Opportunities and challenges when 3D accelerating mobile user interfaces

Mikael Persson
e-mail: mikael@tat.se
TAT AB

Karl-Anders Johansson
e-mail: k-a@tat.se
TAT AB

Abstract

This paper addresses new techniques and challenges for user interface design for small-screen devices made possible by the recent availability of 3D graphics hardware. A survey of current research on the topic for desktop systems is presented and applications of these techniques for small screen devices are proposed. Also general differences in user interface design between large and small screen devices are highlighted.

Keywords: Graphics hardware, user interfaces, mobile devices

1. Introduction

Supporting graphics hardware in the user interface is becoming increasingly more popular for desktop systems, latest versions of Mac OS X and Microsoft Windows will employ 3D hardware for a range of new user interface techniques and effects. Currently we are seeing an explosion in graphics performance for both 2D and 3D on handheld devices made possible by the introduction of graphics hardware. This explosion is mainly driven by gaming but may be exploited for enabling a better user experience. Designing user interfaces for small screen devices is a tedious task mainly focused at organizing screen content such that large data sets can be presented in an intuitive way and to provide an interaction hierarchy that is easy to understand and use. The introduction of 3D graphics hardware makes a new range of techniques available and feasible on mobile devices that may help with providing more appealing, enjoyable and intuitive user interfaces. This paper aims to present previously described technical solutions for accelerating the user interface with graphics hardware on desktop systems and describe what challenges that must be overcome in order to realize these techniques on mobile devices with current and future graphics hardware. We also intend to highlight some interesting user interface design techniques found in the desktop space that may be applicable to mobile devices with graphics hardware.

In section 2 a survey of how 3D acceleration is currently being used in user interfaces as well as an overview on current research on the topic for desktop systems is presented. Section 3 contains a summary of technical limitations and challenges encountered when attempting to implement similar systems on mobile platforms. Finally in section 4 we outline aspects of user interface design for small screen devices related to the techniques described in section 2 as well as suggestions for new research.

2. Survey of 3D acceleration in desktop UIs

The dominant user interface model for desktop systems is window management. In [Myers 1988], Myers defines a window manager as "*a software package that helps the user monitor and control different contexts by separating them physically onto different parts of one or more display screens*". He adds: *"Before window managers, people had to remember their various activities and how to switch back and forth*". Generally speaking the window manager controls the physical display area and manages the different user interface components and applications that aspire on drawing graphics. Tasks that fall on the window manager are typically overdraw management, moving and resizing windows and controls as well as managing pointer devices such as a mouse. Some current systems [Graffagnino 2002; McCartney 2002] target this piece of software for 3D acceleration.
Traditionally user interface graphics systems draw directly to the display frame buffer via the window manager. The window manager controls which areas of the display are owned by a given application. If a certain area of the display need to be updated the window manager orders the application to draw its content directly to the display in the given area. Exploiting graphics

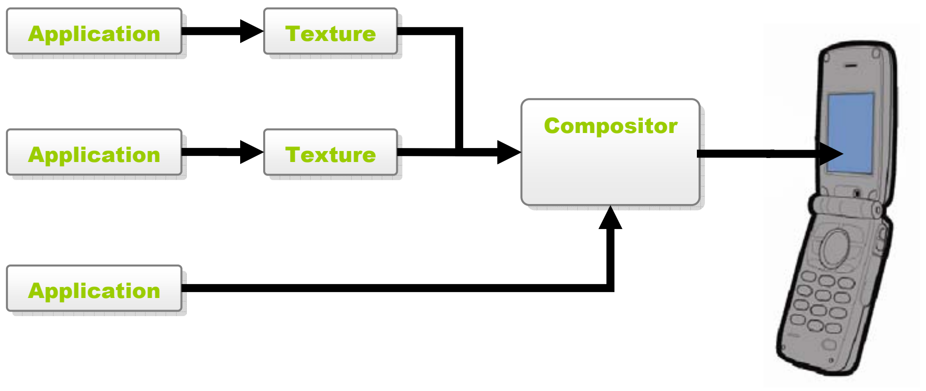

Figure 1. A common approach to accelerating a traditional window manager is the compositing approach, illustrated above.

