

# A Framework for Networked Interactive Surfaces

Tom Cuypers    Karel Frederix    Chris Raymaekers    Philippe Bekaert  
Hasselt University - tUL - IBBT  
Expertise Centre for Digital Media  
Wetenschapspark 2, 3590 Diepenbeek, Belgium  
{firstname.lastname}@uhasselt.be

## ABSTRACT

The development of interactive surfaces has led to a number of applications as it allows for natural interaction and collaboration. The use of co-located collaboration on this kind of surfaces provides new possibilities. It is our belief, however, that an even higher degree of collaboration can be achieved by overcoming the boundaries of a single device or setup. Therefore, we extended our previously built multi-touch framework to realize collaborative multi-device setups. In order to assess the framework, a networked virtual world and three other applications were built to demonstrate its power.

**Index Terms:** H.5.3 [Group and Organization Interfaces]: Collaborative computing—

## 1 INTRODUCTION

Interactive displays are often used as a platform for co-located collaboration. Especially well suited for this purpose are multi-touch surfaces as these allow multiple users to interact with the same interface simultaneously, using their hands directly on the interaction surface. This leads to a more natural collaborative interaction as opposed to the more limited possibilities of WIMP interfaces. Possible applications of interactive surfaces range from building and pre-visualisation tools [2] to large public displays [17].

Whilst very useful, the idea of co-located interaction can be extended even further. By letting multiple interactive setups communicate with each other over a network, a highly interactive distributed collaboration platform can be achieved. Through the combination of multi-touch interfaces and video conferencing, applications such as virtual environments can be created where people collaborate closely and intuitively whilst not necessarily being on the same location, thus reducing potential travel costs and time.

In this paper, we present a software framework for building applications that support collaborative interaction over a network. This framework provides the basis for synchronization of various single- or multi-touch setups in a network, and the development of collaborative applications.

## 2 RELATED WORK

In 2000, Jun Rekimoto et al. [18] came up with Multiple-Computer User Interfaces (MCUIs). They investigated interaction techniques that can overcome the boundaries among multiple devices. Recent developments in multi-touch technology has led to many applications, including collaborative problem-solving and decision-making.

### 2.1 Multi-Touch Displays

In 1984, Lee et al. [14] implemented an active capacitive sensor matrix that was capable of tracking multiple touches simultaneously. It had a very small resolution and suffered from interference between

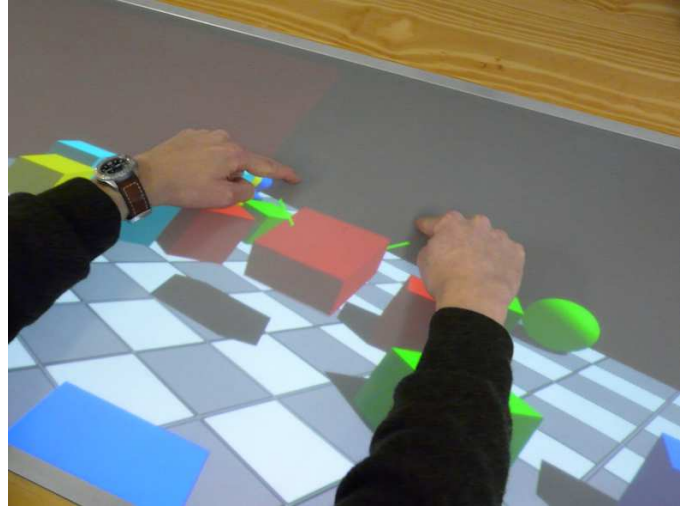


Figure 1: Interacting in a networked virtual environment using a table-like form multi touch interface

sensors. A few other examples of early multi-touch devices are the MTC Express [22] and JazzMutant [15]. In 2002 Mitsubishi Electric Research Laboratories introduced the Diamond-touch multi-user input surface together with the DiamondSpin Software, allowing a limited amount of identified users to interact together on a digital screen [20, 7]. In 2005, Jeff Han [10] created a vision-based multi-touch surface. This technology known as the multi-touch sensing through frustrated total internal reflection (FTIR), allows an unlimited unidentified amount of users to interact on a screen at the same time. The Microsoft Surface is a well-known example of this technology [4]. It is a 30-inch display in a table-like form, where a small group of people can interact with it by using touch, natural gestures or placing physical objects on the screen. Multi-touch displays are not limited for big screens; smaller devices like the well-known Macintosh iPhone also make good use of this technology.

