Concurrent Manipulation of Multiple Components on Graphical User Interface

Kentaro Fukuchi

2006.10.23

Acknowledgments

I would like to thank the many people who have helped and assisted me on the path towards this dissertation.

I thank Prof. Satoshi Matsuoka, for the past 10 years of encouraging and endurance of my procrastination. Without his great help, I could not write any piece of this thesis.

I next thank Prof. Hideki Koike. The critical path of this thesis was saved by his kind and careful advises.

The main part of this research was achieved with the great help of Dr. Jun Rekimoto. He saved my career when he told me that SmartSkin was developed and needed applications for it. That was my turning-point.

I also thank Prof. Masaru Kitsuregawa and Prof. Masashi Toyoda, for giving me the opportunity of developing an application based on the great achievements.

Thanks to many current and former people in the Matsuoka Lab. and Koike Lab. Every time I visited to the room, they told me their exciting current research topics, including extremely funny ideas in some cases.

I acknowledge and appreciate the support of my family.

Finally, I dedicate this thesis to my friends.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Currently most computer systems used in the office or at home employ a **Graphical User Interface** (GUI). In GUI systems, graphical components are displayed on a screen and the user manipulates the components using a **pointing input device,** such as a mouse or a pen. Conventional GUIs provide only one pointing input device, and the user can manipulate only one component at a time.

On the other hand, in daily life, we manipulate two or more objects concurrently and naturally. For example, although while driving a car, it may appear that a single object, i.e., the car, is being manipulated by the driver, in fact, the driver is simultaneously manipulating the steering wheel using his hands and the pedals using his feet. In addition, the driver occasionally pushes buttons on the console using one hand while controlling the steering using the other hand. Moreover, experienced drivers can perform complicated operations such as stepping on the clutch pedal with one foot while stepping on the brake and the accelerator pedals with the other foot, controlling them simultaneously.

In addition, several special purpose machines require concurrent manipulation, particularly machines that operate in real time. For example, an audio mixer has many sliders – over a hundred in some cases – that are used to adjust the volume of individual sound sources, and these sliders can be manipulated independently. A skilled operator manipulates these sliders concurrently using both hands.

These examples suggest that the user wants to control the machine more closely, precisely or with a high degree of certainty, and that the interfaces of such machines are designed to satisfy the requirements for such control. In other words, the users wants to control the machines certain degree and the interfaces of such machines are designed in such a way as to effectively transmit their intentions to the machines.

On the other hand, when such a machine is reproduced using computer software, a mouse and a keyboard are generally used to transmit the intentions of the user to the software. However, can a mouse be a substitute for a steering wheel, pedals, or dozens of volume sliders? In many cases, the user is frustrated with the conventional input system for operating these types of software. A steering wheel and pedal device can be purchased for racing games, and, for audio mixing software, a mixing console that can be connected to the computer is highly recommended for professional use.

However, the use of such devices means increased cost to the user and a larger hardware footprint. One reason why computers are used for various applications is their flexibility, computer hardware can be used for various applications by simply changing the software. Therefore, flexibility is an important subject when considering advanced input devices.

The goal of this research is to provide a generic input system that allows the user to concurrently access multiple components of a GUI system, so as to transmit with the greatest fidelity the intentions of the user to the computer software.

1.2 Subject

1.2.1 Input device

The subject of this thesis is the development of a technique for transmitting the intentions of a user to an application efficiently and with a high degree of certainty. In order to improve the efficiency, several methods exist in various hierarchies, from application improvement to training techniques for the user. The input device, which is the interface between the user and the computer software, is considered herein.

The input device is the first object the user touches in the process of interacting with the computer. The input device drastically limits the amount of information that is transmitted from the user at the beginning of the interaction process. The amount of information that the user is assumed to be greater than that which can be transmitted to the computer using a mouse or a touch panel. As such, in the present thesis, we attempt to develop input device capable of providing wider transmission paths and replacing existing input devices.

1.2.2 Concurrent manipulation

When a user uses an application, by definition, the user changes the internal states of the application. Therefore, it is important to allow the user to change the internal states freely. In general, an application has multiple internal states. Using the example of the car, the car has various internal states, such as position, speed, acceleration and steering angle. Even though the driver simply wants to arrive at the the destination, he must control sufficiently the internal states of the car during operation. For more advanced operations, such as a car race, the driver must change multiple internal states quickly and precisely, which requires an adequate control.

Applications provide the user with components for changing its internal states. Most applications offer multiple components as GUI widgets on a screen for changing multiple internal states. However, as stated previously, since the user can manipulate only one component at a time using an input device, at any time, the internal states that can be changed are restricted by the bounds of component. If the input system allows the user to manipulate multiple components simultaneously, then multiple internal states of the application can be changed.

Therefore, in the present thesis, we attempt to develop an input system that will allow concurrent manipulation of multiple components to control multiple internal states of an application bound by these components.

1.3 Approach

1.3.1 Requirements of a concurrent input system

We herein attempt to develop an input system that enables concurrent manipulation with a high degree of flexibility that can be used for several applications. The requirements of the proposed system are summarized below.

1. Ease of use, without any special body equipment.

2. The system uses the same type of sensation as concurrent manipulation systems used in daily life.

3. The system is not specific to certain application and can be used for various applications.

4. The system does not restrict the software.

5. The system requires less effort to apply it to the conventional applications.

1.3.2 Interaction techniques for concurrent manipulation

Two interaction techniques were proposed for concurrent manipulation, **multipoint input** and **bulldozer manipulation**.

A multipoint input solves the limitations of the conventional input systems, whereby only one component can be manipulated at a time. This allows the user to point to multiple components on a screen, and thus to manipulate the components using more than one hand or finger.

In addition, since it is difficult to manipulate more than ten components (the number of fingers), a bulldozer manipulation technique is proposed to enable more massive concurrent manipulation.

In daily life, various parts of the hands, such the edges or palms of hands, are used in object manipulation, such as gathering dust scattered on a desk or brushing off bread crumbs. Similarly, bulldozer manipulation allows the user to gather or brush off components on the screen using his hands.

1.3.3 Implementation

Three input systems were developed in order realize the two interaction techniques described above.

The first system tracks the motion of an input block on the desk by a vision analysis technique. The user can manipulate up to eight input blocks manually to control multiple components.

The second system employs SmartSkin, a sensing architecture based on the capacitive sensing technique, to track the motion of hands or fingers directly. A computer screen is projected onto the surface of the sensor, and the user can touch and manipulate multiple graphical components on the screen using his fingers. In addition, the bulldozer manipulation technique was build into the input system by tracking the shape and the motion of hands on the surface. Various experimental applications were developed and evaluated.

The third system employs laser pointers to point to a screen remotely. The system uses a video camera to track the motion of laser spots on the screen The second system, described above, is intended for desktop applications, which assumes that the user is in front of the computer, whereas this system can be operated at a distance from the screen.

1.4 Contributions

A taxonomy of concurrent manipulations was proposed and prior studies were categorized therein. The goal of concurrent manipulation was then designed. A multipoint input system with physical devices was developed and its effectiveness was confirmed from the results of an evaluation test. A non-device-input multipoint input system on a human-body sensor was developed. An evaluation method to test the effectiveness of a multipoint input system that requires continuous concurrent manipulation was proposed, and an evaluation test was conducted on the system. The effectiveness of the system was confirmed. The bulldozer manipulation technique was proposed for concurrent manipulation without positional input, and the technique was developed with a human-body sensor. A number of applications for real use were built on these input systems and load tests were run, confirming their efficiency.

1.5 Thesis organization

The thesis background of this thesis is first described (Chapter 2). Next, the advantages of concurrent manipulation and the goal of this research are described (Chapter 3). For the following discussion, a taxonomy of techniques for concurrent manipulation is then introduced (Chapter 4) along with an overview of previous research (Chapter 5). Next, the basic design of the input systems for concurrent manipulation are proposed (Chapter 6), and three prototypes are proposed (Chapter 7, 8, 9). The details of these prototypes and the results of evaluation tests are reported in corresponding chapters. Next, a number of real applications for these prototypes is reported (Chapter 10). Finally, conclusions are presented (Chapter 11).

Chapter 2

Background

In order to transmit a request by computer that the user wants the computer to perform and transmit the result from the computer to the user, there are two procedures between the user and the computer:

- request from the user to the computer
- answer from the computer to the user

This transaction between a human and a computer is referred to as a **Computer-Human Interface** or simply a **User Interface** (hereinafter abbreviated **UI**). A well-designed UI helps the user to make the best use of the computer and to achieve the desired goal.

2.1 A brief history of user interfaces

2.1.1 Before the graphical user interface

Since the first computers were used for calculation, early interfaces consisted mainly of the input and output of numbers. For example, the Harvard Mark I had a punch-card reader and card puncher, whereas ENIAC had plugboards and banks of switches[67].

In the era of mainframe computers, most computers still had their own consoles that consisted of a number of switches and lamps. In 1951, UNIVAC I, which was equipped with a keyboard (unityper) and a printer, was released. The keyboard could be used to program the computer using a human-readable programming language.

In the 1960s, the *video display terminal*(VDT) was introduced and became the standard input device in 1970s. It consisted of a keyboard and a video display that could display characters. The VDT enabled the *command line interface*, which simply read the text input by the user and wrote the results onto the video display.

2.1.2 Graphical user interface

Recently, the **Graphical User Interface** (GUI) has become the primary user interface for desktop computers. Computer systems with a GUI have graphic displays that show information to the user, and the user can input requests by an input device while viewing the information.

One of the first GUIs was Ivan Sutherland's Sketchpad[54], which consisted of a graphical display and a light pen that could be used to point to a position on the display and transmit the coordinates of that point to the computer. Like the light pen, an input device that is used to point to a position on the display to obtain the coordinates of the position is called a **pointing device**. Sketchpad introduced the basic style of GUI, that consists of a graphical display and a pointing device.

The basic model of current desktop UI systems can be traced back to Douglas Engelbart's NLS (oNLine System) published in 1962[10]. Engelbart invented the pointing device that is today referred to as the *mouse*[9]. The user manipulates the mouse to move a pointer on a graphical display, and can press a button on the mouse to input specific commands. Before NLS, functions were provided to the user through a physical input console with a number of buttons. NLS replaced these buttons with graphical components on the display. When the same interface is implemented as a hardware console, it is constrained by physical actions and adds cost. In contrast, the software implementation of the interface was free from such constraints and facilitated interface building.

Subsequently, GUI systems were improved primarily though improvements to graphical components on displays. In 1978, Xerox Palo Alto Research Center developed a computer system called Alto[6]. Alto had a GUI system that consisted of windows, icons and menus similar to those seen in present-day GUIs. The framework that Alto introduced is referred as **WIMP** (Window, Icon, Menu, Pointing device), and it has become the primary architecture used in today's GUI systems[61].

2.2 Evolution of the user interface

2.2.1 Development of the computer

The development of the computer has driven the development of the user interface. Since the first computers were used for numerical calculations in most cases, the computers read and wrote sequential numbers or characters (text).

When *mini computers*, such as PDP-8, became available, they were installed in universities, laboratories and offices and came to be used for various applications.

In 1975, the first personal computer, the Altair 8800, was produced, and since that time computers have been employed for personal use. In 1977, the Apple II and the TRS-80 became widely used, again extending the applications of computers. These computers had graphic outputs and were equipped with a keyboard. In 1981 IBM produced the comparatively inexpensive IBM-PC, and the popularity of personal computers grew. With the development of the computer market, many third-party vendors produced software or peripherals for these computers.

In 1983, Apple released Lisa, the first commercial computer that provided a graphical user interface. Apple released the successor to Lisa, the Macintosh, in 1984. The significant advantage of Lisa and Macintosh was that it was shipped with a mouse. Since then, the mouse has become standard equipment with most computers. In 1990, Microsoft released Windows 3.0, and the GUI became the most common user interface for desktop computers. Since high-resolution display and the mouse have become standard, most applications for desktop computers provided GUI.

