DeskAlign: Automaticlly Aligning a Tiled Windows Desktop

Grant Wallace, Han Chen and Kai Li
Department of Computer Science, Princeton University, Princeton, NJ 08544.
{gwallace,chenhan,li}@cs.princeton.edu

Abstract

Tiled projector arrays are effective at meeting the needs for scalable, cost effective, higher resolution displays. Increases in PC performance have allowed small tiled displays to be driven from a single PC with multiple graphics cards. In this paper we present a system for automatically aligning the Windows Desktop of a tiled display. This system consists of three primary procedures: detecting projector misalignment, calculating corrective transformations, and real-time warping of the desktop. This allows users to run any Windows 2D, 3D or video application without modifications or special software support. Our experiments indicate that the system is able to achieve real-time warping with minimum system performance degradation.

1. Introduction

Large format collaborative displays are increasingly useful in a variety of application environments including control rooms, CAD design, education, and business. For collaborative displays to be effective, it is important that they are easy to use and provide enough resolution and size to readily view the group's work.

Figure 1: A DeskAlign installation in PPPL Control Room

One common way of building a high-resolution display is to tile together a set of projectors. Tiled displays have typically been driven by specialized graphics pipelines such as the SGI Onyx, or by a cluster of PCs each rendering a single tile. These types of rendering systems have performance advantages, but can be cumbersome for creating effective collaborative environments. Graphics pipeline computers are too expensive and specialized for general computing needs, while a cluster of computers requires special tools to use and manage. Ideally a collaborative display will be driven by a system that people find easy to use and are familiar with. The increasing capabilities of commodity PCs can make this possible. A single PC with multiple graphics cards can potentially drive upwards of 16 displays with 4 to 8 displays being a more optimal setup. Luckily, Collaboration environments often require only small tiled arrays. This makes building a tiled collaborative display from a single PC an attractive choice because it can present an intuitive user interface such as the Windows Desktop in an economical fashion.

An important aspect when tiling together projectors is to get precise geometric alignment. Even small amounts of misalignment lead to gaps and double images which make the display unacceptable. Manual alignment is possible but tends to be time consuming and less accurate than automated approaches. Automated approaches are fast and accurate, reducing the maintenance overhead, but must be applied in real-time to the displayed imageries. Applying the alignment requires that a projective warp be applied to each projector's output. Warping the imagery is typically done by modifying the rendering application. Unfortunately this is not easily done with a Desktop environment where the source code may not be available.

The remainder of this paper presents a system we developed called DeskAlign which automatically aligns and warps the windows desktop of a tiled display driven by a single PC. Some previous work will be discussed in Section 2. Section 3 will talk about design choices in creating such a system. Section 4 will describe our system's implementation and section 5 will present some evaluations and experiences with using DeskAlign. Section 6 will conclude.

2. Background

The goal of aligning a tiled desktop can draw on the experiences of several categories of research, but is not completely addressed by any of the current tiling systems. In this section we will look at some previous work and compare some existing systems.

2.1. Automatic Alignment

Automatic alignment of tiled displays is well documented in the literature. There have been a variety of techniques that have emerged for different display environments. Single camera view alignment of a planar screen is covered by PixelFlex [11]. Scalable alignment of a planar screen using multiple camera views is done by H. Chen *et al* [1], Y. Chen *et al* [3]. Aligning displays on an arbitrary surface can be found in Raskar et al [8]. These techniques all involve detecting projector feature points with a camera. The relative position of the feature points with one another is then used to extract a set of transformations that, when applied to the displayed imagery, make the projectors appear aligned from the cameras viewpoint. The techniques typically yield sub-pixel accurate alignment in a matter of minutes.

2.2. Projector-Camera User Interfaces

Several systems have effectively used projector-camera interactions to create enhanced user interfaces. Smart Projectors [10] uses a camera to correct for projector keystoning, create a laser-pointer user interface and do shadow elimination. Also the Everywhere Display [7] uses dual headed graphics cards to do two pass rendering resulting in one display output that is automatically keystone corrected on any surface it projects to. This also has the facility for tracked user interaction.