acceleration without breaking compatibility with existing applications is generally achieved by introducing a compositor that manages the display instead of the individual application via the window manager, see figure 1. This is a common approach that is available in current generation Apple OS [Graffagnino 2002] and will be available in next generation of Microsoft OS [McCartney 2002]. In this approach the application draws its user interface to a texture accessible by the graphics hardware using existing methods. The different textures containing the application UIs are composited using the graphics hardware. Furthermore the compositor may use the graphics hardware to apply a wide range of effects during compositing, such as per-pixel transparency, pre-window fade and transform control, independent of the original application user interface code. In conjunction applications specifically written for the compositing pipeline may use the graphics hardware directly to further enhance the user interface. The compositor generally access the graphics hardware via an open API such as OpenGL [Graffagnino 2002; Kawahara and Byrne 2005] or a proprietary such as DirectX [McCartney 2002].

With a system as described above, a range of different new user interface concepts become available. A popular use case is organising windows on the desktop. In [Kawahara and Byrne 2005; Robertson et al. 2000; Rousel 2003] windows may be stacked at the side of the screen by rotating them in 3D, see figure 2. This gives the user overview of the content of the windows while freeing up space for more applications. Given a high performance multi-tasking system the content of the windows may be updated continuously such that the user can monitor any activity while enjoying the space for other applications. Other solutions to this problem include the *exposé* feature of Mac OS X Tiger. Instead of stacking the windows using perspective transforms the *exposé* feature tiles all of the open windows - scales them down and arranges them, so that all are completely visible. This allows the user to get an overview of all open windows. It also allows the user to select one of the miniature windows which will bring that window to front, this feature makes *exposé* useful both to get an overview of current running applications as well as application switching. Both features would not be possible without seriously draining system resources if not graphics hardware were employed.

Some researchers propose designs based on a compositor WM approach that are more decoupled from the traditional desktop metaphor UIs. G. Robertson et al. have presented a solution called *The Task Gallery* [Robertson et al. 2000] where user tasks (i.e. applications windows) appear as artwork hung on the walls of a virtual art gallery. The Task Gallery aim to exploit the spatial cognition and memory inherit in humans and is based on the theory that if presented with a virtual environment that is more like the 3D environment that we live in new and old users will find interaction more intuitive and enjoyable. According to the authors user studies have shown that the Task Gallery helps with task management in the way that users easier remember where they "put" their windows in the 3D metaphor. Although this concept is far away from the traditional desktop metaphor many designs that have found their way into current commercial systems such as stacking windows with perspective transforms are presented in this paper.

Interesting research projects are *Project Looking Glass* [Kawahara and Byrne 2005] and *Ametista* [Rousel 2003] two open source window mangers that are based on the 3D interaction metaphor. These projects aim to provide a platform for research on 3D user interfaces in a real working window environment. Perhaps the most interesting project is [Kawahara and Byrne 2005] where

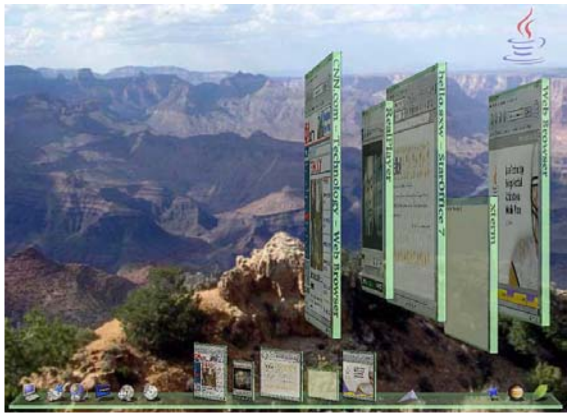

Figure 2. Project Looking Glass among others uses 3D perspective transformations to stack windows at the side of the screen to free up space for other applications. (Image courtesy of Project Looking Glass.)