At present the CPU, memory and graphics chips have become commodities, and powerful yet inexpensive computers are available.

2.2.2 Convergence of input device

While computer applications have become greatly diversified, the input device of the desktop computer has come to consist of a keyboard and a mouse. Before personal computers became popular, each computer system had its own specific console that was specifically designed for that computer. In addition, joysticks, dials and light pens were often used. Engelbart et al. developed an input device called the Chorded keyboard, which had a set of keys like a piano and allowed simultaneous inputs of multiple keys. Such input devices were designed to enable effective control by allowing multi-finger input and are related to the present research. At that time, since the most applications were built from scratch, the input devices could be designed specifically for the target application.

However, when the personal computer became popular, this sort of diversity was reduced. One reason is that supplying specific devices with a personal computer system increases the cost of the system. Another significant reason was that many third-party vendors provided packaged software as the market grew. In order to sell more software, the vendor targeted the most popular environment. If the software depends on a specific environment, then its potential market shrinks. Hence, the interface for most commercial software was designed for the standard environment, which can be manipulated using standard input devices. On the other hand, for vendors that provide a special device, it is difficult to support many applications that depend on the device.

Therefore, the interface of personal computers converged rapidly. At first, for video game applications, the keyboard and the joystick were popular input devices, but the mouse was soon accepted as a standard input device for GUI applications.

2.2.3 Diversity of applications

As described above, computers are currently used for various applications. At first, computers were used primarily for scientific or accounting calculations. Recently, however, computers are used for various purposes, including databases, computer graphics, and sound processing. Moreover, small commoditized computers with dedicated software can be used for specific applications that require dedicated hardware to solve the problems. Next, the background of the diversity of applications is described.

Rapid improvement in computational power Because the speed of computation has improved greatly over the past 50 years, computer can now perform a huge number of calculations in a reasonable time, which enables the computer to be applied to solve problems that involve several calculations. In addition, this decreases the latency of the response time of the GUI and improves the usability of the desktop computer systems.

High-precision output The performance of graphics processing has improved to the point that high-resolution images can be produced. According to this improvement, computers are used for printing, graphics design, and film production. In music production, since the sampling rate has become high enough for professional use, digital audio processing has become mainstream from the consumer level to the commercial production level. In addition, because of the computational speed, complicated signal processing can be performed on a generic computer to be used for real-time signal processing. Indeed, a number of audio processing applications are used for stage performances.

Increases size of storage devices The size of secondary storage devices has increased sharply, while the cost has decrease. Currently, it is difficult to find a hard disk drive smaller than 100 Gbytes. Therefore, huge amounts of data such as audio or video files can be processed on personal computers. In addition, many individuals store a number of music and movie files on their storage devices, and applications for processing these data continue to evolve.

Commoditizing of the computer Computers have become less expensive and their performance has improved. In addition, the Internet has grown and has become faster, and the transfer of data between computers has become easier. As such, people can now work cooperatively over networks by sharing data, which has boosted the diversity of applications.

2.2.4 Diversity of GUIs

As the graphic processing power of personal computers increased, many GUI applications were developed. The target domain of computer applications has expanded, and various tasks are now run on computers. As a result, application GUIs have become diverse.

As an example, with the improvement of processing power and quality of audio input/output, today many types of acoustic equipment for music production are developed as computer software. In many cases, computerization improves the convenience and thus becomes very popular. For example, reversible operations (undo) or save/load of current status are typically not supported by existing hardware-based equipments, while computer software provides them as fundamental features. These features greatly improve the efficiency of music production.

On the other hand, computer software lacks some of the features of hardware consoles, making some operations impossible or inconvenient. For example, Figure 2.1 shows a hardware-based audio mixing console for professional use, which has more than 200 components (sliders, knobs and buttons) to control various parameters of audio sources. As seen in the figure, the hardware-based console provides a very complex interface to the user.

In contrast, Figure 2.2 shows an audio mixing console implemented as computer software. The components shown in Figure 2.1 are provided as graphical components. By this interface, the basic features of the audio mixer are realized in the computer.

However, the software lacks an important feature of the hardware console. All of the components on the console can be manipulated simultaneously. In fact, skilled operators manipulate the sliders and knobs concurrently using both hands. The software console, however, uses a common GUI system that has only one pointing device, so that the user can manipulate only one component at a time. This is a serious problem for professional use, and an external mixing console device is usually used in order to solve this problem.

2.2.5 Visual language and interface builder

At present, light weight languages are very popular among application programmers and application users alike. In particular, some task specific languages for music or visual production have become important for solving problems that can not be solved with existing applications.

In languages used for music or visual production, there are a number of GUI-based languages with which applications are programmed by arranging graphical components. This kind of languages is called *visual language*. Max[48] and its successor,

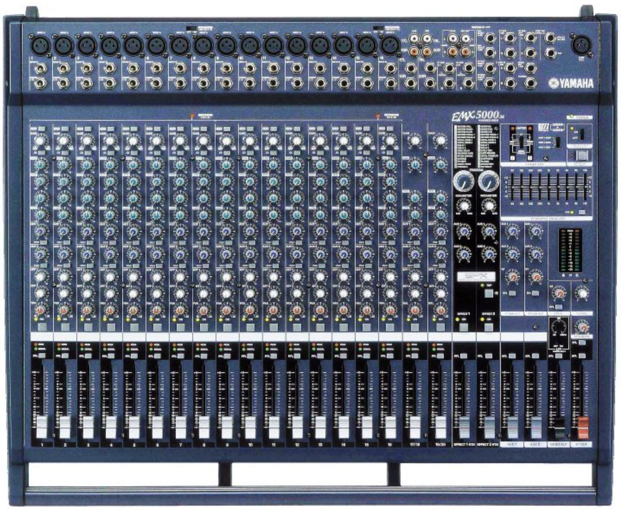


Figure 2.1: An example of a hardware-based audio mixing consoles

This photo shows a YAMAHA EMX-5000-20 console, which can mix 20 audio channels. This image is taken from YAMAHA's web site.

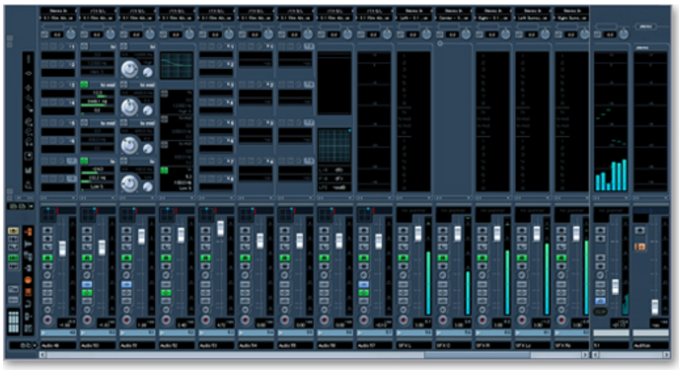


Figure 2.2: An example of a GUI for audio mixing software (Nuendo)

This screenshot of an audio mixing console of Nuendo, music production software developed by Steinberg Media Technologies. This image is taken from Steinberg's web site.

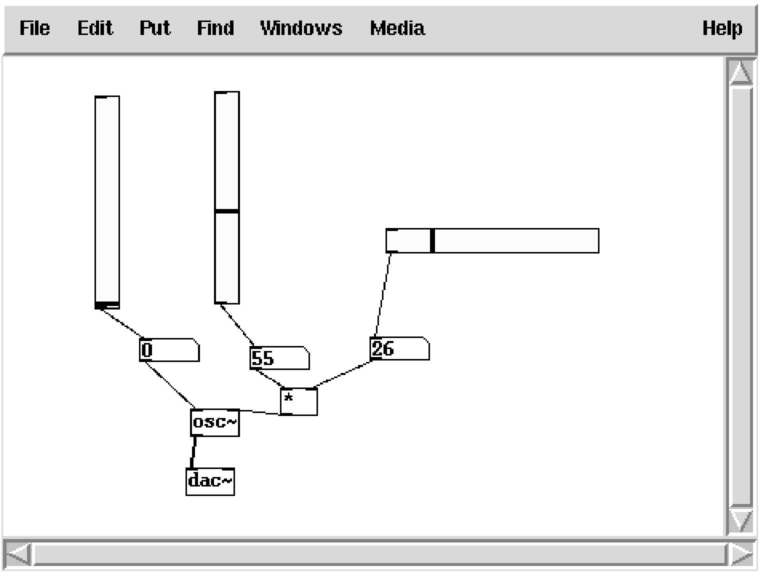


Figure 2.3: An example screenshot of Pure Data

The two vertically-long rectangles are vertical sliders, and the two boxes that are connected to the sliders display their current values. The horizontally-long rectangle is a horizontal slider. The black band of each slider can be moved using the mouse.

Pure Data[49] are visual language for music production, and enabled data flow-based signal processing.

Typically, these languages provide a number of GUI components to accept real-time input from users for parameter control. For example, the user can connect a slider to a component, and its value is transmitted to the component when the slider is moved. Figure 2.3 shows a screenshot of Pure Data.

A complicated program may hold many controllable components. However, as described above, these components cannot be manipulated simultaneously on a conventional GUI system. This is a serious problem for real-time music or visual production.

2.2.6 Architecture of the input device

The mouse and the keyboard have remained the primary input devices ever since Douglas Engelbart introduced them with NLS. Although the usability of the mouse, for example, has been improved, its basic function, i.e., pointing to a position on a display and clicking a button, has not changed.

One improvement was achieved by adding a scroll wheel, or tilt wheel, to the mouse. The wheel is manipulated by a finger, enabling various manipulations of a component but the target of mouse operation remains only a single component. However, by the combination of keyboard and wheel operation, some applications allow switching of the target component without moving the mouse. This indicates the potential demand by the user for select a target component from among many components on a display as quickly as possible.

As alternative pointing devices, there are pen tablets and touch panels. Both recognize a pointing action of the user, and transmit the designated coordinates to a computer.

To solve the problem described in the previous section in which only one component can be manipulated at a time because the GUI system allows only one pointing device, additional input devices are often used. In this case, each application requires a specially-designed input device. For example, a mixing console device, which is similar to stand-alone audio mixers, is used to control the audio mixing software. At present, a number of generic mixing console devices, which are supported by various applications, are commercially available. For example, one device provides a simplified console of an audio mixer (as shown in Figure 2.1), which has eight sliders and eight knobs. Interestingly, not only audio production applications, but also a number of visual production applications, support the device for parameter control. One such application binds a pair of sliders to the X and Y coordinates of a positional input. In addition, a number of visual languages, described in the previous section, also supports such mixing console devices, and it is possible to bind a slider component on the display to a physical slider on the console. However, the positional orders of components and sliders may differ and the system is not intuitive.

2.3 Direct manipulation

Direct manipulation is an interaction method to provide the user with the feeling that he is directly manipulating a component on a display.

A command line interface, which was the primary interface before the advent of the GUI, provides an interactive interface. Here, the user inputs a command by a keyboard to transmit an order to a computer. The subject of the order is one or more of the internal states of the application, but the user cannot change them directly. Instead, the user orders the computer to change them by transmitting the corresponding identifiers, such as a number or a name corresponding to each state.