2.3. Display Tiling Systems

There are several systems that have been developed to help bring desktop environments to tiled displays. Most of them involve running a client-server architecture. Usually one computer will act as a proxy. This proxy looks like a single display to the applications, but it then divides the display content and redistributes it to the tile nodes. A windows implementation of such a system is the Virtual Display Driver (VDD) [4]. VDD creates a virtual Windows Desktop of arbitrary resolution. When applications running on that computer make GDI drawing calls, the calls are intercepted, scaled and sent to the appropriate nodes of the tiled display as shown in Figure 2. Distributed Multiheaded X (DMX) [6] is a similar proxy for X Windows environments. A Xserver runs on one PC and accepts display commands. It then redistributes these X rendering commands to the cluster nodes (see Figure 3). Both VDD and DMX operate with 2D drawing primitives in order to reduce the network bandwidth which would be required when sending pixel information. Another solution is an adaptation of Virtual Network Computing (VNC) [9]. VNC allows a user to connect to a remote computer and view/interact with the desktop. It requires the remote computer to run VNC Server which compresses and ships pixel information to the client computer. This application was modified in a release called VNCWall. The VNC Server was modified allowing it to handle requests for multiple rectangular subsections of the display. This allows each node in the display to connect to the server and ask for a different subsection thus creating a tiled desktop.

One common aspect of the above systems is they are designed to run on a display cluster. The goal of our system is to run from a single PC. The proxy based systems can be used in loopback mode where the client and server both reside on the same PC. This is one potential solution to cre-

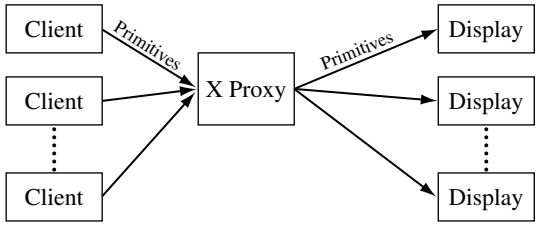

Figure 3: Distributed Multiheaded X

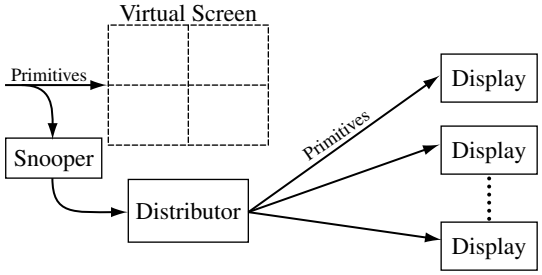

Figure 2: Virtual Display Driver for Tiled Display

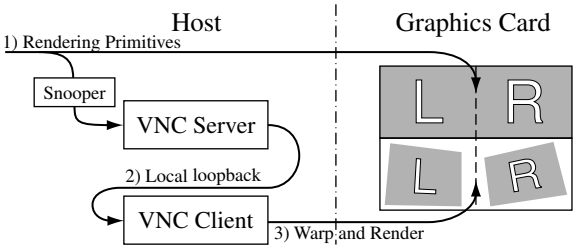

Figure 4: VNC loopback mode on a single PC

ating a single PC tiled desktop. However, in tests we've done on these types of systems, several limitations exist. One limitation is the performance overhead incurred by the extra loopback copying. For a VNC system in loopback mode (Figure 4), we found there was about a one second delay for window refresh and dragging. Also mouse cursor movement is jerky and delayed. Another problem is that these systems typically snoop on the 2D rendering calls made in order to track updates. Applications that render directly to the graphics card such as 2D and 3D apps using DirectX will not be properly rendered. One method to get around some of these limitations is to do post-rendering alignment transformations. We will look at some of these options further in section 3.

3. Design Choices

When creating an automatically aligned tiled desktop, there are several design choices that need to be made including: the projector alignment algorithm; method of applying the transformations; and how to distribute the desktop content to the tiles.

As mentioned in section 2 there are several existing techniques for determining projector misalignment. Choosing one appropriate for the screen configuration should yield good results. The method of detecting misalignment is independent of the other design choices and can easily be switched.

The second two processes, applying the transformations and tiling the Desktop, are somewhat coupled. If the system responsible for tiling the desktop has information about the positions of the projectors, then it can adjust the amount of content it ships for each projector. For instance, if a projector covering a small area of screen is surrounded by projectors covering larger areas, then the tiler can send a smaller section of the desktop to the one, while sending more to the others. This allows the physical positioning of the projectors to be very coarse and still produce a good final result. If the tiling system has no knowledge of the projector positions (other than which grid area it occupies) then it sends the same resolution to each display and the post rendering transformations must make the size of the projectors' output match that of the smallest. This has the disadvantage of wasting projector resolution or requiring a more precise physical placement of the projectors.