### 2.2 Multi-Touch Frameworks

Since 2006, there has also been quite a lot of research into multi-touch user interfaces, largely thanks to Jeff Han. Many people have started building their own hardware, and some commercial ones are also starting to become available. Xenakis [1] and reactIVision [12] propose a framework which is suitable for multiple users, yet their interaction techniques mainly focus on recognizing surface patterns. IntuiFace [11] is a commercial platform for designing applications based on multi-touch interactions. Their software framework is not available as a separate solution, but it is linked together with their multi-touch hardware device. They focus mainly on single-device collaborative applications, although it is possible for multiple IntuiFace devices to communicate with each other.

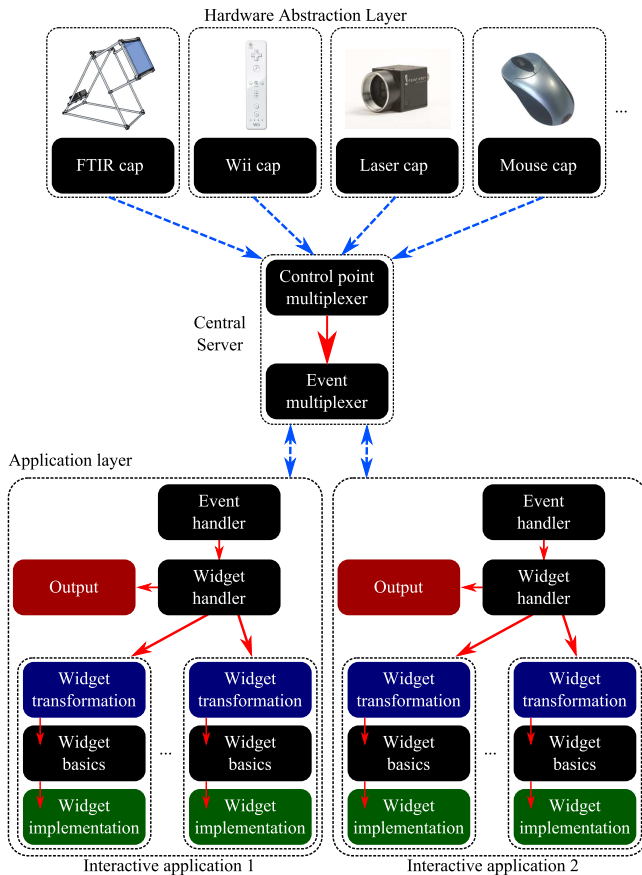


Figure 2: Framework is divided into 3 layers: Hardware Abstraction Layer (HAL) for transparent use of touch point, a Central Server (CS) for transparent use of touch points and events from different multi-touch devices and a Application Layer (SL) for application development.

Other multi-touch architectures like Tisch [8] focus on hardware independency, but still assume that touchpoints are coming from the same hardware.

### 3 FRAMEWORK

We propose a framework for collaborative interaction between multiple users on different interactive surfaces to create hardware independent multi-touch and multi-user software. In the next section we will give a short overview of this framework which is divided into three layers. These layers create a transparent interface between hardware and software on one or multiple computers. In the other sections they will be explained in more detail.