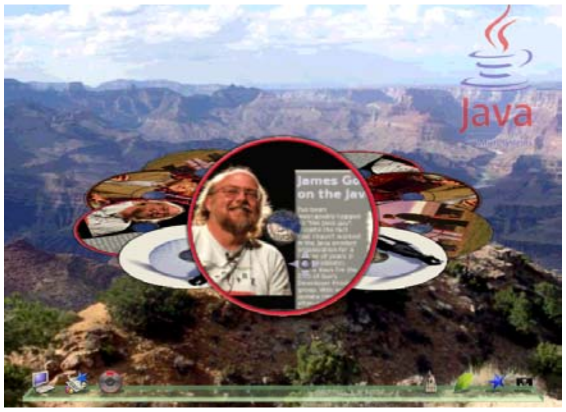

Figure 3. Project Looking Glass includes a media player with a free form 3D user interface. Using 3D many new and interesting UI techniques become available. (Image courtesy of Project Looking Glass.)

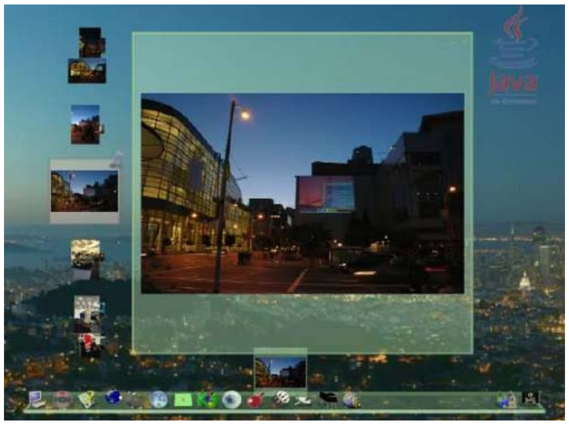

Figure 4. Project Looking Glass also includes a photo browser with a free form 3D user interface. (Image courtesy of Project Looking Glass.)

Java is used as the platform for building applications that expose 3D user interfaces. The window manager has a number of built in interaction paradigms such as stacking windows, task bars and how navigation is performed in the 3D space that the applications occupy. Besides from that novel interaction methods may be introduced by the individual projects. Some interesting applications have been developed as part of the project, most notably a media player and a photo browser. These applications have free form 3D user interface floating in the application space as oppose to traditional applications that are contained within their windows, see Figure 3 and Figure 4.

Another popular field of user interface research that doesn't require hardware acceleration but that definitely benefits from it is *zoomable user interfaces* or ZUI. Both OpenGL [Blythe and Munshi 2004] and the emerging standard OpenVG [Rice 2005] are based on hierarchical transformations which combined with more graphics performance via hardware acceleration makes a great platform for ZUI. The goals of ZUI is to better visualize large data sets on small display areas. The capability and performance to seamlessly change level of zoom enables a range of interesting user interface techniques. Photomesa [Graffagnino 2002] is an example of a browser that uses novel layout mechanisms (quantum treemaps and bubble maps) that allows users to see as many photos as possible and maintain context. It allows users to group photographs by date, filename and directory as well as employ zooming techniques to display the photographs. The basic concept is fairly simple, photos are clustered with regard to a number of parameters such as data, texture, color etc and ZUI is used to let the user seamlessly navigate the collection of images. The user directly control the level of zoom and the software attempts to always present the level of information that is suitable for the given level of zoom. It has been shown [Cockburn and Savage 2003] that techniques where the user directly controls the degree of zoom does not aid in searching for images, generally when a new degree of freedom for navigation is introduced the learning curve for the UI is steeper. An interesting extension of ZUI to remedy this problem is *speed dependent automatic zoom* (SDAZ). This concept is based on the fairly simple assumption that when the user is actively searching for an image a thumbnail overview is more suitable and when the user settles down on a given image a more zoomed in view is wanted. A photo browser based on the SDAZ concept is described in [Igarashi and Hinckley 2000; Cockburn and Savage 2003], and according to the authors user studies show that this technique aid some users in searching for photos.