In order for the tiling system to be aware of the projector alignment, either the desktop system must be able to handle this information or a proxy system must be used. Current desktops do not have the capability to handle detailed projector position information. This type of integration would require the use of a proxy such as VDD, VNCWall, DMX, or possibly future versions of Windows

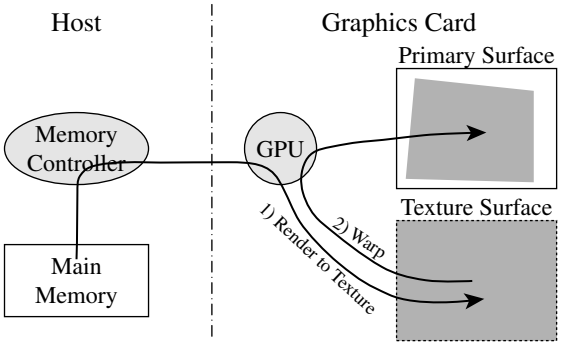

Figure 5: Rendering architecture of NVKeystone

Terminal Services [1]. Proxies can add considerable overhead as data must be shipped to the proxy and then redistributed to the display nodes. Even if everything is on a single PC it still requires copying the data around as opposed to just sending drawing calls to the graphics card.

If the tiling system is unaware of projector alignment, as is the case with normal Windows Desktop multi-monitor support, then we must apply alignment transformations after the content has been rendered on the graphics card. These transformations can be done in one of three places: the graphics card, the projector, or between the graphics card and projector. There are some projectors on the market which can apply arbitrary projective transformations, but currently they are quite expensive. There are also companies that make video switches that can perform transformation of the video coming from the graphics card. Again these tend to be expensive. So currently the most cost effective place to perform the transformations is on the video card.

It would be best if the graphics card natively supplied an API to allow for post-rendering transformations. We initially thought the NVIDIA NVKeystone extension would provide exactly this API. NVKeystone allows one display to be projectively warped to correct for any off-axis projection (Figure 5). However, after some experimentation, we found two major flaws with the NVKeystone extension. First, it only allows the warping of one monitor per computer. Since our goal is to run a tiled display from a single computer this effectively eliminates the utility of NVKeystone. The second limitation is the lack of a programmable interface to NVKeystone. It only presents a GUI which allows you to drag the corners of the display. Ideally we would like to pass in some transformation information calculated from our alignment algorithm.

In the absence of graphic card support for post rendering transformations, we decided to do a two pass rendering approach. We want to keep all of the rendering on the graphics

[1]Current versions of Windows Terminal Server only allow one client connection at a time and have no subregion support.

card so we don't hinder performance. In order to do this it is necessary to double buffer and have control of the buffer swapping. Since the Windows Desktop is doing the rendering, we don't have this type of control. As an alternative, we decided to use another graphics pipeline on the same card for the second pass rendering, such as the approach in [7]. Dual or quad headed graphics cards have 2 or 4 graphics pipelines, one per head. To perform two pass rendering we pair pipelines together and don't use the display from the first pass pipeline. Thus we turn two graphics pipelines into one. This reduces the number of potential displays, but allows for the on-board performance that our system needs.

4. DeskAlign System

We have implemented a system called DeskAlign which uses a camera to collect geometry information from the projectors, calculates the appropriate transformations for each projector and then aligns the Desktop running on a single PC. The system is comprised of three software components. 1) A Windows application which collects configuration information from the user and takes pictures of feature points on the display. 2) A Matlab program which processes the feature point images and determines the transformations which should be applied to each projector. 3) A DirectX application that performs the second pass rendering on the graphics card as well as optionally drawing feature points on the projectors or presenting a GUI with draggable projector corners for manual alignment adjustment. Components 1 and 2 are based on the Camera Homography Tree (CHT) alignment algorithm [1]. Component 3 is a modification of the Princeton Image Viewer [2].