#### 3.1 Framework Overview

An overview of this framework is illustrated in figure 2. It is separated into three layers, which can be compiled as different executables, communicating with each other using UDP packages. This allows that different layers can transparently run on different computers. The first layer is called the hardware abstraction layer (HAL), which serves to record and stream touch points of a specific interactive device. These touch points contain the screen coordinates in a range between 0 and 1 to be resolution independent on different interactive surfaces. Together with other information, like the display size, these touch points are streamed to the central server. This server, which constitutes the second layer, receives the touch points

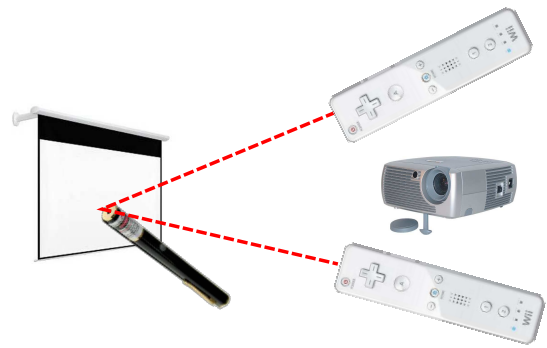


Figure 3: Wiimotes for multi touch interaction: 1 or more Nintendo wii controllers are used to detect the infrared light pen which the user can turn on and off to create touch points

from several hardware devices and events from different applications. Further it will be synchronized and merged together. This multiplexed information is then broadcast to all requesting applications, so they can be used to control different implemented widgets.

#### 3.2 Hardware Abstraction Layer

This hardware specific layer records touch points from a certain interactive surface and stream it to a server in a uniform way. Therefore the application can be created independent from the hardware device. We already added a few types of single and multi-touch devices to the framework as for example a multi-touch table based on frustrated total internal reflection [10].

The multi-touch table makes use of computer vision techniques to locate the multiple touch points. The screen consists of an acrylic plate in which infrared light is injected. On top of the acrylic there is a compliant surface film and a rear-projection film. Touching the projection film will cause optical contact between the compliant surface and the acrylic, resulting in local scattering of the infrared light. An infrared camera, located below the surface, is able to detect the scattered light. After a brief calibration between the projector and the camera, the coordinates and size of all the touch points can be obtained correctly. This is done in the *FTIR cap* component of the hardware abstraction layer.

Another multi-touch interface, as illustrated on figure 3 is based on Chung Lee's interface [3] and uses 1 or more Nintendo wii controllers. Each controller is able to record the 4 brightest infrared light-sources in a scene at a frequency of 100Hz. We use infrared lights which can be turned on and off to 'touch' a projection screen. The captured coordinates of these lights are streamed from the wii controllers through bluetooth to the *wiimote cap* component and converted to screen coordinates. Therefore a calibration preprocessing step is required. The use of multiple controllers allows to compensate for occluded touch points.

A similar kind of multi-touch interaction uses red laser pointers instead of infrared lights to point at the projection screen. The location of these projected red dots can then be captured by one or more RGB camera's. The *laser cap* component receives the images from this camera and converts them to screen coordinates.

#### 3.3 Central Server

The central server is divided into 2 components. The first component or *control point multiplexer* is responsible for receiving touch points from several devices and will synchronize and merge them together. These points are then passed to the *event multiplexer*. This second component also receives events created by different appli-

cations and broadcasts these together with the touch points to all requesting applications.

### 3.4 Application Layer

The third layer of the framework can also be divided into multiple components. The main component or *widget handler* serves for handling the touch points, controlling different widgets and generating output. Other components are the *event handler* and the widgets which are objects that the developer can create and add to the program to create new applications. For this layer, we extended our existing Eunomia framework [5].

The *event handler* is responsible for obtaining the touch points from a single hardware device or from the central server. These touch points are processed to add a unique identifier which is persistent over time and passed to the *widget handler* as an event. This main component has a list of widgets which he controls and executes. It associates the touch points with the right widget by its location on the screen. This component is also responsible for creating the output which can be displayed on the interactive surface.

Widgets receive the touch points through event passing. These touch points are first used for transforming the widget. Its location, scale and orientation are adjusted by the movements of the touch points. When the developer creates a new widget for the program, he can keep these transformations or override them with his own implementation. The *basic implementation* of the widget contains functionality for rendering and event passing. Event passing is used to let widgets in the same program communicate with each other, but also for communication between widgets on different devices. In case of multiple devices, these events are passed to the central server, which will distribute them to all the requesting interactive surfaces.