On the other hand, a direct manipulation-oriented interface provides a visual component that represents an internal state, and the user can point to the component to tell the computer which internal state change, as if touching and manipulating the internal state directly, even when the operation is performed via an external input device. Figure 2.4 represents this relationship. With a command line interface, the user must recall the corresponding identifier to change an internal state. In contrast, in the case

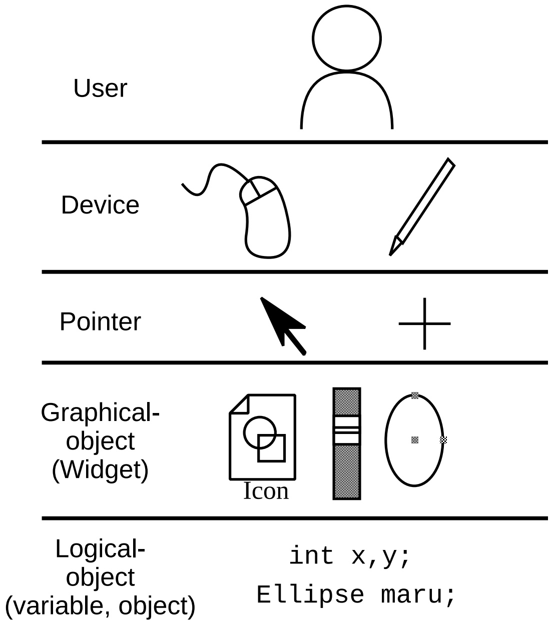


Figure 2.4: Hierarchical relationship between user and application

of direct manipulation, the user feels that he is manipulating the internal states without any intermediate layers.

NLS and Alto are early examples of direct manipulation. Most GUI systems are designed to be manipulated directly, but Shneiderman clarified the features of direct manipulation as follows and introduced a design guide:

- Components in which a user (possibly) has an interest should always be displayed
- A user can interact with a component without the command language
- The result of the request should be immediately reflected by the component on the display

GUI systems usually realize these features with GUI components (*widgets*) and a pointing input device to provide the user with direct manipulation.

2.4 User of the computer for creating art

In this section, a special application field that has special requirements for user interface is described.

Along with the progress of computer technology, computers have come to be used for art and entertainment. The primary reason for this is that computers can now process and output high-definition data. When computers first came to be be used in the creation of art, some users filmed graphics on a display to make an animation, while others recorded individual sound tracks using a multi-track recorder and then edited and finalized the tracks using existing post-production systems.

With the significant growth of computational power, a number of procedures that would have required a significant processing time on an older computer system can now be performed in real time. For example, synthesizing sound waves by software can now be performed in real time, and is thus used in live performances. Moreover, a number of current computer systems are used for live performances on stage.

In particular, in live visual performances, using conventional equipment, it is not possible to generate or arrange visuals on a screen on stage. However, current computers are used to generate real-time computer graphics. Recently, a performing style called **VJ** (Video Jockey, or Visual Jockey) has become popular, in which live visuals are produced on a screen in combination with a DJ or a live performance.

In real-time performance, the interaction layer between the performer and the computer is very important, and there are several situations in which the interface becomes important, including the following:

- Improvised performance
- Using the response of the audience
- Troubleshooting
- Synchronization with the stage performance

In these cases, the performer has to input his intention to a computer quickly and with a high degree of certainty. In addition, the system should accept various inputs.

However, while the output layer of computers has been improved significantly, the input layer has not been improved significantly in recent years. Most performers still use a keyboard and a mouse or a MIDI keyboard. On the software side of the interaction layer, the conventional GUI, which assumes a single mouse input, is employed by most performance tools and has the problem described above. In order to accept a variation of inputs, a complete set of components must be provided on the display. However, this increases the cost of input and decreases real-time interactivity [1]

The poor input problem is a common problem among artists and performers. In the music performance area, a number of conferences and workshops that focus on improving the inputs of performance tools are held annually and artists, scientists and engineers gather and discuss these problems.

[1] For example, when a user inputs characters by a software keyboard, an amount of time is required to move the pointer to each key and targeting.

Chapter 3

Design Goal

As described in the previous chapter, a typical GUI system employs a keyboard and a mouse for the sake of commodity, because most applications use this environment.

On the other hand, as described in Section 2.2, several applications have been developed that cannot be fully exploited by existing input devices, and such applications will likely increase in number as computer use increases.

In this research, the input environment is improved in order to solve this problem for applications that use various and improvised interactions, especially for live performances, described in Section 2.4. The focus here is concurrent manipulation of multiple components on the computer screen provided by the GUI system. When multiple components can be manipulated concurrently and independently, it is possible to transmit the intention of the user to the computer quickly and to vary the interaction.

Next, the effectiveness of concurrent manipulation is described.

3.1 Time-multiplex vs. Space-multiplex

There are two methods by which to change multiple internal states of an application simultaneously: *time-multiplex* and *space-multiplex*.

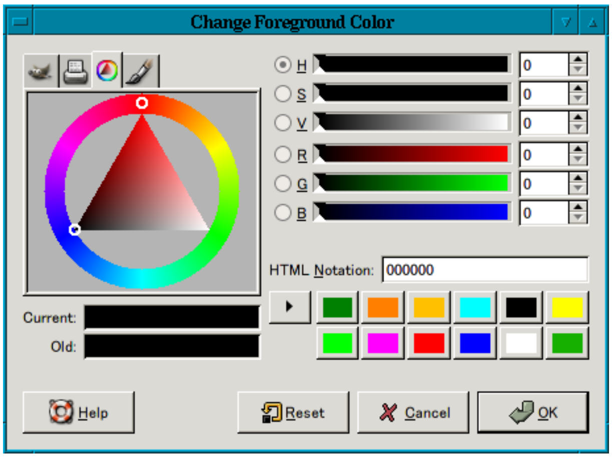
With a time-multiplex interface, the user changes the internal states one by one. By increasing the number of steps of input sequence, the user can input a complicated order to the application. This interface can be built on an existing input environment but requires time to complete the input. Some applications provide intelligent assistants to help the user and decrease the input cost.

On the other hand, a space-multiplex interface allows the user to change multiple states concurrently by providing multiple components. It requires a relatively large interface space, but offers intuitive interaction.

3.2 Components and concurrent manipulation

Let us consider the situation in which multiple components are provided by an application. In most cases, user's attention is focused on one task at a time. Even if the user has multiple tasks, he will perform the task sequentially.

However, as described in Chapter 1, an application may hold multiple internal states, and these states must be changed to complete the task. For example, the task of drawing a figure includes a number of sub tasks, such as decision of its composition, choosing a color, choosing a brush and drawing a line. Of course, these sub tasks can be divided into several sub-sub tasks. In most applications, these sub tasks or sub-sub tasks are bound to icons or menu items. Figure 3.1 shows a screenshot of a console to choose a color of Gimp[19], a drawing application. The console is designed for the task of choosing color, but the GUI contains multiple sliders to change the values of red, green and blue of a color. In other words, the choosing a color task includes sub tasks of changing the red level, changing the green level and changing the blue level, and each task is represented as slider.



● H 0
○ S 0
○ V 0
○ R 0
○ G 0
○ B 0

HTML Notation: 000000

Figure 3.1: A console with multiple components: Color selector of Gimp

As seen in this example, the user must manipulate multiple components even if there is only one task to perform.

3.3 Efficiency of concurrent manipulation

By concurrent manipulation, multiple components on the screen can be manipulated by the user simultaneously. As a simple example, when there are two components, the task can be finished in a half the time by concurrent manipulation. In fact, there are some restrictions of hand motion or limitations of human cognition, so that the improvement may not be in proportion to the number of components manipulated concurrently. However, a number of applications can be performed efficiently.

In addition, multipoint input by the fingers enables various styles of input, compared to single-point input. For example, a conventional mouse translates the motion of the user's hand into positional data, and buttons on the mouse translate clicking motions by the fingers as additional input data. Recently, a scroll wheel has been mounted onto the mouse. The wheel translates the vertical motion of the finger into an additional input. Some wheels also sense horizontal motion. As demonstrated in this example, the motion of the fingers is very flexible but is not used effectively by conventional input devices. If an input system recognizes the motions of the fingers, it will enable more flexible interaction. Concurrent manipulation by multipoint input enables such interaction by dividing an interaction into a combination of manipulations of multiple components.

Concurrent manipulation is common in daily life. Objects that exist independently can be manipulated concurrently. For example, goods on a table can be manipulated simultaneously by using both hands to clear space as needed. Pieces on a chessboard can be manipulated simultaneously during the initial arrangement or at the end of the game. In contrast, in a GUI desktop system, the movement of file icons requires sequential manipulation or icon selection before moving the icons. Concurrent manipulation enables these types of manipulation on a computer and provides an intuitive interaction.

3.4 Forms of concurrent manipulation

In this section, three typical forms of effective concurrent manipulation are described.

3.4.1 Subject includes multiple components

The subject of manipulation exists, and multiple components are provided to enable various interactions. For example, let us consider the situation in which a user has drawn an arc using a graphic editor. In this case, the arc is the subject, but the editor provides a number of components with which to edit the states of the arc, such as the position of its center, the radius, and the start and end angles (Figure 3.2). Another example is color selection, as described previously. When an application provides multiple components for the manipulation of a subject, concurrent manipulation will be effective.

This type of architecture can be seen in many applications. A graphic editor is a good example. Its many features include sub components, which can be manipulated concurrently in many cases. For example, a line has start and end points, which can be moved independently. A rotating operation can be performed by designating the center of the operation and the rotation angle.

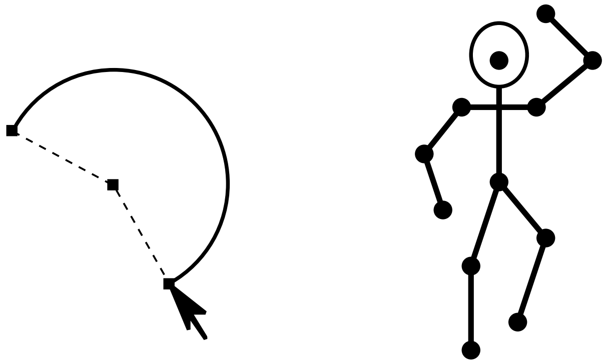


Figure 3.2: Subject containing multiple components

3.4.2 Multiple subjects are controllable

When an application has multiple independent subjects and corresponding components are displayed simultaneously, concurrent manipulation is efficient and enables the components to be moved simultaneously.

For example, when a desktop interface provides multiple file icons on the screen, concurrent manipulation enables them to be moved simultaneously. Using a mouse, the icons are moved one by one, or the user first selects a group of icons and then moves the group. The former is slow, and the latter has decreased flexibility.

In addition, real-time strategy computer games involve a massive number of characters that are controlled by the computer, and these characters can be given orders by the player. Recently, methods of enabling a player to effectively give orders to characters have become important. Concurrent manipulation can therefore be applied to these types of games.

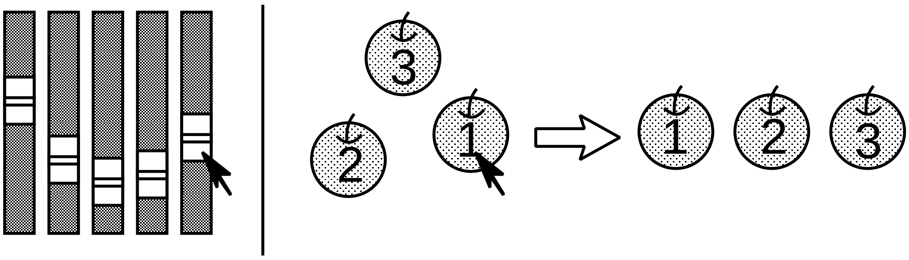


Figure 3.3: Manipulating multiple components simultaneously

3.4.3 Cooperative works by multiple users

When a computer system provides a screen that is shared by multiple users in order to provide collaboration with each other, it is expected that any users can naturally interact with any components on the screen. For example, very large displays are currently used for collaborative works. Typically, two or three users stand in front of a display and manipulate an application on the screen. In these cases, the application should allow concurrent manipulation of multiple components by the users. When concurrent manipulation is enabled on the screen, the system can allow the users to simultaneous interaction in a natural manner, thus enabling collaborative or competitive work. In addition, even for an application that is designed for a single user but employs concurrent manipulation, it is possible that multiple users will share the application and perform collaborative work. In addition, this enables collaborative concurrent manipulation between distant places by using remote manipulation via a computer network. In such cases, the network latency in the systems may cause unexpected results, especially when multiple users manipulate the same object simultaneously.