3. Technical limitations and challenges

The introduction of graphics hardware for mobile devices presents great opportunities for building more intuitive and overall better user interfaces. However, when attempting to facilitate the research presented in section 2 for mobile devices a number of challenges arise, some of these challenges and limitations are covered in this section.

Lately, good standards for accessing the hardware such as OpenGL|ES and OpenVG have emerged. OpenVG is a much newer standard, and there are no hardware implementations yet. This section will focus on limitations of OpenGL|ES 1.x as this has been tried in the industry. Most OpenGL|ES graphics accelerators for the mobile market today offer quite good raw rendering performance and give major speed improvements on pure blitting and alpha blending. OpenGL|ES provides a convenient and standardized way of performing controlled composition of bitmaps and is capable of supporting at least half of the Porter-Duff blending operations [Porter and Duff 1984]. Details on blending limitation follow below. Current graphics hardware is capable of performing full screen bitmap composition at rather impressive frame rates as would be expected since the most important target software is games. Using the hardware graphics accelerator in a window system would mean enabling many of the features of a compositing based window system as described in section 2 without putting much load on the CPU. When using OpenGL|ES for this purpose, window content needs to be accessed as texture data. Most of today's OpenGL|ES implementations require textures to be uploaded to dedicated memory which presents a number of technical challenges that must be addressed:

Small texture memory
The typical OpenGL|ES accelerator on the market today targeting QVGA displays has about 1MB of VRAM (video ram) to be shared by frame buffer and textures. This clearly indicates that the window system must contain some sort of VRAM virtualization approach in order to fit all windows in "virtual VRAM". This is a problem that is common with desktop solutions [Graffagnino 2002; McCartney 2002]. By treating the VRAM as an on chip "L1" cache, it is possible to page in textures from a system RAM "L2" cache. Another issue is if the device has enough RAM to hold the "L2" cache. If not, the window system must resort to issuing a repaint command to the client in order to get the bitmap data. If this happens for every frame, we will not gain any performance boost from using the hardware graphics accelerator. More likely, the added overhead will in fact make the UI slower than if not using any hardware acceleration.

Slow texture transfer
In order to save power, all bus widths and -speeds are generally kept thin and slow on mobile devices. Also, graphics hardware manufacturers do not typically prioritize optimizations of the particular data path for transferring texture data since most games and benchmarks focus on fill rate and polygon count. This definitely has an impact on the VRAM virtualization mentioned above since it relies on being able to quickly swap in textures from RAM. The general solution to this problem is texture compression which lowers the impact on narrow busses but we will see below that this is not always feasible for user interfaces. The best workaround for the slow texture transfer is to reduce the number of client side updates and instead rely on effects and animations that are possible to perform on the graphics accelerator. Examples of this are scaling, moving, and opacity changes.

UI graphics is not suitable for texture compression
User interface graphics such as text and fine lines are very sensitive to compression artifacts. This effectively rules out texture compression for these tasks. Also, since applications and window content is rendered on the device in real time, texture compression times would pose a problem since it further delays the uploading of the texture data. Not being able to use texture compression further increases the texture memory problems (size and speed) mentioned above.

Texture size
OpenGL|ES 1.x supports neither texture sizes larger than 256x256 nor sizes other than powers of two. This means that texture RAM can not be utilized to 100% efficiency since windows rarely have sizes that are powers of two. It is possible to use tiling to work around this problem, in which case source bitmaps are broken down into small tiles that can be allocated from larger textures. The main problem with this is the vastly increased complexity of

the drawing operations while maintaining good performace. There is also a significant increase in the geometry complexity which may also cause performance degradations.