The ability to orient widgets is a very important aspect of tabletop collaboration, as explained by Kruger et al. [13]. They pointed out three roles of orientation in this context: comprehension, coordination and communication. Users can orient objects, like for example pictures or pdf files, to be most readable for themselves (*comprehension*). An object's orientation can indicate its ownership (*coordination*). An object could also be oriented by one collaborator to another collaborator, indicating that person 1 is communicating directly with person 2 (*communication*).

## 4 IMPLEMENTATION

In this section, we will go more into detail about the implementation of the framework.

### 4.1 Language

The application layer and central server is fully implemented using the C/C++ programming language. This language is chosen because of its speed and amount of available libraries. The hardware abstraction layer on the other hand is implemented using different languages. Some of them like the *wii-cap* is implemented in C#. Because of the communication through UDP packages, this layer is completely separated from the rest of the framework and can therefore be implemented with any programming language on all platforms.

The graphical user interface is created using the OpenGL library, and the FFmpeg [9] and Devil [6] library are used for loading media content.

### 4.2 Central Server

The input control points from each device are sent directly to the central server through UDP. The server then adds some metadata, like for instance the host IP address, scales the coordinates to the [0, 1] interval, and sends them further to all subscribed clients (also using UDP). The server provides the possibility to add unlimited IP

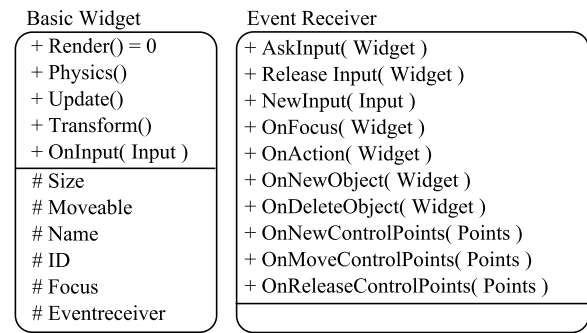


Figure 4: An overview of some basic properties and functionality for implementing widgets.

addresses, or there is also the option to use broadcasting or multicasting instead of sending to the clients directly. The event system works more or less the same way. When a client generates an event, it sends an event request to the server in XML format. The server then checks this request and if it is valid, the server send an event message to each of the subscribed clients (also in XML format).

### 4.3 Event Ordering and Critical Section

The applications we implemented do not need synchronisation, but other applications may require to know the order of events. Because the program can run on different devices there is no global clock. To solve the event sorting problem a vector clock [16] can be used. This is a logical clock added to every event making it possible to define its logical order.

Another problem that can occur is data that becomes inconsistent on different devices. To prevent this from happening, Picart and Agrawala [19] described an optimal solution to use these logical clocks to create a critical section.

### 4.4 Implementing Widgets

Creating a new widget or application is a simple and straightforward procedure. A new widget must be inherited from the basic widget class. This class provides the widget with a set of functions and properties such as size, transformations, a unique ID, etc. New functionality can then be added to the inherited class. The main obligated function is the render function, defining how this widget is rendered. Other functionality which can optionally be reimplemented are events such as touch points events, physics updates and input events. An overview is shown on Figure 4.

In each application there should also be a central event receiver. If the application only exists of a single widget, then that widget can also function as the event receiver. A pointer to the event receiver is passed through to each widget, to let it know which events can be sent. The implementation defines which events are caught and what will be executed. Example events are new input, focus changed and widget actions.

## 5 APPLICATIONS

Based on this framework we implemented a virtual environment where multiple users can interactively build or manipulate a virtual world. We also describe three other applications on top of our framework to show its power. The first application is a virtual environment, which allows to collaboratively move objects. A second application a collaborative media browser to show the collaborative visualization of information on several devices. The third application is a multiplayer pong game. This is meant to show that the latency is low enough to play interactive games. The last application is a collaborative drawing canvas to illustrate non-co-located collaboration, and the events passing between the multiple devices.