4.1. Detecting Projector Positions

The first part of our system is 'DeskDetect'. This is an application that gathers the projector feature point information. It has three main functional components: configuration settings, camera control and DeskAlign communication.

The user typically begins by entering some configuration settings. These settings can be saved in a config file and later reloaded for convenience. The type of configuration information needed is

1. DeskAlign hostname,
2. Projector rows, columns and resolution,
3. Camera coverage and resolution,
4. Feature point patterns,
5. Image file path and prefix.

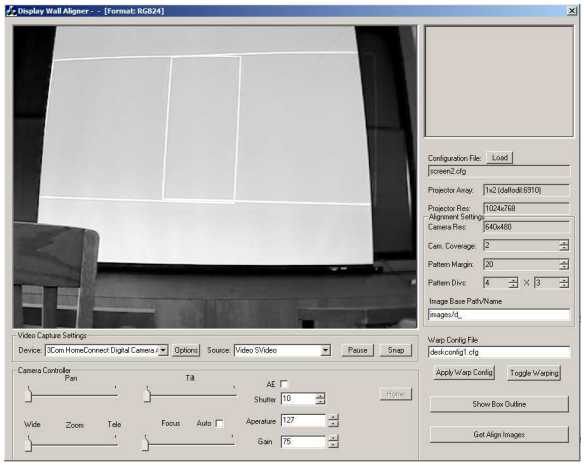

Figure 6: DeskDetect GUI for collecting projector configuration information.

The user then selects the video source to use. DeskDetect can use any video source that supplies a WDM video driver. Camera control, including shutter speed, aperture, gain, focus, zoom, pan and tilt, is available for certain cameras. Once the camera is configured correctly the system is ready to acquire feature point images.

To begin acquiring feature points we need to communicate with the DeskAlign display system. DeskAlign and DeskDetect can run on the same or different PCs. DeskAlign controls the output to the projectors and will be responsible for drawing the feature points needed for image collection. DeskDetect communicates with DeskAlign to toggle the warping on or off, and send drawing commands. The first step is to toggle off DeskAlign warping. This stops the display of desktop content and puts DeskAlign in drawing mode. Now we can send 2D drawing commands to the displays. Once in drawing mode, we typically want to first check the rough alignment of the projectors. DeskDetect sends drawing commands to highlight the edges of the projectors. It is important that the projectors have some overlap so that the result can be seamless. Next we can acquire feature point images. DeskDetect sends drawing commands to DeskAlign and then takes pictures of the screen. We use the CHT method of feature point detection which uses horizontal and vertical lines as its primary features. Once all the feature point images have been acquired, we are ready to process the images and determine appropriate projective transformations to apply to each projector.

4.2 Determining Projector Homograhpies

The projector transformations are determined by a set of Matlab routines based on the CHT algorithm. These routines do image processing to extract the feature line infor-

mation. Feature points are then found from the intersection of horizontal and vertical lines. The feature points allow us to find a homography from camera space to projector space. Once the projector corners positions are known in camera space we can begin determining an optimal transformation to apply to each projector. There are two criteria we must keep in mind. First, we want the entire desktop contents shown, it is not advisable to clip part of the desktop if it doesn't quite fit. Second, since we are performing these warping transformations after the rendering stage, it is not possible (or at least very slow) to ship pixels back and forth between displays. So we want each display to show all of its content and only its content. This means we want to display in the largest bounding rectangle contained within the intersection of the displays (uneven edges will show black). And we may have to reduce the final display size even more to match the smallest zoomed projector. Also we will need the overlaps of successive rows and columns to line up sufficiently in order for a continuous seam to be drawn. If the overlaps aren't sufficiently large or aligned we may not be able to show all portions of the desktop. The final results of the optimization are the pixel positions within each display at which the content is to be shown. This information is then sent to DeskAlign so that it can apply these transformations.

4.3. Applying Alignment Transformations

DeskAlign is a Windows application that performs the second pass rendering transformations according to the alignment information created in the previous step. DeskAlign performs the following tasks. First it determines the number of monitors and their resolutions. Based on the monitor information it creates two sets of monitors, A and B. Set A will be the monitors containing the unmodified Windows Desktop Content. Nothing needs to be done to A monitors, Windows already handles this. Set B monitors will show the warped Desktop content. To accomplish this, DeskAlign opens a full-screen mode window on each monitor in Set B. Then a periodic timer is set (currently every 1/30 sec) and at each timer event DeskAlign copies the pixels from the frame buffers of Set A monitors to texture memory. Then the content is warped from texture memory to the Set B frame buffers using texture mapping hardware. The end result is the contents are copied from Set A monitors and warped to Set B. Only Set B outputs are attached to projectors. So the number of display outputs is reduced in half in order to perform the alignment warping.