Blending limitations on OpenGL|ES 1.x
As mentioned above, only half of the 12 Porter-Duff blending modes are supported by OpenGL|ES 1.x. The reason for this is that in OpenGL|ES1.x there is no support for destination alpha. The ones that remain are “Clear”, “Src”, “SrcOver”, “DstIn”, “DstOut” and “Dst”. However, there is support for additional blending modes, for instance a version of “SrcOver” without the requirement of RGB being premultiplied with alpha

OpenGL|ES and other concurrent hardware
On a mobile device there is likely to be special hardware for decoding video and, if applicable, displaying a camera viewfinder. This can conflict with OpenGL|ES operation. Normally OpenGL|ES would not be running when using these special hardware features but with a hardware accelerated window system, OpenGL|ES is running all the time. Note that this differs very much between different OpenGL|ES implementations and integration decisions. For instance, an OpenGL|ES accelerator may include an interface for a camera and require that viewfinder- and snapshot data is stored in the chip’s embedded memory. This may end up consuming most of the accelerator’s RAM, not leaving any room for the window “L1” cache mentioned above, thus forcing the window system to revert to non-accelerated mode. Another example could be a graphics accelerator that actually does not at all support running the OpenGL|ES core when the camera viewfinder is active even if the RAM issue was solved.

4. UI design challenges and possibilities

In general when working with small-screen devices only one window/application may be simultaneously visible which has brought us back to the point where it is difficult to remember and organize activities. Generally for small-screen devices as opposed to desktop systems the concept window is analogous with screen, i.e. the window occupies the entire screen. User interaction is layed out in a hierarchical fashion, selecting an option on one screen generally presents another screen with more options, pressing the back key returns to the first screen. In these systems user interface designers often struggle with keeping the context such that the user is always aware of where a certain choice will take him/her and where the back key will lead at any given time, which as the complexity of the system grows becomes an enormous task.

Another major challenge and difference when designing user interfaces for mobile devices versus desktop systems is that on a mobile device you are most likely targeting first time users. On a desktop system you are designing a tool and optimizing it for maximum efficiency, on a mobile device on the other hand you are designing a system that has to be intuitive enough to be usable by first time users. Generally mobile devices do not come with a thick user manual and anyway you are expected to be able to use the device without reading any more instructions than what is presented on the display in the user interface.

Mobile 3D graphics presents many interesting opportunities for improving the problem with context shifts and its implications on usability as outlined above. Different kinds of transitions are very effective in helping the user to remember which context is currently active.
With the performance of graphics hardware and 3D a wide range of new transitions are available and a lot of research is required to

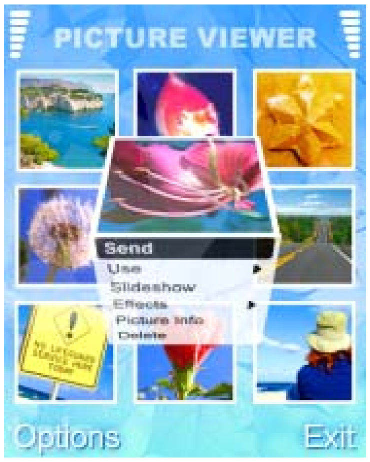

Figure 5. By connecting context menus associated with an object in a visually appealing way the human sense for spatial context may make these transitions more intuitive.

evaluate these transitions and how they may aid the user. One example of new transitions that may help the user is presented in figure 5, by connecting context menus associated with an object in a visually appealing way the human sense for spatial context may make these transitions more intuitive.

Both graphics and general processing performance is increasing at a rapid rate however display sizes are still limited by physical constraints such as the fact that the device must be able to fit in the pocket or in the palm of your hand. The introduction of high performance 3D graphics may be exploited to virtually extend the size of the screen by introducing new techniques for better organizing screen content. Section 2 discussed some concepts on desktop systems that attempt to solve this problem. This section will further discuss these topics in the context of mobile devices.