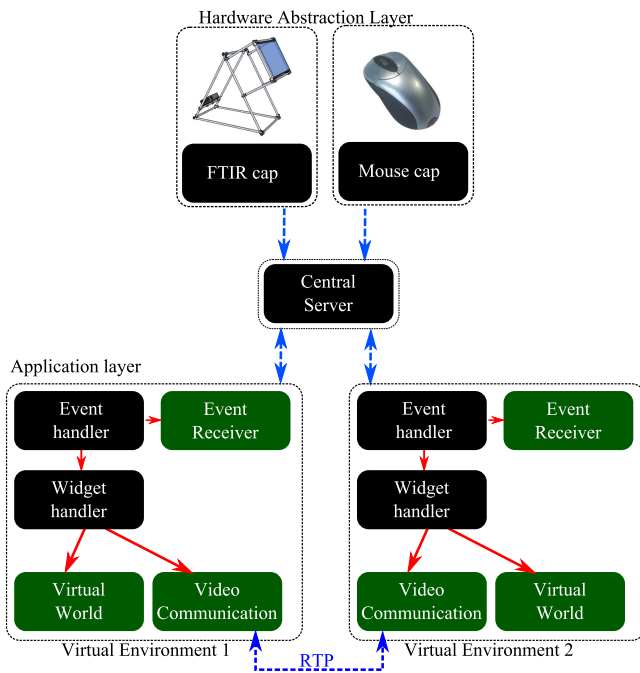


Figure 5: An overview of the implementation of a interactive networked virtual environment using our framework.

### 5.1 Virtual Environment

As main application we developed a virtual 3D world. The interactive display can be used for navigating through this environment. In this world we place several types of building blocks like cubes and spheres. Each user can move or modify these blocks by applying forces onto them, using the interactive interface. These forces and animations are handled using the ODE physics library [21]. This way several users can simultaneously modify this world from a single or multiple interactive displays. The virtual environment is hosted on the first device starting this widget. Forces coming from other devices and updates regarding the virtual world are handled through the event handler. An example of this application is presented in Figure 6(a). Although the objects' positions and orientations are synchronized, they don't share the same viewpoint. This allows the participants to work on different parts of the virtual world or to both have a different view on the same objects. As the users don't have to be physically located in the same room, they need another communication method. We therefore implemented a video and audio communication widget which streams recorded audio and images from the webcam over the network to the other devices, making use of the RTP protocol. The implementation of this application is presented in figure 5. New components that were implemented for this application are marked in green. The virtual world class is inherited from the basic widget class. New events need to be added for things like world state updates. The video communication widget is also based on the basic widget class, but it makes use of RTP to stream audio and video peer-to-peer over the network.

### 5.2 Collaborative Media Browser

The second application is a collaborative media browser. This application lets the user view different kinds of media widgets like photos, videos and digital books. These objects can be rotated, translated and scaled using one or more touch points. These touch points can originate from the device the user is working on or from a partner who is working on a networked device. In order to visual-

ize the other party's actions, the touch points are also shown on the display. By using different colours for the different participants, the movements of a participant's fingers can easily be followed.

For easy communication between multiple users the audio and video communication widget from the virtual environment can be used as shown on Figure 6(b). This also applies to other applications created upon this framework.

### 5.3 Multi-Player Multi-Touch Video Game

For entertainment purposes we show a multi-player pong game. In this game every player has one pallet which can be controlled using a multi-touch device. This pallet is a widget which can be manipulated like the media widgets from the media browser, with the restriction that it cannot be scaled. Furthermore, the pallet only responds to the touch points coming from the same device as the software is running on. Because of the low latency of the network, the players could challenge each other with minimal delay. An example of this game is shown on Figure 6(c).

