A large-scale touch sensitive display

Michael Jenkin, John Tsotsos, Alexander Andreopoulos, Andrei Rotenstein, Matt Robinson and Jeff Laurence
Computer Science and Engineering and Centre for Vision Research
Faculty of Science and Engineering, York University
Toronto, Ontario, Canada, M3J 1P3

Abstract — Large scale displays present a number of challenges in terms of physical construction and software control. This paper describes a software and hardware infrastructure that supports extremely large interactive display surfaces. The resulting device is capable of supporting multiple users interacting with the display surface simultaneously and the display of complex interactive graphics over expansive high resolution displays. A specific implementation 36”×54” with a pixel resolution of 5040 × 3150 pixels is presented.

I. INTRODUCTION

Regardless of the available size of a particular display surface there almost always seems to be a desire to increase its size. But the construction of extremely large interactive display surfaces presents a range of complex technical problems that must be overcome. This includes fundamental limits in terms of a device’s ability to actually repaint the physical display, the need for a highly parallelizable rendering infrastructure, and issues related to the need to have multiple concurrent points of interaction on the surface.

Suppose one wanted to construct a very large display. Generating large low-resolution displays is relatively straightforward and there are many examples of technologies that support this type of display. What is more difficult is the development of very large displays with constant (or near constant) pixel densities. The scale of such a device quickly approaches the limits of existing video hardware to actually repaint a surface (current video display systems rarely support displays of larger than 2000x1000 pixels). But even more fundamental limits exist beyond this. If the display is to present moving imagery with a minimal refresh rate of above 30hz, then actually repainting the physical display by reading a single serial display of (say) 5000x3000 pixels with 24 bits/pixel would require a video refresh pipeline capable of transmitting $1.3 \times 10^9$ bytes/second. The display of large scale display surfaces requires some sort of mechanism to parallelize the physical repainting of the display.

Actually generating content to paint the display raises another issue. Modern graphics displays obtain much of their performance due to the graphic hardware’s ability to offload ‘low level’ operations from the main CPU. These graphics processors are typically tied to specific video resolutions which are much lower than the types of resolutions desired for large scale displays. Nor does existing graphics display hardware have the ability to provide the needed parallelism in terms of the need to actually paint pixels.

Parallelizing actual pixel painting and graphical rendering would appear to be essential components of any hardware and software infrastructure to support large scale graphical displays. Yet this introduces other issues related to synchronization of the rendering and display process, especially if these tasks are spread over loosely coupled hardware.

Finally, there is the issue of interaction. Small display screens are typically designed for use by a single user. As the size of a display surface grows there is the potential for multiple users to wish to interact with the same display surface concurrently. Providing a mechanism to enable multiple users to interact with the display and to then incorporate such multiple user input within a display process is also a challenge. Although there have been some successes with the development of multiple-touch interaction technologies (e.g., [1] and Apple’s iPhone), there are significant limits to existing technologies. Frustrated Total Internal Reflection [1] requires a significant empty space behind the display surface to permit projection and interaction monitoring, while the iPhone only supports two point interaction.

In short, the development of a large scale interactive display surface presents a range of technical challenges. The development of a hardware and software solution to this task is the primary goal of the Active Desktop project. An embodiment of the technology described in [2], [3], the Active Desktop (see Figure 1) is an example of an extensible thin interactive display.
surface. The current device provides a display surface of approximately 55"x36" with a pixel resolution of $5040 \times 3150$. This paper describes the basic physical construction of the device, the underlying rendering software infrastructure and the design and development of a multi-input interaction mechanism for the display.

The Active Desktop addresses the problem of actually displaying a large-scale surface through a tesselation of smaller displays each with its own controlling display unit. This provides the necessary parallelism to fill the video display at an acceptable rate. The content for each of the display units is rendered in parallel and a synchronization software layer is developed to ensure that the various elements of the display are properly synchronized. Finally, a unique vision-based multi-touch input display layer is developed to provide multiple-user interaction with the display. The final system is a general-purpose display device suitable for a wide range of applications.