DeskAlign is currently implemented as a DirectX application. DirectX will only perform an on-card pixel copy to memory locations within the same address space. If the copy appears to cross address space boundaries, DirectX first copies the pixels to main memory and then back again. For performance reasons it is important that paired monitors have single address space memory. This is true in NVIDIA's

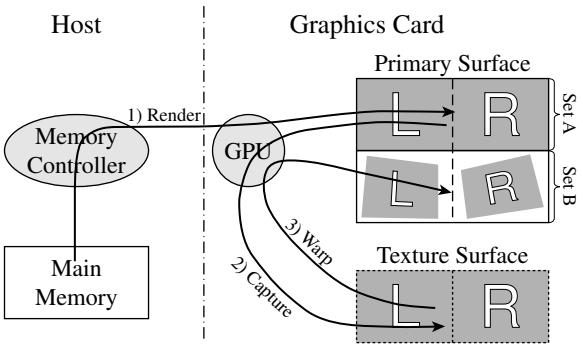

Figure 7: DeskAlign Diagram

"Span" mode. In span mode you can span two monitors together so they look like one monitor to Windows. This insures a continuous address space shared between the two monitors.

5. Evaluation and Experiences

5.1. System Evaluation

We evaluate several aspects of our system including, the set up time, the alignment accuracy, and the usability.

We set up our system using a PC with a 866 MHz Pentium III processor and 256 MB memory, a PNY NVIDIA Quadro4 400 NVS graphics card and two Compaq MP1800 projectors. The system requires a relatively short set up time of about 10–15 minutes. This has the advantage of making it portable. Once all hardware and software have been installed in the PC, the setup basically involves hooking up the projectors and a camera, capturing some alignment images, running the alignment algorithm to generate the projector transformations and then sending this new configuration to DeskAlign. The alignment steps take under five minutes, and most of the setup time is in hooking up the projectors, booting the computer and configuring the NVIDIA driver. A final manual adjustment in software can also be made to get the resulting display as large and square to the screen as possible. This is done by dragging the corners of the display to the appropriate positions similar to a technique used in [10]. The resulting display is aligned to within one pixel when using an inexpensive webcam.

One way to consider the usability of this system is to consider if a user can perform the same tasks as with a normal desktop and with the same performance. In other words, does tiling add resolution without removing other capabilities. With regards to the first question, our system is able to run and view all application output except for pixel data rendered with a hardware overlay. Some video applications use a hardware overlay, but generally this can be disabled. Also the mouse is typically rendered with an overlay but this can be disabled in the "Display Properties" settings

of Windows. The DeskAlign system uses about 10% of the CPU, most of which is due to hardware contention. We will look further into some of these issues in the next section.

5.2. System Experiences

In this section we will detail some of the experiences and lessons learned while setting up and using the DeskAlign system.

The NVIDIA driver was somewhat tricky to configure correctly. We require each pair of displays to be in "Span" mode so that DirectX will see them as sharing memory. This can be accomplished after turning off "treat all displays as separate devices". This must be done for each display pair and so two reboots are required for a quad card. After rebooting, "Span" mode is only enabled if monitors are detected on each of the VGA outputs. This requires an AB monitor switch to send the appropriate signal when no monitor is present.

We initially found that DeskAlign was using a lot of CPU time, sometimes up to 90% when running PowerPoint. After some investigation it turned out that most of this performance hit was contention for writing with the blitter. In particular we found that when "Clippit", the Office Assistant, was running on the warped output screens there was the greatest contention. Moving "Clippit" off the output screens and onto the desktop screens reduced the contention in half. We further reduced the contention by running blt commands with the DONOTWAIT flag, and by putting a 10 ms delay between blitter calls. This reduced our contention down to below 10% on average. "Clippit" can still increase this amount to 20% if moved onto our render screens. Contention also increases linearly with the refresh rate with about 10% contention at 30Hz refresh. If we comment out the blitter commands in our code, DeskAlign CPU usage is essentially 0%.