Stacking is a very common user interface technique used to display a set of components, icons, windows or pages in a compact manner. The components appear to be stacked or piled on top of each other with only a portion or tab visible. This tab is used to switch between the current active component and the component indicated by the tab, this is commonly referred to as tabbed windows introduced in [Beaudouin-Lafon 2000] and extended in [Beaudouin-Lafon 2001]. Stacking is especially attractive for small screen devices since the technique allows for quick navigation among pages on a limited space, where otherwise each page had to be its own window. However, a common problem of packing information tightly is that it will most likely be less intuitive. In such cases, visual perception will become critical for the user to understand what is displayed.
A user study [Kjelldahl 2003] has shown that the visual cues perspective and shadow have a substantial positive effect on perceiving position. Figure 6 shows two examples of stacking images (for instance in a photo viewer application): One without the perspective and shadow and one where the two visual cues have been used to help the user perceive the scene.
With the use of modern graphics hardware and the compositor approach, effects such as perspective transforms (texture mapping) and shadow techniques may be used on arbitrary user interface components to achieve the right visual cues and thus reduce the risk of the user misinterpreting the intended use.
As described in section 2 recent products and research on desktop systems that employ graphics hardware in the user interface

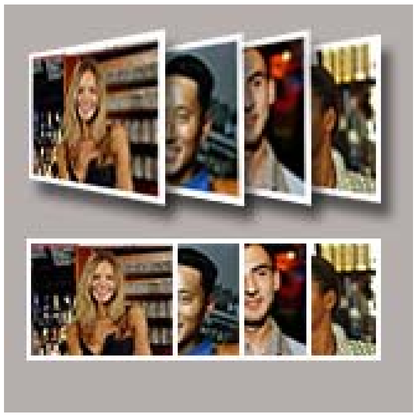
Figure 6. Normally tabbing is achieved by simply displacing the images (bottom). Using visual cues such as perspective and shadows the UI may be made more intuitive (top).

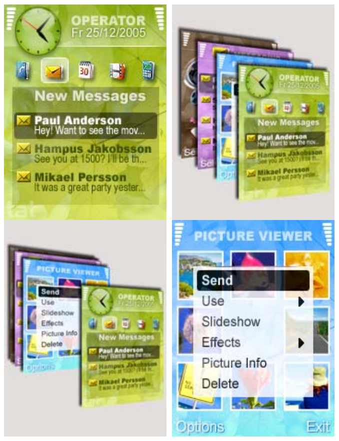

Figure 7. Another example of where stacking in perspective can be used. Here used for application switching.

attempt to achieve better organization of screen content by stacking windows at the side of the screen with perspective transformations thus exploiting these visual cues. Although stacking windows by the side of the display would be feasible on a mobile device, the small screen size would most likely render the window content impossible to see. The same unfortunately apply for the *exposé* feature. Unless only two or three applications are running, scaling down all open windows such that they all fit the display is not feasible for most mobile devices due to the small screen. Instead stacking combined with the visual cues perspective and shadow may be used to give an intuitive overview on the limited display space, see figure 7. This effect may be useful for more intuitive application switching or simply to get an overview of currently open windows.

Improving screen content organization and virtually extending the display on small screen devices is one of the biggest challenges for user interfaces designers and much research is needed. The recent introductions of OpenGL|ES and OpenVG plus hardware acceleration will, as mentioned in section 2, provide an excellent platform for *zoomable user interfaces*. Zoomable user interfaces is a popular research field for both mobile and desktop systems with the goal of better visualizing large data sets on small displays. As storage capabilities of mobile devices and available information via network connections increase rapidly, the need for new and better ways to visualize this data on small screen devices are imperative. ZUI have great potential for improving screen content organization in a natural way. The PhotoMesa [Bederson 2001], also mentioned in section 2, has also been developed in a version for PocketPC [Khella and Bederson 2003]. As mentioned in section 2, problems were encountered when introducing new navigation methods, i.e. direct controls for zooming. The lack of sophisticated input devices such as a mouse/point on many mobile devices is likely to make it even less attractive to introduce new navigational degrees of freedom. This makes methods such as SDAZ very interesting. Interesting research would be to extend the concept of SDAZ to work well with navigation keys and other systems that do not have pointer devices.