3.5 Difficulty of applying concurrent manipulation

The nature of concurrent manipulation requires complex manual operation. As such, the ease-of-use requirement is sometimes not satisfied. Moreover, users who have injured hands or elderly users may have difficulty in performing concurrent manipulation. Thus, the input system must not force the user to perform concurrent manipulation.

Applying concurrent manipulation to a conventional application requires a large amount of arrangement. Most applications are designed based on a single-point input system. Even if multiple components that can be manipulated concurrently are included, these components must be manipulated sequentially. For example, in a desktop system, when a file icon is dragged and dropped into the trash can, the system displays a message asking the user “Do you want to erase this file?”, and all interaction is blocked until the user responds. Concurrent manipulation will not function effectively in such cases. Usually, a design change for concurrent manipulation is difficult because exclusive procedures or contradictory inputs must be considered.

Chapter 4

Taxonomy of Interaction Techniques

In this chapter, a taxonomy of interaction techniques is introduced for use in the discussions that follow.

4.1 Single-point input / Multipoint input

Conventional GUI systems allow the user to point to one position on the screen at a time. This type of input system is referred to as a **single-point input**. In contrast, the **multipoint input** allows the user to point to multiple positions at the same time. An input system that does not depend on pointing input is called an input system without pointing.

4.2 Space multiplex / Time multiplex

Fitzmaurice et al. introduced the concepts of the **space multiplex** and the **time multiplex**[11]. A space multiplex interface provides multiple components that can be manipulated by a user concurrently, and the user can control them simultaneously without any previous arrangement. In contrast, with a time multiplex interface, the user can control multiple components after some previous manipulations. For example, manipulating multiple components one by one is an explicit time multiplex interaction.

A multipoint input system enables space multiplex interaction, whereas a single-point input system inevitably requires time multiplex interaction.

An input system can provide space multiplex interaction and time multiplex interaction at the same time. For example, when there are ten components and the user manipulates two of them five times, this interaction is both space multiplex and time multiplex interaction.

4.3 Direct pointing / Indirect pointing

The pointing method can be categorized as **direct pointing** or **indirect pointing**. When a user touches a component on the screen directly or manipulates an input device on the screen, the input system is a direct pointing system. Touch panels and tablet displays fall into this category. In an indirect pointing system, there is an input surface separated from the screen and the user interacts with the input surface. A mouse or a pen tablet without a display fall into this category.

Note that the term **input device** is used to indicate a physical object that is used on an input surface to input positional data.

4.4 Input system with physical devices / without physical devices

In order to input positional data, a system that employs an input device, such as mouse or a pen, and measures its position is called an **input system with physical devices**, or simply a **device-based input** system. On the other hand, a system that allows direct touching, such as a touch panel, is called an **input system without physical devices**, or simply a **non-device-based input** system, or a **finger-pointing** system when the system is used for pointing input. Note that the touch panel is usually referred to as an input device, but in the present paper, the touch panel is categorized as an input system without physical devices, because the position of the touch panel itself is not used for pointing input. Moreover, a system that has the user situate a position sensing device, such as a data glove, is categorized as a system without physical devices, because the user is not aware of the device during manipulation.

The advantage of non-device-based input is that it is not restricted by physical conflicts. The manipulation is free from interference between the input devices and the physical behavior of the device. On the other hand, the accuracy of position recognition of a device-based input system is generally higher. Moreover, the user can manipulate input devices with various parts of his body, and any item can be used to manipulate such systems.

4.5 Specific device / Generic device

In the input system with physical devices, there is a relationship between the input device and a component on the screen, and the relationship must be determined before manipulation. If the relationship is fixed during runtime, then the device is a **specific device**. In contrast, if the device can be related to any component at any time, the device is called a **generic device**. When using a generic device, the user has to **attach** a component to the device. If the component becomes unnecessary, the user may **detach** the component from the device. Let us consider a mouse, for example. The mouse is a generic device, and its positional data is reflected by a pointer on the screen. When the pointer is moved onto a component and a button is pressed, the component is attached to the mouse, and the user can manipulate the component by moving the mouse. When the button is released, the component is detached.

When using a specific device, the shape or color of the device can be customized to the corresponding component. This is intuitive if the outfit of the device is the same as the component on the screen. In addition, the shape of the device can be optimized for manipulation. However, this means that the system must have a complete set of specific devices for the components included in the application. Therefore, the possibility exists that the system will require an enormous number of devices. This requires space to store the devices and the cost of finding the appropriate device. In addition, let us consider the case in which two or more applications are switched on the system. Since the set of components is different for every application, it is required to change the input devices on the input surface. Here, we refer to the set of input devices for an application as the **working set**. When the application is switched, the user must replace the working set with the new set.

On the other hand, when generic devices are used, these devices can be bound to any components of any applications, the outfit of a device cannot be customized. In addition, the user must maintain the relationships between the devices and the components. If there is only one generic device (single-point input), then one relationship should be maintained, which is easy for the user. In a direct pointing system, an input device and a corresponding component are placed in the same place, and an explicit relationship can be seen. In contrast, in an indirect pointing system, the relationships are implicit, and it is difficult to maintain them without supplemental information. The problem of switching of working sets does not occur on a system with generic devices. The user can continue to use the devices for the next application.

4.6 Relative position input / Absolute position input

The input for pointing method that can point to an absolute coordinate on the screen is called **absolute position input**, whereas the input for a pointing method that uses relative coordinates to move a pointer on the screen is called **relative position input**. A direct pointing system, such as a touch panel, uses absolute position input. Some indirect pointing system, such as pen tablets also employ absolute position input. The mouse and the trackball use relative position input.

Chapter 5

Related Research and Systems

In this chapter, previous research and systems that exhibit some of the properties of concurrent manipulation are reviewed. The systems are characterized by the taxonomy introduced in Chapter 4.

5.1 Multipoint input systems with physical devices

5.1.1 Bricks

The Bricks system[12] allows two-handed manipulation with a pair of generic physical input devices. The system consists of a rear-projection table and two six-degrees-of-freedom mice, the positions and orientations of which are recognized.

A drawing application was implemented on the Bricks system for evaluation. On the application, the user can draw a rectangle or a ellipse by pointing to two positions with the mice using his hands. No quantitative evaluation was run on the system, but 20 subjects were trained to manipulated the mice concurrently, and the subjects could eventually draw figures by two-handed manipulation.

5.1.2 Graspable User Interface

The Graspable User Interface[13][11] provides an input environment that allows the use of eight generic or specific input devices simultaneously. The system consists of a 2×2 grid of Wacom's tablet devices, and each tablet recognizes two input devices, for a total of eight possible input devices.

In the evaluation test, a subject was asked to control four components on the computer screen by manipulating input devices on the tablet and to track target objects for which the position and orientation changed every 0.05 seconds. Three conditions of the input environments were compared: with four specific devices, with eight generic

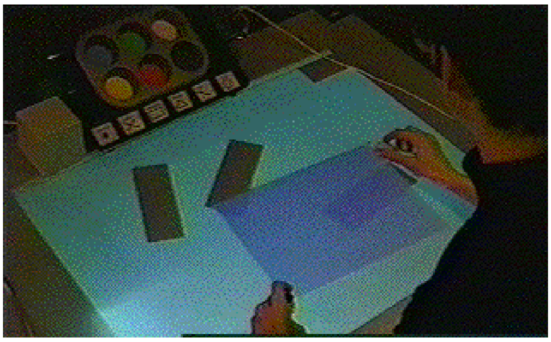


Figure 5.1: Bricks

Two bricks are used to simultaneously translate, scale and rotate the rectangle. This figure is quoted from [13].

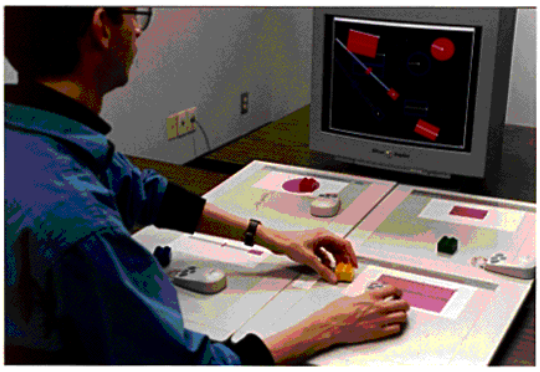


Figure 5.2: Graspable UI

There are eight generic devices on the input surface. This figure is taken from [13].

devices, and with one generic device. The specific devices were reported to provide the best performance.

The implementation of this multipoint input system is different from the proposed implementation in following manner. First, in Fitzmaurice's system, generic devices were attached to their corresponding components from the beginning, and the detaching operation is not described. In the proposed system, as described in Section 7.4.3, the attaching/detaching operation is described as essential for a multipoint input system with generic devices, and the operations are implemented in the prototypes describe herein. Second, the input devices of Fitzmaurices's system were too big to manipulate multiple devices by fingers simultaneously. In addition, each device could be manipulated on a corresponding tablet. In contrast, we implemented an input system that allows the manipulation of eight devices on an input surface freely.

5.1.3 DoubleMouse

DoubleMouse[36][37] is a multipoint input system that consists of two mechanical mice. The system allows the user to manipulate two components concurrently. In addition, they introduced a number of techniques that use the double mouse, including selection rectangle and cursor warp. They reported that by manipulating the two corners of a rectangle, multiple icons can be selected quickly. To use the cursor warp technique, the user must first click an icon using one mouse and then click the position where the user wants to move the icon using the other mouse. This reduces the movement distance of the mouse.

This input system does not achieve completely concurrent manipulation. For example, the user cannot move two sliders simultaneously. Moreover, it allows concurrent manipulation of only two components. In addition, because the mice provide relative distances, the system has a problem with disagreement of the individual coordinate systems. (See Section 6.1.2.)

5.1.4 Digital Tape Drawing

Digital Tape Drawing[1] implements a *tape drawing* technique that is used in the field of industrial design on a computer system. The tape drawing technique is a drawing method of a curve with a colored adhesive tape on a canvas. The user first sticks one end of the tape to the canvas and then unrolls the tape with his right hand (dominant hand). The user then sticks the tape to the canvas by sliding his left hand along it. The right hand (dominant hand) is used to adjust the curvature of the curve.

Digital tape drawing uses two 3D mice and a rear-projection screen. The user stands in front of the screen and holds the mice with both hands. The start point of a curve is determined by clicking a button on the right mouse. When the user moves the right mouse, the system displays a guide line between the start point and the position of the right mouse. When the user moves the left mouse, the pointer moves along the guide line, and the user can draw the curve with the pointer by pressing a button on the left mouse. This enables the user to draw a smooth curve by manipulating both mice simultaneously.

Since the 3D mice have dedicated features, this system is categorized as a specific device-based system. However, it may be easy to apply this drawing technique to a generic device-based system.

5.1.5 Laser Spot Tracking

There are several laser pointer tracking systems that recognize the position of a laser spot on a screen, and some of these systems can track multiple laser points. LumiPoint[7] is a pointing system used for collaboration work on a large display. A number of users share the display and can point to the display with multiple laser pointers. However, this system assumes that each user has only one laser pointer, which is different from the goal of the present study. Oh and Stuerzlinger[39] reported a multiple laser pointer tracking system that distinguishes the laser pointers by having the lasers blink in different patterns. This system also assumes that each user has one laser pointer. Neither system describes multipoint input by a single user.

In this research, a multipoint input system with laser pointers is described in Chapter 9.

5.1.6 Phidgets

Phidgets[20] provides a set of building blocks for sensing and control devices, which can be connected to a computer to build an interface that the user can touch and manipulate directly. The system consists of electronic devices such as buttons or volume sliders, and a central board that connects these devices and a computer via USB. The system also provides a number of Visual Basic plugins for writing software to reflect the input from the physical interface by Phidgets.