### 5.4 Collaborative Drawing

For this final application, we created a simple drawing canvas widget on top of our framework as shown on Figure 6(d). The idea is to let multiple users collaborate on a single drawing. When a user makes a change, the event gets broadcast to all participating users. This means that every user in the network can contribute, and each user gets assigned a different colour. This way, the input from all different users can easily be distinguished.

## 6 DISCUSSION

The use of networked collaborative applications provides a next step in the development of collaborative applications. As users don't need to go to the same location, it reduces travel time and money, while still providing the same collaboration as co-located applications as the participants can see each other and communicate by means of video conferencing.

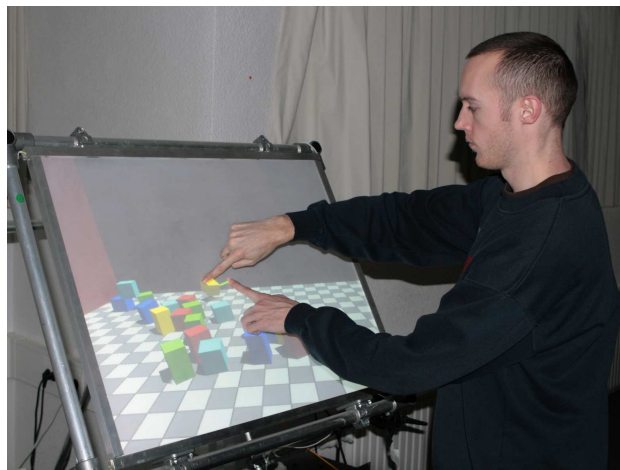
Another advantage of networked collaboration is the possibility to involve more people in the collaboration process without getting in each other's way. Unlike with co-located collaboration, the participants don't have to stand in front of the same screen. Awareness of each other's action is still possible as a unique colour is assigned to each user. However, problems can still arise when too many people work together. First of all, the video streams take up screen estate. It is possible to hide or scale down certain videos, but this only provides a limited solution. Furthermore, when too many participants are involved, it is necessary to introduce a locking mechanism as it becomes more difficult to be aware of which widgets are already being used. This could, however, reduce the flow of working.

## 7 ACKNOWLEDGEMENTS

Part of the research at EDM is funded by the ERDF (European Regional Development Fund) and the Flemish government. The work presented in this paper was funded by IBBT through the HI-Masquerade project.

## REFERENCES

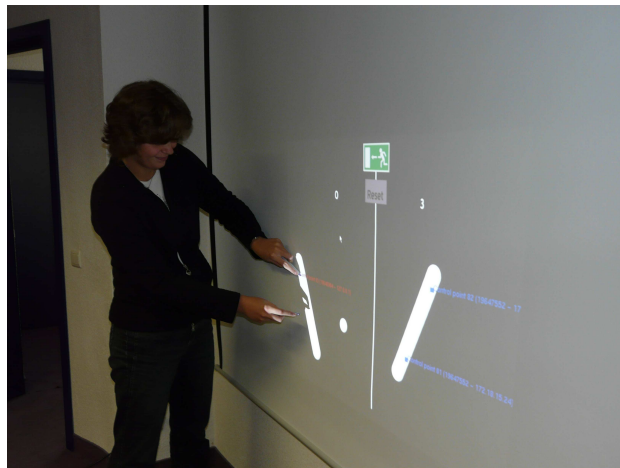
- [1] M. Bischof, B. Conradi, P. Lachenmaier, K. Linde, M. Meier, P. Pötlz, and E. André. Xenakis: combining tangible interaction with probability-based musical composition. In *TEI 2008*, pages 121–124, 2008.
- [2] M. Cardinaels, K. Frederix, J. Nulens, D. Van Rijsselbergen, M. Verwaest, and P. Bekaert. A multi-touch 3d set modeler for drama production. In *IBC2008*, pages 330–335, 2008.
- [3] J. Chung Lee. Low-cost multi-point interactive whiteboards using the wiimote, 2008. <http://www.cs.cmu.edu/johnny/projects/wii/>.
- [4] M. Corp. The history of microsoft surface the making of microsoft's first commercially available surface computer., May 2007.



