. This increase in resolution causes problems with graphical user interfaces designed for lower resolution screens, as all information gets smaller and smaller. We present two novel techniques to make graphical user interfaces on high-resolutions screens more accessible and usable. We introduce a new in-place, localized zooming technique that works on a per widget basis. We also present a novel widget magnification technique that implements special modalities for common user interface elements, which affords widget-dependent magnification.

KEYWORDS: Graphical user interfaces, widgets, accessibility, human performance.

INDEX TERMS: H.5.2 [User Interfaces]: Graphical User Interfaces

1 INTRODUCTION

Computer monitors are getting bigger and bigger and resolutions, i.e. the number of pixels, continue to increase. This trend exists both on handheld devices, such as smartphones, as well as traditional laptops and desktops. In general, bigger displays are preferred as users can display more information and see more detail simultaneously. More relevant to our current work is that the pixel density of screens, i.e. the number of pixels per inch (ppi), is also increasing steadily. Currently smart phones and portable book readers use screens with 150-300 ppi, laptops 100-250 ppi, and desktops 100-200 ppi.

However, most graphical user interfaces (GUI's) are designed for monitors with a constant pixel density, typically about 100 ppi, the lower end of today's displays. This makes all user interface elements, including text, appear (too) small on displays with higher densities. This causes problems for users with poor eyesight, but affects even average people when they upgrade their system. In many common GUI toolkits there is little or no support for scaling of widgets and their elements, such as images or text, to resize or render them differently depending on the resolution. Partial fixes to this problem exist in the form of a global, system wide scale factor or full screen pixel-wise zoom. However, this reduces either the information content per window, or provides less screen space for other programs. This quickly becomes a problem if the user is working with more than one application at a time. The zoom/pan facilities associated with the "canvas" widget used in browsers, text editors and drawing programs etc., address this problem for that type of widget, but this solution does not generalize. Reworking existing GUI toolkits to provide scalable widgets is labour intensive and time consuming and will likely require significant adaptation in existing applications. Hence, methods are needed to enhance the readability of existing GUI's without reducing content.

In this article, we present a novel technique to make user interface widgets more scalable for different screen resolutions and thus more accessible. We explore target expansion for GUI's composed of densely tiled widgets and present new widget-dependent magnification techniques.

* e-mail: bhavna@cse.yorku.ca

† e-mail: wolfgang@cse.yorku.ca

2 PREVIOUS WORK

A GUI consists of many densely tiled targets, i.e. widgets. In such arrangements, the user needs to be able to quickly select and easily interact with any given widget. For this situation, dynamically expanding targets have been proposed based on their implications on human performance [1, 2, 3]. This established that dynamically changing a widget's size aids in selection tasks. Ruiz and Lank [2] presented a cost/benefit model for expanding targets in a tiled arrangement using kinematic endpoint prediction. They found a net benefit in expanding targets that occupy less than 4% of the display resolution.

Another area of relevant work is interface customization, i.e. adaptive and user-adaptable interfaces. Among this large body of work, Façades is particularly relevant [4]. This system permits seamless copy and paste of window regions using direct manipulation techniques, while maintaining full interactivity and without changes to the application code. Façades implements this by obtaining widget-related information through the accessibility API's supported by modern GUI toolkits and by redirecting events and screen output accordingly. Figure 1 shows the event management and image flow in Façades.

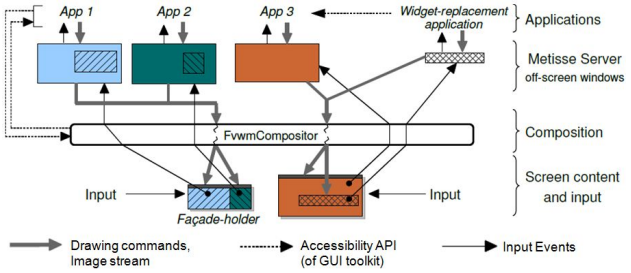

Figure 1: Façades architecture, from [4].

3 NEW TECHNIQUES

The basic idea behind WidgetLens is a localized widget zoom. Essentially, we map the widget under the cursor to a larger overlay window displayed over it. This design choice is motivated by the fact that we work in a (fixed) densely tiled arrangement of widgets, where there is likely no free surrounding space. Hence, we cannot move other widgets "away", similar to the OS X Dock.

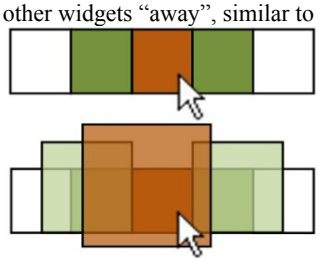

Figure 2: Illustration of widget displacement relative to cursor.

For this localized widget zoom, we present two new techniques using pixel magnification and widget replacement. Both use the accessibility API of modern toolkits to retrieve information about widgets, such as position and/or interaction possibilities.

3.1 Local Widget Zoom

In this version of a WidgetLens, the pixel image of the current widget is used as a texture for an enlarged version, which is overlaid on top. The widget in focus is thus enlarged in each direction (by default a factor of two). We move this enlarged version proportional to the distance of the corresponding edge of the original widget from the cursor, which provides a sliding effect. Mouse motions are mapped to guarantee that as soon as the mouse cursor leaves the original widget, the local widget zoom switches to the corresponding next widget. Any mouse or keyboard interaction with the fully zoomed version is passed through to the original widget.