In general, it is difficult for the user to build a specialized input device. Practically, it is impossible for the software developer to construct a specific device by himself. However, by using Phidgets, it is easy to build custom devices by assembling the building blocks of Phidgets.

However, the assembled device is specialized for its target application, and the device cannot be used for other applications. Therefore, this device is dedicated to a specific application. In other words, it is a specific device. It is impractical to rebuild the device every time the application is switched. Thus, Phidgets are not suited to building generic input devices, which is the goal of the present study.

5.1.7 Smart Toy

Zowie Intertainment developed a position sensing technique based on RFID technology [46]. This sensor can recognize multiple devices and their positions simultaneously. Each device has an LC circuit that has a unique resonance frequency. The devices are placed on an input surface that has a grid-shaped antenna. The system transmits wave signals in time-division multiplexed frequencies. If a device is present on the antenna that has a corresponding resonance frequency, then the system detects the device. The system quickly switches the excited electrodes, so that the position of the device can be detected.

The number of the devices depends on the scanning range of the frequency and its accuracy. The latency is increased when the range is widened.

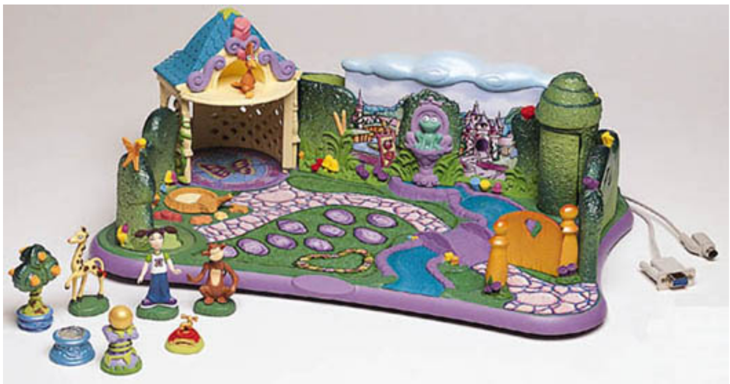


Figure 5.3: Smart Toy (Ellie's Enchanted Garden)

Quoted from Web pages of E-M Designs, inc. (`http://www.emdesigns.com/portfolio/clientlist/zowie/dtl_ellie.html`)

5.1.8 Tangible User Interface

MIT Tangible Media Group introduced various systems, called **Tangible User Interface**, that users can touch (tangible) and use to directly manipulate a system. A number of these systems that allow multipoint input are adopted herein.

metaDESK

metaDESK[58] is a tabletop interaction system with Phicon. An example application is a map browsing system. metaDESK provides two Phicons, which represent two buildings in the area, and the map displayed on the table reflects the positions of the Phicons.

This system provides specific input devices. The use of generic devices with metaDESK was discussed in the paper that introduced the metaDESK system, but this concept is not implemented herein.

Illuminating Light

Illuminating Light[59] is an optical simulation system for rapid prototyping manipulated on a tabletop. It provides specific devices, which represent a laser, a mirror, a lens or a beam splitter. The system recognizes the positions and orientations of the devices on the table by detecting IDs on the devices from a video camera above the table. Since the devices are tracked in real time, users can manipulate them concurrently. The

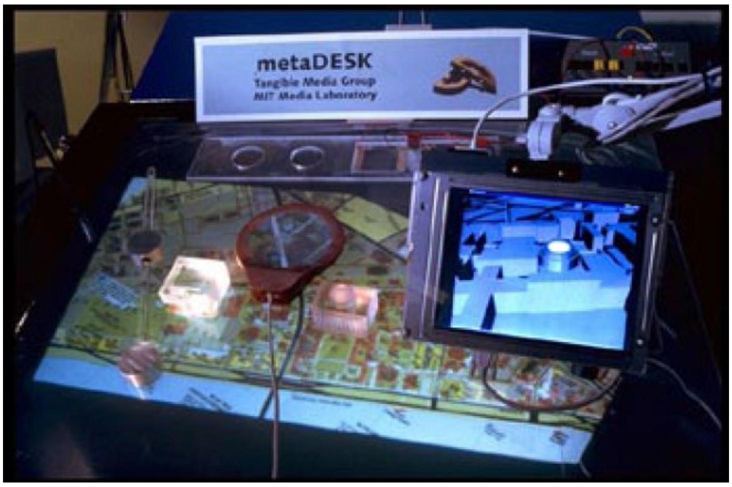


Figure 5.4: metaDESK

Quoted from [58].

system calculates the optical simulation and displays the results on the table top in real time, therefore, when the user moves a device, the result is instantaneous.

The devices and the interface are specialized for the application. The use of generic devices on this system has not been proposed. The initial paper describes the efficiency of tangibility and the intuitiveness of the interface. The authors reported that concurrent manipulation was useful for collaborative work by multiple users.

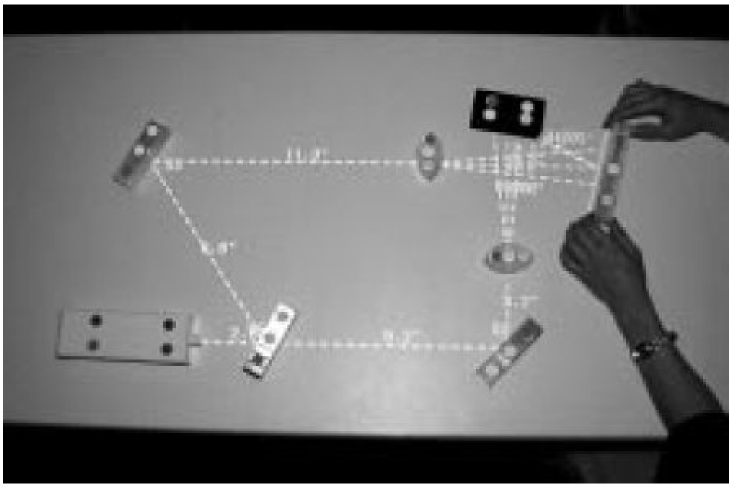


Figure 5.5: Illuminating Light

Quoted from [59].

Urp

Urp[60] extends Illuminating Light to lighting simulation. The system provides Phicons of buildings. When the Phicon is placed on the table, the system calculates and displays the shadow of the building. The system also provides a device to change the material of the building. By touching a building with the device, the user can change the wall material.

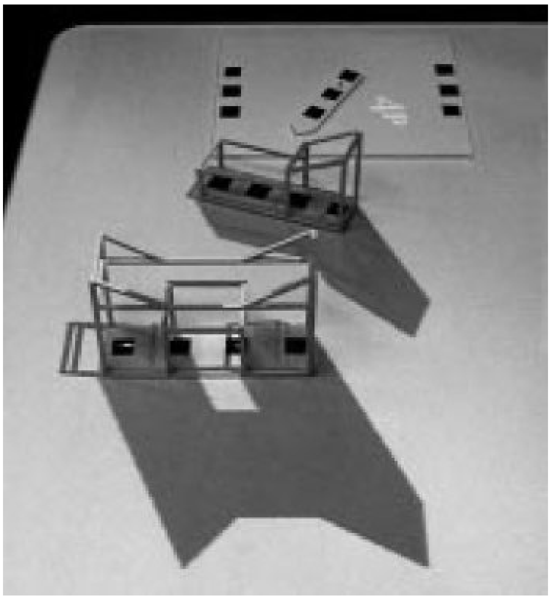


Figure 5.6: Urp

Quoted from [60].

Sensetable

Patten et al. introduced a generic device-based multipoint input system using two Wacom tablets[44]. The system provides mouse-sized multiple input devices, as shown in Figure 5.7. Each device has a unique device ID. Each tablet can recognize no more than two device IDs, but each device switches the ID on and off randomly, so that the system can detect all IDs. This causes a tracking latency of less than one second. In addition, each device has a dial on the top surface that modifies its device ID, so that the system recognizes not only the position but also the value of the dial.

In this implementation, the latency is increased if two or more devices are used simultaneously. But Patten et al. reported that because each device is mouse-sized, users did not typically move more than two devices at a time. However, this can be a problem when multiple users use the system.

Because this system is a generic device-based system, attachment and detachment of devices to components in the screen. On Sensetable, a component is attached when a device is moved in the component. When there are many components on the table, however, it becomes difficult to select one from among them. To avoid accidental selection, Patten et al. introduced two methods: dynamic adjustment of the spacing of components near a device that has no attached components. In addition, it is required to put a device on a component for a while to attached it. To detach the component, the user shakes the device.

This system is similar to the generic device-based multipoint input system described in Chapter 7. The proposed system achieved low latency and allows the user to manipulate more than two devices concurrently. On the other hand, each device provides only positional information and has no additional interactive parts.

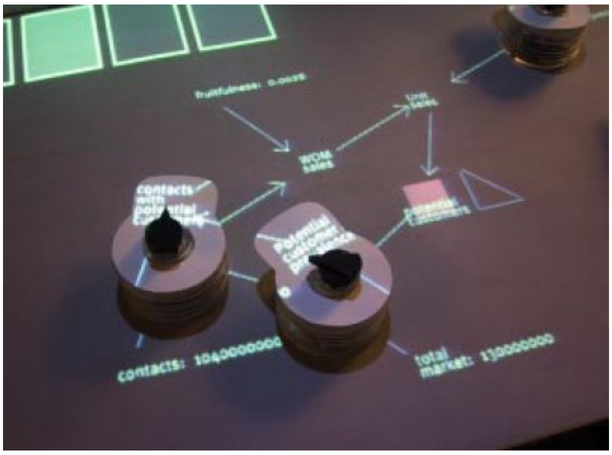


Figure 5.7: Sensetable

Quoted from [44].

Audiopad

Audiopad[45] is an interface for musical performance. As shown in Figure 5.8, input devices are manipulated on an input surface that overlays a computer screen. The devices contain RFID tags, and their positions are detected by a time-division scanning technique, as described in Section 5.1.7). Each device has two RFID tags so that the system can recognize its position and orientation. The scan rate is not described explicitly in the present paper but appears to be approximately 10 scans per second.

By attaching a sound track to a device and moving the device toward the center of the screen, the volume of the sound track increases, and vice versa. This concept is identical to that of MidiSpace, which is introduced in Section 5.4.3, but this system allows direct concurrent manipulation of the volumes. However, this can also be achieved using a common audio mixing console. In fact, the mixing console allows more precise and higher concurrent manipulation. In contrast, Audiopad provides an intuitive visual representation of the relationships between the sound tracks and the input devices and their direct manipulation. In addition, Audiopad allows more complicated manipulation such as the switching of sound tracks.

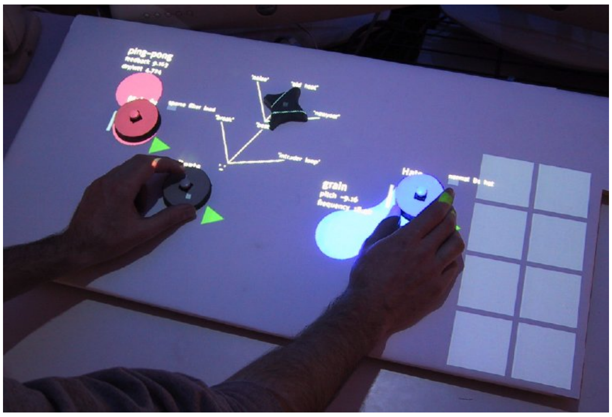


Figure 5.8: Audiopad

Quoted from [45].

Actuated Workbench

In most multipoint device-based input systems, the positions of devices are transmitted to a computer and are used to manipulate corresponding components. Even if the computer changes the position of the components, since the corresponding device cannot be physically moved by the computer, the component will be moved back to its original position. If the computer detaches the binding automatically to avoid this problem, the user must move the device and reattach the component.