The remainder of this paper is organized as follows. Section II describes the overall system design and the basic physical construction of the device. Section III describes the underlying software infrastructure that provides the necessary display synchronization. Section IV describes the user input technology which is based on the use of a network of video cameras to monitor interactions with the device. Finally Section V describes ongoing plans for the Active Desktop.

II. System Overview and Physical Construction

The design and development of a display and interaction system involves solving two basic underlying problems (i) how to actually develop the necessary hardware to provide the display and interaction and (ii) how to develop the appropriate software infrastructure. The basic approach taken with the Active Desktop is to generate the display through a network of of the shelf standard computer hardware (see Figure 2). The display itself (see Figure 1) is made up of flat panel displays.

The basic physical construction is shown in Figure 3. A collection of nine flat panel displays are mounted within a mechanical structure. These displays are just “standard” flat panel displays which have been extracted from their housings. Connectors from the panels are on their back allowing for the displays to be mounted in a tight tesselation although there are small seams between the individual displays. As the connectors are on the back of the displays (as are the mounting points for the displays) there are no issues involved in mounting the centre unit of the 3x3 matrix of displays. Note that although the displays are mounted in a 3x3 matrix here, there is no reason that the arrays need to be square, rectangular or even planar.

Figure 4 shows the underside of the table structure. The particular flat panel displays have their power and data connections mounted on the rear of the display. This allows connections to be made without resulting in particularly large seams between the panels that make up the display.

The network of computers that drive the matrix of displays are just standard (“off the shelf”) computers with reasonably high performance graphics cards. Some of the elements of the network drive one of the display units while some drive two.

One of the elements in the network is designated as the master node. This computer contains two network interfaces. One which connects to the other nodes in the table, and one which connects to the outside world. The master node does not drive any of the display nodes.

Each one of the nodes in the display run Linux and in particular supports the X Windowing system[4]. Given X’s ability to support virtual displays that span multiple physical displays this enables simple 2D display programs to operate in a straightforward manner and it also provides the necessary software infrastructure on each display node to provide basic
The ability to operate X displays allows for not only a large “normal” display, it is also possible to generate X displays that support windows in arbitrary orientations. Figure 5 shows a sample X display operating over the Active Desktop with a number of traditional X applications (terminal, clock, etc.) operating on it with some of the windows operating at orientations not aligned with the horizontal-vertical raster. This ability is important as users grouped around the Active Desktop may not all be grouped around one “bottom” edge and thus the normal window orientation may not be aligned with the normal display raster.

The process of managing windows with X at arbitrary orientations can be accomplished in a number of different ways. Here we take advantage of the shape and damages extensions to X in order to maintain rectangular windows for the client applications but to render these rectangular windows as non-rectangular on the Active Desktop.

### III. Rendering

Although multi-screen X is supported under X it is not sufficiently flexible for the various rendering tasks that are required. In order to provide a more flexible rendering environment, a software rendering infrastructure known as VE[5] has been adapted to generate synchronized graphics displays that span the elements of the Active Desktop. The VE library provides support and a framework for a number of key requirements related to displays that span multiple physical display units including support for:

- Screen geometry and positioning.
- Parallel rendering of screens on multiprocessor machines.
- A general input device mechanism.
- A simple interpreted language for specifying the physical structure of the environment including screens, input devices and user-specific parameters.

VE does not abstract the actual rendering of the displays, nor is it tied to a specific graphical rendering infrastructure (although here X11 and OpenGL are assumed), rather VE concentrates on providing a common environment to coordinate the rendering processes running on each of the display nodes.

The basic layout of the software infrastructure that drive the Active Desktop and how it is distributed between the master and slave nodes in sketched in Figure 6. VE threads run on each of the display and master nodes (one thread runs for each display attached to a slave node). VE distributes the rendering process across the physical display screens. From an application point of view the user writes a single version of the rendering code and it runs (appropriately configured and synchronous) in each of the rendering threads.