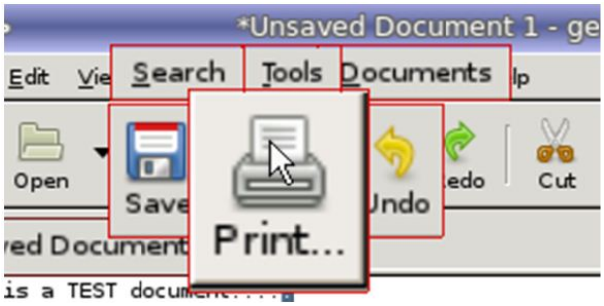

Figure 3: Widget in focus 2x zoomed and partially obscured widgets 1.5x zoomed. Red boundaries added for visualization.

As neighbouring widgets may be hidden by the expanded widget, we generate partially enlarged versions and overlay them at the respective corner closest to the cursor in an intermediate layer, which increases the visibility of these surrounding widgets. Figure 3 depicts the local widget zoom.

3.2 Smart WidgetLens

Straightforward magnification causes an undesirable blur in text and images. To address this, we present a novel form of widget lens that replaces widgets with higher resolution versions. This is implemented by traversing the widget tree of the underlying application and by duplicating each widget on a one-to-one basis in a separate window. The replication then uses larger fonts. If the system recognizes also the filename of an icon, it also uses a higher-resolution version of the corresponding image. Then, each of these enlarged versions is instantiated as a façade and overlaid on the original window as described in the local widget zoom (Figure 4). For simple widgets, such as buttons or menus, all interactions, such as mouse clicks are simply redirected to the original widget. The display mechanism, mouse events, and partially obscured widgets are handled as described above.

More complex widgets, such as combo-boxes or text fields, are managed differently. In this situation there are more interaction possibilities, such as selection, scrolling, or editing. We address this by allowing the user to interact directly with the overlaid widget and pass the resulting higher-level events, such as text changes or selection events, to the original widget. Another issue is that simple magnification may cause parts of the widget to fall outside of the monitor. Hence, we limit the number of lines, list items, and/or the number of characters to make sure that the widget stays within the boundaries of the monitor. We currently do not provide WidgetLens functionality for canvases. A naïve magnification would only lead to an explosion of white space. Moreover, we can assume that the application already provides zoom facilities for a canvas widget.

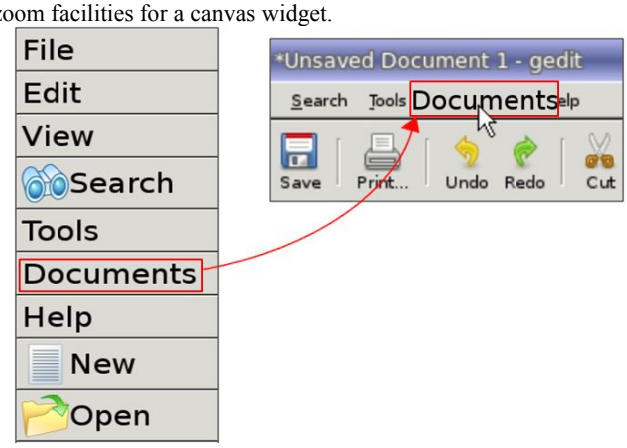

Figure 4 Left: Façade window with larger font (normally invisible), Right: User view.

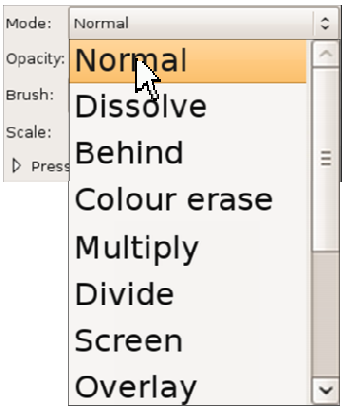

Figure 5: Combo-box with larger font and scrollbar.

4 Conclusion and Future Work

We presented a new method for end-users to interact with user interface elements on screens with high pixel densities. Based on the Façades system, we implemented two new methods that increase the visual accessibility and interactivity of GUI widgets.

We are working to implement various other effects, such as high-contrast display of widgets. Moreover, we are currently improving the animations to make the zoom effect less noticeable and dependent on cursor speed. Also, we are expanding the set of smart WidgetLenses further. Finally, we are considering a user study to analyze the benefits of WidgetLenses.

References

[1] A. Cockburn, P. Brock. Human on-line response to visual and motor target expansion. In *Proc. Graphics interface*, volume 137, pages 81-87, ACM GI, 2006.

[2] J. Ruiz, E. Lank. Speed pointing in tiled widgets: understanding the effects of target expansion and misprediction. In *Proc. Intelligent User Interfaces*, pages 229-238, ACM, 2010.

[3] M. McGuffin, R. Balakrishnan. Acquisition of expanding targets. In *Proc. SIGCHI*, pages 57-64. ACM, 2002.

[4] W. Stuerzlinger, O. Chapuis, D. Phillips, N. Roussel. User Interface Façades: Towards Fully Adaptable User Interfaces, pages309-318, ACM UIST, 2006.