In many cases, an application provides various forms of automatic support to the user. For example, save and restore current status and undo operations are fundamental services of computer applications. In order to implement these services, the abovementioned problem must be solved. Thus, the computer should move the input devices. In fact, a number of commercial high-end audio mixing consoles can move sliders and knobs automatically.

Actuated Workbench[43] extends a tangible multipoint input system to solve this problem. This system employs an array of electromagnets to move input devices on its top surface.

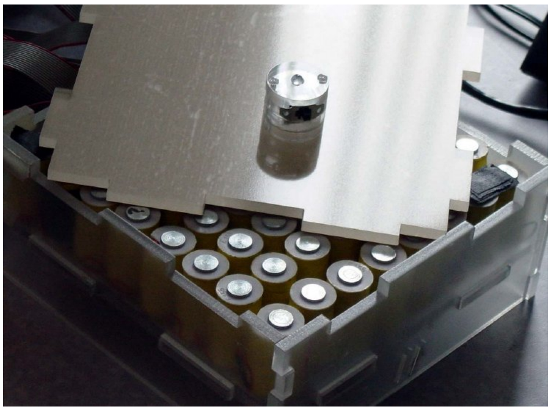


Figure 5.9: Actuated Workbench

Quoted from [43].

5.2 Multipoint input systems without physical devices

5.2.1 Enhanced Desk

Enhanced Desk[34][40] is a tabletop interaction system similar to metaDESK, but Enhanced Desk recognizes the positions of the fingers by a camera to provide multipoint input. An image of the hands is taken from above the desk, and the vertical motion of the fingers cannot be detected. Therefore, Enhanced Desk does not recognizes whether a finger is touching the surface, but the system can recognize changes in the number of fingers, as well as gestures by the fingers.

Since it does not recognize touching motion, in order to recognize manipulations such as ‘clicking’, the system employs bending or pinching gestures, but these are not intuitive. The SmartSkin prototype system (Chapter 8) does not require an external camera and has no occlusion problem. In addition, since this system can recognize the touch of a finger, it does not require any special gestures.

5.2.2 HoloWall

Matsushita et al. developed HoloWall[32], an interaction system with an infra-red camera. The system consists of a rear-projection screen and an infra-red camera that is placed behind the screen. The user stands in front of the screen and touches it with his hands or arms. Behind the screen, an infra-red light is irradiated and an object on the screen reflects the light, the infra-red camera then detects the light. The computer display is projected onto the screen.

Matsushita et al. also described a two-handed interaction on HoloWall that implemented a multipoint input (two points can be input). In addition, they introduced

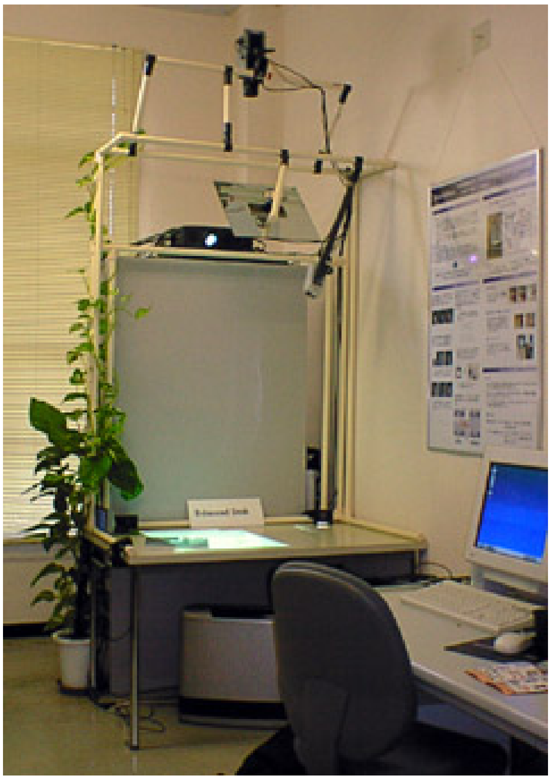


Figure 5.10: Enhanced Desk

shape-based input using the arms.

5.2.3 Dual Touch

Dual Touch[31] is a multipoint input for the touch screen of a PDA. Basically, Dual Touch assumes an input by the fingers or a pair of styluses. Although the system accepts dual pointing, when the second point is touched, the first touched point must not be moved. This means that concurrent manipulation is not possible. A sample interaction technique based on Dual Touch, called Tap Step Menu, has also been introduced. The first touch is used to select a menu from a menu bar, and then a list of menu items is displayed. The second touch is used to select an item from the menu list. This menu selection technique can be implemented on the proposed multipoint input system.

5.2.4 DiamondTouch

DiamondTouch[8] is a multi-user interaction system that accepts concurrent two-handed input. The significant feature of DiamondTouch is that the system can identify who is touching the input surface.

The system consists of a table covered by a DiamondTouch sheet, and a number of chairs around the table. The input surface is a grid-shaped receiver antenna, and each chair has a transmitter that transmits a wave signal with a unique frequency. When a user sitting in a chair touches the input surface with his finger, a wave signal from the transmitter of the chair is received by the antenna via his body. The receiver antenna consists of vertical and horizontal electrodes, and the system senses the power of the signal for each electrode and performed peak detection. Thus, when two fingers are touching the surface, a rectangle-shaped integral closure of the fingers is recognized. This gesture can be used for selection.

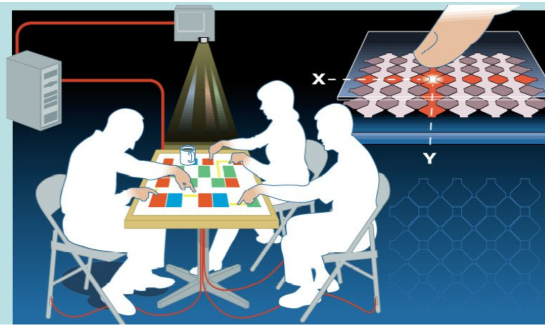


Figure 5.11: Diamond Touch

This image is taken from Diamond Touch product specifications sheet.

5.2.5 Fingerworks

iGesture Pad[25] from Fingerworks is a tablet-shaped generic pointing input device. iGesture Pad recognizes the positions and shapes of the fingers or hands on the sensor board. Through a bundled device driver, this device can be used as a standard single pointing device, such as a mouse. Multipoint input is not allowed, but when a second touch is detected it is handled as a button click, and the third touch is handled as a double click.

5.2.6 TactaPad

TactaPad[55] by Tactiva is a tablet-shaped generic multipoint input device that also has a tactile feedback mechanism. TactaPad allows indirect multiple pointing gesture by the fingers. In an indirect finger pointing gesture, the position of the finger is not recognized until the finger touches the input surface, which means that the user does not know where he is going to point. TactaPad has a video camera mounted above the input surface, as shown in Figure 5.12, to capture images of hands on the surface and the image is overlaid onto the computer display so that the user can recognize the position that he is touching.

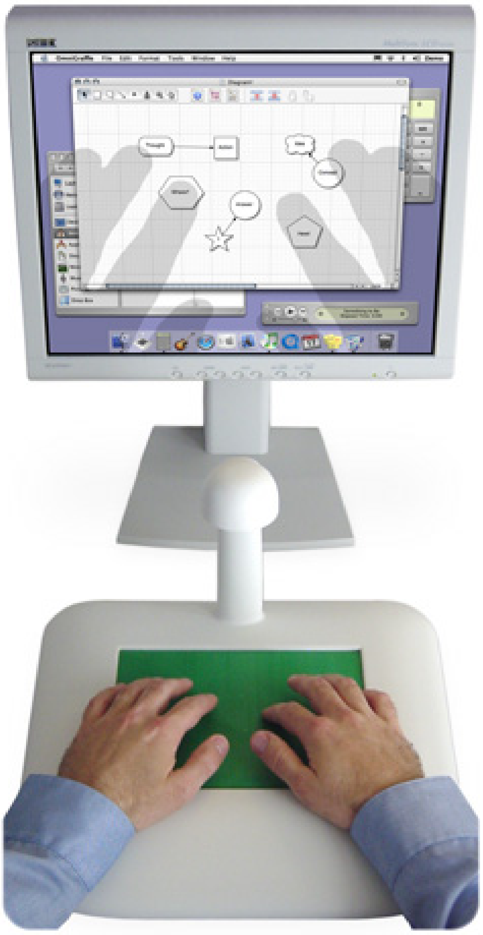


Figure 5.12: TactaPad

This image is taken from Tactiva's web site.

5.3 Two-handed interface

5.3.1 A study in two-handed input

Buxton and Myers demonstrated the efficiency of a two-handed input in a compound task[5]. By their system for the evaluation test, the subject manipulated a usual pointing device by his preferred hand and a slider by his nonpreferred hand. One of the evaluation tests involved a compound task of positioning and scaling a component. A rectangle on a screen could be moved by the pointing device and scaled by the slider.

The goal of the task was to fit the rectangle to a designated target.

In the evaluation, the interface was built with generic input devices, however, since there was only one component that could be moved, there was no need for attaching or detaching, so that the subject did not need to maintain the relationship between the device and the component.

They reported that subjects could quickly perform concurrent manipulation, indicating that concurrent manipulation is natural for humans. Currently, a mouse with a scroll wheel can be considered to be analogous to the pair of a pointing device and a slider.

5.3.2 Passive Real-World Interface Props

Hinckley et al. introduced a two-handed interaction system with two 3D mice[22]. The system was applied to a neurosurgical props interface (Figure 5.13). The nonpreferred hand oriented a doll's head, and the preferred hand oriented a cross-sectioning plane. Both devices were specific devices dedicated to the application. Therefore, two 3D mice can be used to efficiently control two virtual objects. This system is not extended to concurrent manipulation of more than two objects.

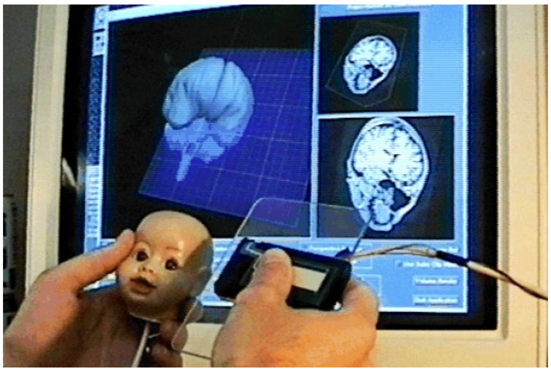


Figure 5.13: Props interface for neurosurgical visualization

This image is taken from [23].

5.3.3 Bimanual gesture input

Gesture input is often employed in pen input devices. Typically, a gesture recognition system tracks the motion of a pointing device and identifies a gesture from among a number of registered gestures.

On a multipoint input system, gesture input could be extended to multi-motion. Some gesture recognition techniques for sign languages recognize bimanual gestures [52][62][4][64][63]. They detect two-handed motion from video images. Since typical multipoint input devices do not have additional buttons or a scroll wheel, gesture input will help users to input varied operation.

Brand, Oliver and Pentland introduced the Coupled Hidden Markov Mode (Coupled HMMs) for bimanual gesture recognition[4]. The HMM is useful for recognizing a single-motion gesture. However, when two HMMs are paired with track bimanual motion, the symbol space becomes too large and it is not efficient. Linked HMMs reduce the symbol space, but the relationship is non-causal. Coupled HMMs can compound two HMMs with a causal relationship. For example, a non-causal model cannot represent the fact that the right hand is lifted *then* the left hand is lowered, while a causal model can.

Wilson and Bobick introduced Parametric HMM[64], which can represent a parameterized gesture. The usual HMM cannot distinguish, for example, between motions of hands that are separated by distances of 20 cm and 40 cm. In contrast, parametric HMM represents the quantity of the motion by adding quantifying parameters. Nonlinear Parametric HMM[63] can represent not only the distances but also the speeds of motions.