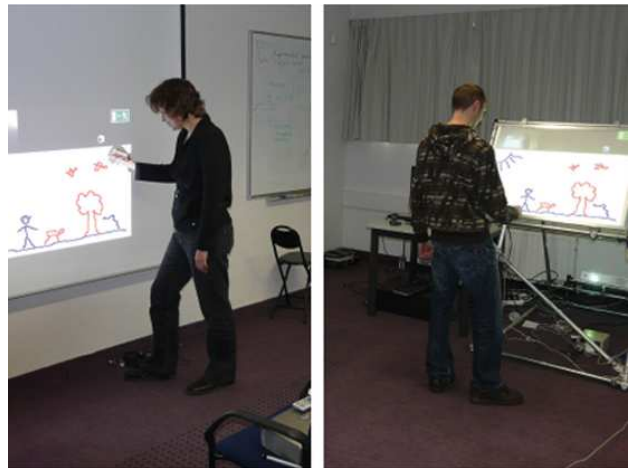
(a) Virtual Environment



(b) Collaborative media browser



(c) Multi-Player Multi-Touch Videogame



(d) Collaborative Drawing

Figure 6: (a) A networked virtual world which can be manipulated by applying forces using interactive displays (b) Browsing through videos on a vertical multi-touch display while video chatting (c) Playing pong using infrared infra-red lights tracked by wii-motes. (d) Collaborative drawing on a vertical multi-touch display while video chatting.

[5] T. Cuypers, J. Schneider, J. Taelman, K. Luyten, and P. Bekaert. Eonomia: Toward a framework for multi-touch information displays in public spaces. In *HCI 2008*, pages 31–34, 2008.

[6] Devil. <http://openil.sourceforge.net/>, 2009.

[7] P. Dietz and D. Leigh. Diamondtouch: A multi-user touch technology. In *UIST 2001*, pages 219–226, 2001.

[8] F. Echtler and G. Klinker. A multitouch software architecture. In *NordiCHI 2008*, oct 2008.

[9] FFMpeg. <http://www.ffmpeg.org>, 2009.

[10] J. Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *UIST 2005*, pages 115–118, 2005.

[11] Intuilab. Intuiface. <http://www.intuiface.com>.

[12] M. Kaltenbrunner and R. Bencina. reactivation: a computer-vision framework for table-based tangible interaction. In *TEI 2007*, pages 69–74, 2007.

[13] R. Kruger, S. Carpendale, S. D. Scott, and S. Greenberg. Roles of orientation in tabletop collaboration: Comprehension, coordination and communication. *Comput. Supported Coop. Work*, 13(5-6):501–537, 2004.

[14] S. Lee, W. Buxton, and K. C. Smith. A multi-touch three dimensional touch-sensitive tablet. In *CHI 1985*, pages 21–25, 1985.

[15] Lemur. Jazzmutant. <http://www.jazzmutant.com>.

[16] F. Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms*, pages 215–226. North-Holland, 1989.

[17] P. Peltonen, E. Kurvinen, A. Salovaara, G. Jacucci, T. Ilmonen, J. Evans, A. Oulasvirta, and P. Saarikko. It’s mine, don’t touch!: interactions at a large multi-touch display in a city centre. In *CHI 2008*, pages 1285–1294, 2008.

[18] J. Rekimoto. Multiple-computer user interfaces: “beyond the desktop” direct manipulation environments. In *CHI 2000*, pages 6–7, 2000.

[19] G. Ricart and A. K. Agrawala. An optimal algorithm for mutual exclusion in computer networks. *Commun. ACM*, 24(1):9–17, 1981.

[20] S. D. Scott and S. Carpendale. Guest editors’ introduction: Interacting with digital tabletops. *IEEE Computer Graphics and Applications*, 26:24–27, 2006.

[21] R. Smith. Open dynamics engine. <http://www.ode.org>.

[22] S. F. Technology. Tactex. <http://www.tactex.com>.