Given this ability to generate and manage large and complex interactive graphical displays it is straightforward to render large and complex 3D environments across the entire Active Desktop. In addition, as rendering on each panel of the desktop
Fig. 7. A Quake model walkthrough operating on the Active Desktop. Each processor has its own local model to render and each processor performs the rendering task in parallel. The display is synchronized by the master node.

is performed locally on that panel’s associated processor, extremely fast rendering is possible. Figure 7 shows a view of the Active Desktop rendering a portion of a Quake world. The basic (static) world model is distributed to the slave nodes prior to rendering. During the rendering phase only dynamic state information needs to be pushed out to the slave nodes. In this case this is primarily information related to the pose of the viewer and simulation clock-related information.

IV. MONITORING USER INTERACTION

In addition to managing the process of rendering the synchronized visual display, each of the computer nodes in the Active Desktop also support a small number of video cameras. The entire collection of video cameras (16 in the current configuration) view interactions on the surface of the Active Desktop.

For each camera a process operates (see Figure 6) that monitors the individual camera and processes the signal monitoring finger-table-top interactions. The process of extracting the finger table-top interactions is formulated as a finger-centroid detection task (see Figure 8). Here the output of the finger-centroid detection task is illustrated. Each camera image is processed looking for skin coloured regions at the table surface. Given the angle of incidence the table surface acts as a mirror and finger table-interactions appear over a large range of rows although at a common column in the image. The rows near the table surface are identified through a calibration process and coherent regions identified as a finger-table interaction. In Figure 8 the rows of the image that are searched for fingers interactions are highlighted in red, and portions of that region that are “finger-coloured” are painted without the highlight. A white bar is drawn at potential finger columns. Note that many different finger positions are possible for each camera.

The actual process of detecting possible finger regions is performed using a Bayesian detection algorithm based on colour information. Following basic approaches in general skin detection (see [6] for a review of Bayesian skin detection), \( P(\text{finger}|\text{colour}) \) is computed based on priors of \( P(\text{colour}|\text{finger}) \), and \( P(\text{colour}|\text{not-finger}) \). These priors are computed by collecting a cadre of sample images of finger-table interactions and views of the table without finger’s present. \( P(\text{colour}) \) is computed in a similar manner.

Each camera transmits its finger columns to a process running on the master node of the display (see Figure 6). Each finger column corresponds to a direction to a potential finger-table interaction. Figure 9 shows the constraints imposed by a finger touching the table at a particular point in the lower left corner of the table. This finger touch is viewed by a number of cameras mounted around the table edge. Each camera that views the interaction introduces a linear constraint between the camera nodal point and a line passing through the interaction. We identify finger-tabletop interaction points by identifying regions of the table that are associated with large numbers of such constraints. Here we use a Hough transform[7] to identify such points. This allows us to identify multiple tabletop interactions concurrently.
V. Summary and Future Work

The Active Desktop is an enabling device that can be used in a wide range of applications. From the application developer’s point of view the device is simply an extremely large display surface which can be painted in a number of different ways, from treating it as a single (but large) X display surface to treating it as a collection of coupled display threads operating on a collection of workstations. To date, the Active Desktop has been used to display standard X applications as well as supporting a range of parallelized rendering tasks including MPEG (Moving Pictures Experts Group) decoding and Quake level walkthroughs.

Multiple touch interaction with the display surface is monitored by a network of cameras that view the user’s finger when in contact with the display surface. A Bayesian skin classification system is used to identify directions to possible interactions for each camera and then these directions are used within a Hough voting scheme to identify possible interaction locations on the surface of the Active Desktop. This approach supports multiple concurrent interactions with the display. Unlike [1] the multiple touch interaction solution described here can be implemented without the need to be able to view the display surface from behind.

Future work includes developing a range of applications for the Active Desktop and developing finger tracking techniques so that the individual finger detections can be decomposed into multiple finger tracks to support drag and similar display interactions.

Acknowledgments

The authors would like to thank NSERC, NCE IRIS, York University and Precarn for their generous financial support. John Tsotsos holds the Canada Research Chair in Computational Vision.

References