5.4 Concurrent manipulation by conventional input device

A conventional input device can attach only one component basically. When a user wants to manipulate multiple components concurrently, usually it is required to serialize the operation and manipulate every component sequentially but it is inefficient when there are many components which user wants to manipulate. Therefore, many interaction systems provide some methods for concurrent manipulation by a single pointing device.

5.4.1 Grouping

The grouping operation is a method to manipulate multiple components as one component. An operation performed of one component of the group is propagated to all components of the group. Therefore, if the user wants to transmit the same request to all of the components, grouping is effective, but the components cannot be manipulated independently. In addition, the grouping operation requires a selecting manipulation in order to make a group.

Rectangle selection

The rectangle selection tool is commonly used in most desktop GUI systems to group icons on the desktop and manipulate them simultaneously. The procedure of rectangle selection operation is as follows. A pointer is moved to a corner of the rectangular region that the user wants to select. A button on the mouse is then pressed, and the pointer is moved to the opposite corner. The button is released, and the icons inside the rectangle become ‘selected’. After the selection operation, the user can move the selected icons as a group or can open a context menu and select an operation.

To use this selection tool, the icons must be aligned in a rectangular area. If they are not, they need to be align before selection, otherwise a number of rectangular icons should first be selected, and then the rest should be added.

Don’t Click, Paint!

Baudisch introduced an effective selection tool for a console that has an array of toggle buttons often seen in a configuration panel[3]. When there are several toggle buttons in the panel, it takes a long time to set the buttons sequentially using a pointing device. This selection tool works effectively if part of the buttons can be set or unset simultaneously by allowing grouping operation. The selection tool allows the user to set/unset toggle buttons by the rectangle selection technique described above. This is similar to rectangle painting in a bitmap using drawing tools. Baudisch called this operation Toggle Maps.

This grouping technique requires a selection step, and can therefore be categorized as time-multiplexing.

5.4.2 Macro control

In order to send requests to components simultaneously by an action, *macro control* is useful. Before sending requests, a user registers a request for each component. The system provides an additional component that sends the registered requests simultaneously when it is manipulated. This process is called macro control.

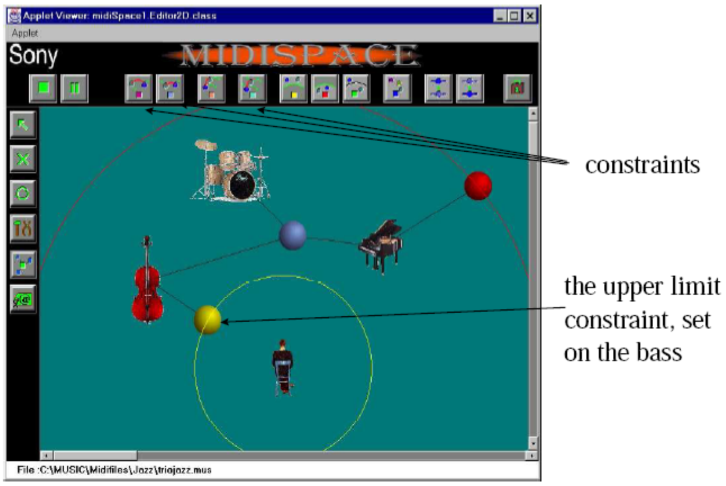
5.4.3 Constraint solver

By adding some constraints between components and applying a *constraint solver* to the constraint system, a GUI system can allow the user to manipulate multiple components simultaneously by manipulating only one of the components. Basically, constraints are set on the positional relationships between components and the user maintains the positions of the components.

MidiSpace

MidiSpace[42] is an audio mixing application that uses a constraint solver to control the volumes of multiple audio inputs. MidiSpace displays a virtual 2D space on the screen. (See Figure 5.14.) There is a virtual audience at the center of the space, and the user can manipulate icons that represent the sound tracks for individual musical instruments. By moving these icons, the user can control the volumes and localization of the instruments. For example, when an icon is positioned to the right, its sound comes from the right. When it approaches the audience, its volume increases. In addition, using various constraints, multiple parameters of the sound can be controlled simultaneously. For example, when a user wants to increase the volumes of both the drums and the bass, he can do this by placing a constraint between them and moving only one of them. MidiSpace provides various constraints, such as symmetrical or synchronous manipulations, which can be used hierarchically, so that most typical audio mixing operations can be performed.

This system requires a pre-process of installing the constraints among the components before concurrent manipulation. Therefore, this system provides a time-multiplex interface.


Applet Viewer: midiSpace1.Editor2D.class
Applet
Sony
MIDISPACE
File :C:\MUSIC\Midifiles\Jazz\triojazz.mus

constraints

the upper limit constraint, set on the bass

Figure 5.14: MidiSpace

This image is taken from [42].

5.5 Conclusion

In this chapter, previous research and systems using concurrent manipulation were reviewed. The systems were characterized by the taxonomy introduced in Chapter 4.

In device-based-input systems, the size and restriction of manipulation was found to be important in concurrent manipulation. In addition, the difference between systems with specific devices and generic devices was revealed. In systems with specific devices, the specific devices are designed so as to externalize certain application components of the computer systems, whereas systems with generic devices maintain application components internally and the generic devices are used only to manipulate these components.

Non-device-based systems were surveyed and the number of inputs was found to be limited in many systems. On the other hand, a number of systems limit the motion of the fingers. These systems are not satisfactory because they are not sufficient for recognizing highly concurrent finger gestures.

Other systems used for concurrent manipulation of multiple components are designed for systems with one pointing device, which indicates that the need for concurrent manipulation may be high in various applications. In addition, a number of previous studies on two-handed input reported that users can naturally manipulate a computer application using both hands. In the following chapter, the design of effective concurrent manipulation is described based on the above review.

Chapter 6

Analysis and Design of Concurrent Manipulation

In this chapter, the design of concurrent manipulation is described based on the knowledge obtained in previous studies described in Chapters 2, 3 and 5.

6.1 Multipoint input

6.1.1 Concurrent manipulation by multipoint input

Multipoint input allows pointing to multiple positions on a computer screen. By using this positional input, it is possible to point to multiple components and move them concurrently.

In a device-based input system, input devices that can be manipulated by fingers are provided. In contrast, in an input system without physical devices, the user touches the screen directly with his fingers and moves his fingers for concurrent manipulation.

In the following sections, the requirements of concurrent manipulation for device-based and non-device-based input system are described the design of the systems is discussed.

6.1.2 Requirements for device-based input system

On a device-based input system, the absolute position of a device in the coordinate system of the input surface is used for positional input. However, in an indirect pointing system, the distance on the input surface could be scaled for the coordinate system of the screen.

Keeping this in mind, the following requirements are introduced:

1. The system can measure absolute positions

2. The system can recognize as many input devices as possible
3. The input devices are sufficiently small
4. The input devices can be moved freely
5. The system can track the devices quickly

The system can measure absolute positions

Next, the reason for needing the absolute position recognition is described. At this time, most pointing devices detect the relative distance because the device can be manufactured simply and inexpensively. In addition, this enables moving the pointer over a wide area with a narrow input surface by moving the mouse repeatedly, so that the space required for mouse operation and hand motion are reduced. In addition, this enables the speed of the pointer to be changed along with the motion of the mouse. This is also effective for saving space.

However, implementing a multipoint input system causes some problems when the system uses multiple relative distance input devices simultaneously. First, the relative distance between two devices may not be replicated on the screen as they are. Figure 6.1 illustrates this problem. In the figure, the user manipulates two input devices to approximate the components, but the devices conflict with each other before approximation. When the number of devices increases, additional conflicts will occur. The second problem is the inconsistency of the axes of the devices. If two device are moved in parallel, the corresponding pointers are expected to move in parallel as well. However, each relative distance input device has its own coordinate system, so that the axes of the devices are not guaranteed to be parallel (Figure 6.2).

For these reasons, absolute distance input devices must be used to construct a device-based multipoint input system.

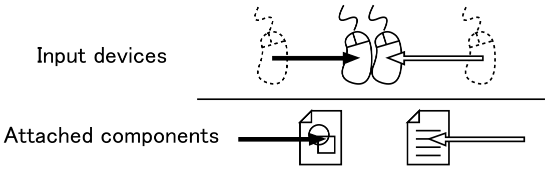


Figure 6.1: Problem of multiple relative distance inputs (1)

The system can recognize as many input devices as possible

The system must be able to use multiple input devices simultaneously to construct a multipoint input system. Because the number of devices limits the number of compo-

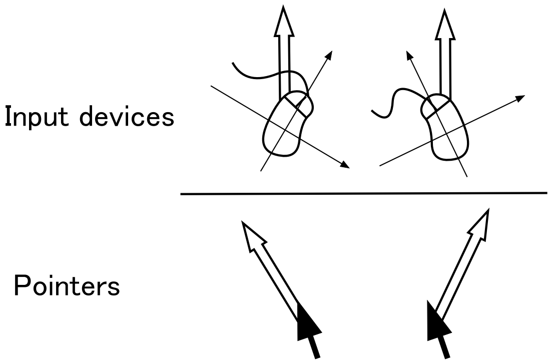


Figure 6.2: Problem of multiple relative distance inputs (2)

nents of concurrent manipulation, the number of devices should be as high as possible. However, the system must allow the user to use any number of devices.

The input devices are sufficiently small

Because multiple devices are manipulated in a certain area, devices conflicts are unavoidable. To reduce the possibility of conflicts, each device should be small. If a device is sufficiently small and light, it can be moved on a fingertip or held in one hand. However, if the device is too light, the device may be moved accidentally. Proper size and weight are still required for stable manipulation.

The input devices can be moved freely

For generic use, each device should be able to be moved in any direction. A wired device or restricted motion is insufficient.

The system can track the devices quickly

The scanning speed influences the scan rate and latency of response. A low scan rate decreases the smoothness of input, and a long latency results in an uncomfortable feel to users. Therefore, the scan speed should be as high as possible.

6.1.3 Requirements for non-device-based input system

Next, the requirements for a multipoint input system without physical devices are discussed. In a non-device-based input system, the position of a fingertip on the input surface is used as a positional input. Here, the absolute position in the coordinate system of the input surface is used. The coordinates can be scaled to fit to the screen.

The requirements are as follows:

1. The system can detect multiple fingertips
2. The system can recognize the absolute position of the fingertips
3. The system can distinguish whether a finger is touching the input surface
4. The system can track the fingertips quickly

The system can detect multiple fingertips

The system must track multiple fingertips to implement a multipoint input system. In addition, it is expected that all of the fingers of the hands are detected in a similar fashion.

The system can recognize the absolute positions of the fingertips

The reason for this requirement is described in Section 6.1.2. The system will fail to provide an intuitive interaction when the relative distance between two fingers is not replicated on the screen.

The system can distinguish whether a finger is touching the input surface

This is the most intuitive manner of using touching motion for attaching a component to a finger. The reasons are as follows:

- This type of system is seen in daily life
- The user can receive tactile feedback from the surface on his finger
- Because the finger is moved vertically, its horizontal position is not influenced

The system can track the fingertips quickly

See Section 6.1.2.

6.1.4 Design of multipoint input system

Next, based on this discussion, the design of a multipoint input system is demonstrated and its characteristics are discussed.

Specific-device-based – direct pointing

This input system recognizes specific devices on its screen. An application on the system provides a complete set of specific devices of corresponding internal states. Because the devices are placed on the screen directly, the system can display information of the component or the results of the user's manipulation around the device, and the system helps the user maintain the relationships between the components and the devices. However, because a device occludes the screen, the device must be moved in order to display information.