We also found that the mouse was not initially visible in our warped output. This turned out to be due to hardware acceleration which uses an overlay for the mouse pointer. We lowered the hardware acceleration by one notch in the display-properties/advanced/troubleshooting dialog box and this returned the mouse to normal rendering mode without any appreciable decrease in performance. Similarly, for video applications, overlay mode must be turned off in order to render to the frame buffer. This is typically an application setting, or automatically adjusted when the window spans multiple screens.

The system is easily used while only viewing the projection screen, there is no need for a monitor. Two refinements are needed to the typical usage pattern, both have to do with application resizing and positioning. The first usage modification is that applications must be resized to cover the full screen rather than automatically maximizing with the "full screen" button. The full screen button on a multi-monitor system typically only covers one monitor. The second usage modification is keeping application windows within the boundaries of the display. Since the bottom of the desktop becomes the top of the warped output, it is necessary to resize objects so that they don't extend down further than the resolution of the visible desktop.

6. Conclusion

Collaborative environments can often benefit from large shared displays. Tiled arrays of projectors are one of the most cost effective means of achieving this. The DeskAlign system presented in this paper provides a method for automatically aligning the Windows Desktop on a small tiled display driven by a single PC. Running from a single PC has the advantage of eliminating clustering tools and environments and allowing users to interact directly with a desktop. The system incorporates the detection of projector misalignment, calculation of a corrective transformations, and real-time desktop warping. The result from using DeskAlign is a high resolution desktop system that is quick to set up and easy to use.

Acknowledgments

The Princeton Scalable Display Wall project is supported in part by Department of Energy grant DE-FC02-01ER25456, by NSF Infastructure Grant EIA-0101247, by NCSA Grant ACI-9619019 (through NSF), by Intel Research Council, and by Intel Technology 2000 equipment grant. Han Chen is supported in part by a Gordon Wu Fellowship.

References

[1] H. Chen, R. Sukthankar, G. Wallace, and K. Li. Scalable alignment of large-format multi-projector displays using camera homography trees. In *Proceedings of IEEE Visualization*, 2002.

[2] Y. Chen, H. Chen, D Clark, Z. Liu, G. Wallace, and K. Li. Software Environments for Cluster-based Display Systems (2001)

[3] Y. Chen, D. Clark, A. Finkelstein, T. Housel, and K. Li. Automatic alignment of high-resolution multi-projector display using an uncalibrated camera. In *Proceedings of IEEE Visualization*, 2000.

[4] K. Li, et al. Early Experiences and Challenges in Building and Using A Scalable Display Wall System. *IEEE Computer Graphics and Applications*, vol 20(4), pp 671-680, 2000.

[5] T. Funkhouser and K. Li. Large format displays. *IEEE Computer Graphics and Applications*, 20(4), 2000. Guest editor introduction to special issue.

[6] K. Martin, D. Dawes, and R. Faith. Distributed Multihead X design. http://dmx.sourceforge.net/dmx.html

[7] G. Pingali, C. Pinhanez, A. Levas, R. Kjeldsen, M. Podlaseck, H. Chen, and N. Sukaviriya Steerable Interfaces for Pervasive Computing Spaces. In *IEEE International Conference on Pervasive Computing and Communications - PerCom'03*. 2003

[8] R. Raskar, M. Brown, R. Yang, W. Chen, G. Welch, H. Towles, B. Seales, and H. Fuchs. Multi-projector displays using camera-based registration. In *Proceedings of IEEE Visualization*, 1999.

[9] T. Richardson, Q. Stafford-Fraser, K. Wood and A. Hopper. Virtual Network Computing. In *IEEE Internet Computing*, Vol.2 No.1, Jan/Feb 1998 pp33-38.

[10] R. Sukthankar, R. Stockton, M. Mullin. Smarter Presentations: Exploiting Homography in Camera-Projector Systems. In *Proceedings of International Conference on Computer Vision*, 2001.

[11] R. Yang, D. Gotz, J. Hensley, H. Towles, and M. Brown. Pixelflex: A reconfigurable multi-projector display system. In *Proceedings of IEEE Visualization*, 2001.