5. Conclusions

High performance 3D graphics hardware will enter the mobile arena driven by the gaming industry and thus it is very likely that the mobile user interface will evolve in the same direction as the desktop systems. Some of the new user interface techniques developed for these systems may provide great solutions for problems inherited in mobile devices such as better screen content organization on small screens via *zoomable user interfaces* or more intuitive interaction flows via transitions. More research is needed to adapt, evolve and evaluate these techniques for small screen limited mobile devices.

References

BEAUDOUIN-LAFON, M. 2001. Novel interaction techniques for overlapping windows. In Proceedings of ACM Symposium on User Interface Software and Technology, UIST 2001, Orlando (USA), ACM Press. Pages 153-154.

BEAUDOUIN-LAFON, M. and LASSEN, H.M. 2000. The architecture and implementation of CPN2000, a post-WIMP graphical application, in UIST 2000, ACM Symposium on User Interface Software and Technology, CHI Letters 2(2). Pages 181- 190

BEDERSON, B. 2001. Photomesa: A zoombable image browser using quantum treemaps and bubblemaps, in Proceedings UIST 2001, ACM Press. Pages 71-80

BLYTHE, D. and MUNSHI, A. 2004. OpenGL|ES Common/Common-Lite Profile Specification Version 1.1.04 (Annotated), www.khronos.org

COCKBURN, A. and SAVAGE, J. 2003. Comparing Speed-Dependent Automatic Zooming with Tradional Scroll, Pan, and Zoom Methods, People and Computers XVII: British Computer Society Conference on Human Computer Interaction. Bath, England. Pages 87-102.

GRAFFAGNINO, P. 2002. Apple OpenGL and Quartz Extreme. Presentation at SIGGRAPH 2002, OpenGL BOF.

IGARASHI, T. and HINCKLEY, K. 2000. Speed-dependent automatic zooming for browsing large documents, Proc. UIST 2000, ACM Press. Pages 139-148

KAWAHARA, H. and BYRNE P 2005. Project Looking Glass. Presentation at Sun JavaOne 2005. https://lg3d.dev.java.net/

KHELLA, A. and BEDERSON, B. 2003. A Zooming Image Browser for the Pocket PC. University of Maryland, Technical Report

KJELLDAHL, L. 2003. A survey of some perceptual features for computer graphics and visualization. SIGRAD2003

MCCARTNEY, C. 2002. Windows Desktop Composition. Presentation at WinHEC 2002, Windows Graphics Architecture session.

MYERS, B.A. 1988. A taxonomy of window manager user interfaces. IEEE Computer Graphics and Applications, 8(5). Pages 65-84

PORTER, T. and DUFF, T. 1984. Compositing digital images. Computer Graphics (SIGGRAPH '84 Proceedings), vol. 18, no. 3, pages 253-259

RICE, D. 2005. OpenVG 1.0 Specification. www.khronos.org

ROBERTSON, G., VAN DANTZICH, M., ROBBINS, D., CZERWINSKI, M., HINCKLEY, K., RISDEN, K., THIEL, D. and GOROKHOVSKY, V. 2000. The Task Gallery: a 3D window manager. In Proceedings of ACM CHI 2000 Conference on Human Factors in Computing Systems, ACM Press. Pages 494-501.

ROUSSEL, N. 2003. Ametista: a mini-toolkit for exploring new window management techniques. In Proceedings of CLIHC 2003, Latin American Conference on Human-Computer Interaction, ACM Press. Pages 117-124.