This causes a problem when the application moves the components. In order to restore the positions of components or assist the manipulation by the user, the application must change the positions of the components. On a specific-device-based input system, it is necessary to move the corresponding devices automatically. However, it is generally very difficult to move hardware devices that are non-wired and placed freely. The addition of mechanisms to the device or the use of a machine to move them is too expensive [1]. In contrast, although the relocation of the devices can be performed by the user, this takes a long time and is boring to the user.

This kind of system is not suitable for a generic input system.

In previous studies, metaDESK and most tangible user interfaces employ this form.

Specific-device-based – indirect pointing

This system is similar to the previous system, but provides an input surface and a screen separately. The system cannot display information of a component around a corresponding input device, but the system does not have occlusion problems. In addition, this system can set a display freely, while a direct pointing system limits the size and angle of its display.

This type of system is not suitable for a generic input system either.

Sensetable and Audiopad fall into this category.

Generic-device-based – direct pointing

Multiple generic devices are placed and manipulated on a display. Since generic devices can be used for any applications, the system provides 10 – 20 devices at most.

Basically, generic-device-based concurrent manipulation has a problem in that it is difficult for the user to maintain the relationships between the components and the devices. However, by displaying information around the devices, the system resolves this problem. Because the system can provide both devices and their information in the same view, it is unnecessary for the user to move his eyes between the display and devices.

The problem of the specific-device-based system, i.e., the difficulty of manipulation assistance by an application, occurs in this type of system as well, but because the relationships between the components and the devices are flexible, this problem can be

[1] Pangaro et al. developed a system that moves devices by an array of electromagnets, as shown in Section 5.1.8. In addition, a number of chess computers move chess pieces using magnets, and certain audio mixing consoles move sliders and knobs using built-in actuators.

avoided. At the beginning of the assistance, the application detaches from the devices components that are to be replaced, and then moves these components. When the assistance is finished, the user reattaches these components to the devices. In a specific-device-based system, the user has to replace all of the devices as they are designated by the application, but this system has a lower cost of motion of the devices.

Sensetable falls into this category.

Generic-device-based – indirect pointing

This system is similar to the previous system, but the input surface and the display are separate. Most desktop systems employ this form of single-point input device. The device does not occlude the display. The display can be placed vertically.

On the other hand, the user must look at the input surface to maintain the devices. This causes the user to move his eyes between the display and the input surface repeatedly. If there is only one pointing device on the input surface, this is unnecessary, because the user can confirm the position of the device by keeping his hand on it. However, it is difficult to do this with multiple devices.

Fitzmaurice et al. conducted a number of evaluations on this type of system. (See Section 5.1.2.)

Non-device-based – direct pointing

In this system, the user points to positions on a display using his fingers to directly input multiple positional data. By detecting touches on the display, the user virtually manipulates the components using his fingers directly. In this type of touch panel system, a component is attached to a finger when it touches the display and is detached when the finger is removed from the display. These binding operations provide an intuitive direct manipulation.

When the user does not touch the display, the application can move the components without any of the binding problems that occur in device-based systems. When the application has to move a component attached to a finger, the application should detach the component or display a message asking the user to release the component.

Diamond Touch and Enhanced Desk fall into this category.

Non-device-based – indirect pointing

This system displays pointers (mouse cursors) to indicate the positions touched by the user on the input surface. There is a problem when the input surface detects the touching of fingers. Since these systems cannot track the fingers when they do not touch the surface, the user has no knowledge of where he will point to without seeing the input surface and his finger, and this decreases effectiveness of concurrent manipulation. Vision-based finger tracking systems avoid this problem, but such systems cannot detect the touch of a finger. Therefore, an intuitive attaching/detaching operation cannot be obtained. To solve this problem for a device-based system, Tactiva proposed the combination of a touch sensor and a video camera. (See Section 5.2.6.)

iGesture Pad and TactaPad fall into this category.

6.2 Non pointing input

As a concurrent manipulation technique without pointing input systems, a technique based upon a shape input device is proposed.

Fitzmaurice et al. reported that bulldozer operation was observed during an experiment, in which a pile of LEGO blocks was given to a subject and the subject was asked to divide them based on color[13]. This type of operation can be performed using a ruler or a rake. Similarly, rather than manipulating each component by a pointing device, several components can be manipulated simultaneously by utilizing virtual conflicts between objects (e.g., hands, rulers, etc.) and components.

The following two methods of implementation of this concurrent manipulation technique are proposed.

6.2.1 Bulldozer manipulation with hands

This technique requires a device to sense the shape of the human hands. The system detects the user's hands on the input surface and moves the components based on the motion of the hands, for example, by using a collision between a hand and a component. By this technique, the user can move several components simultaneously with his hands or arms.

6.2.2 Bulldozer manipulation with a curve input device

This technique uses a curve input device and a positional sensor. A curve input device consists of a flexible strip that is sensitive to bending and twisting. The system tracks the position, direction and shape of the device and calculates collisions between the device and components on the screen. Measurand Inc.'s ShapeTape[33][2] device, which consists a $48 \times 1 \times 0.1$ cm rubber tape for which bending and twisting are measured at 6-cm intervals by two fiber optic bend sensors at 30 Hz, can be used for this type of manipulation. In addition, spatial sensors are used to track the position and direction of the device.

6.3 Discussion on ergonomics

In this section, a number of ergonomic issues of concurrent manipulation are discussed.

Woods et al. reported the results of ergonomics evaluation tests for various non-keyboard input interfaces of workstations[65]. In this report, they discussed ergonomics issues of touch screen interface. In addition, Nielsen et al. described a design guide of intuitive ergonomic gesture interfaces[38]. They ran an evaluation test and reported ergonomics issues of hands-free gesture input. The concurrent manipulation interface is considered to have the same issues. Therefore, the ergonomic issues of concurrent manipulation are discussed based on this report.

6.3.1 Restriction of posture

The structure of the hand restricts the motion of the fingers, and this limits the effectiveness of concurrent manipulation with fingers. In particular, because the fingers must touch an input surface to point to components on a screen, the posture of the fingers is heavily restricted. In contrast, a physical component of an input device can be manipulated with various postures. For example, a slider knob can be manipulated by pushing, pulling, pinching, etc. Concurrent manipulation interface without physical devices should take this problem into consideration.

6.3.2 Stress from extension force

In order to point to multiple positions on the input surface with the fingers, it is usually necessary to extend the fingers, which is known as a stressing gesture[38]. In addition, in some cases, the user must spread his fingers or posture his fingers unnaturally, which causes damage to the user's hand.

On the other hand, because concurrent manipulation can reduce the time of manipulation of the application, the total damage to the hand may be decreased.

6.3.3 Undue force to manipulate

Woods et al. reported that all subjects mentioned that they sometimes needed to exert undue pressure to push buttons on a touch screen[65]. The reason for this is that the sensitivity of the touch screen was low, and so the screen failed to sense light touches. This problem will likely become significant for a multipoint input, because some fingers of the hand may be pressed to the input surface softly during multiple-finger operation. Pressing all fingers against the screen could be stressful.

6.4 Conclusion

The design of input systems that enable concurrent manipulation of multiple components was described. In addition, the requirements of device-based and non-device-based input systems were described, and the design basis of possible forms of concurrent manipulation system was discussed.

Based on this design, three prototypes that allow concurrent manipulation of components were proposed. The first prototype is based on a generic-device-based – indirect pointing system, and the development of a generic input system that is similar to the existing mouse-based input system was considered. The second and third prototypes are based on a non-device-based – direct pointing system, and the development of a more intuitive form of direct pointing input was considered.

Moreover, a non-pointing input system was implement using a human body sensing device, which allows the user to manipulate multiple components by his arms and hands. This input system was integrated into the multipoint input system, and a flexible input system was proposed for concurrent manipulation.

Chapter 7

Prototype 1: Multipoint Input System using Physical Devices

7.1 Overview

In order to evaluate a generic multipoint input system, the requirements of a multipoint input system are listed and existing input devices are examined. However, all of these systems were judged unsuitable. Therefore, multipoint input systems were designed and built. The first prototype employs physical devices so that it is easy to develop. This prototype can be used not only as a generic multipoint input system, but also as a specific multipoint input system so that two styles of multipoint input could be evaluated.

As the result of an evaluation test, concurrent manipulation with more than two devices was found to be less effective. In addition, the difference between the results for specific devices and generic devices was small.

7.2 Related Works

In Chapter 6, several requirements of a multipoint input system with physical devices were presents. Several input devices were examined with respect to these requirements, and no suitable device was found. The results are presented below.

Mouse

A mouse measures only relative displacement. Moreover, even a small mouse is 5 – 6 cm^2 or larger, which is too large to move by a fingertip. Moreover, because of their size mice will collide in many cases when multiple mice are used simultaneously. This is not sufficient for concurrent manipulation.

Tablet

A tablet satisfies most of the requirements, but most tablets cannot handle multiple pointing devices. Wacom's tablet can recognize only two input devices simultaneously. Fitzmaurice constructed an input surface that allows concurrent manipulation of eight devices by concatenating four tablets[11]. However, since the number of devices that can be used on a tablet does not change, each device can be used only in a restricted area on the surface.

Polhemus sensor

Polhemus sensor is a three-dimensional position tracking sensor that uses electromagnetic technology[47]. It provides a pair of wired input devices, which are suitable for the present purpose.

2D Barcode

2D barcodes, such as Matrix[50], AR Toolkit[27] or QR code, can be used as a multipoint input system by measuring their positions using a video camera. This satisfies most of the abovementioned requirements, but the size of a barcode is restrictive. For example, a Matrix code must be larger than 4 cm×4 cm. In addition, since the scan rate decreases when two or more markers are recognized, the barcode cannot be used for real-time applications.

Pressure sensor/Capacitive sensor

Some interactive surfaces based on pressure or capacitive sensing techniques allow multi-touch interaction. These systems cannot be used to sense physical device. However, a prototype system based on a capacitive sensor is introduced in Chapter 8.

7.3 Implementation

As described in the previous section, no suitable input devices were found for the evaluation of multipoint input. Therefore, a prototype implementation of a multipoint input using physical devices was developed.

7.3.1 Basic design

The user manipulates multiple blocks on a table top. The system identifies and tracks the blocks by a video camera. Each block is small enough that the user can move them by his fingers.

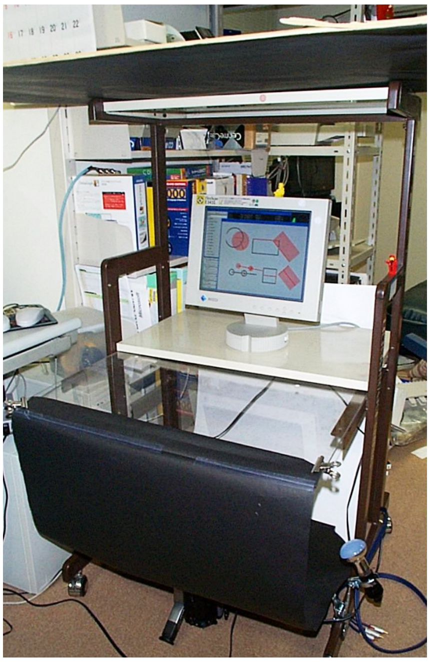


Figure 7.1: Prototype 1: system overview

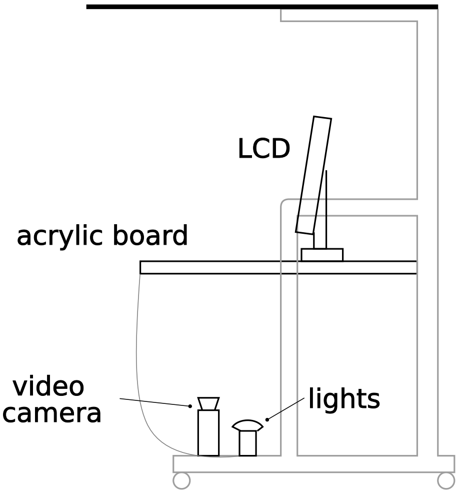


Figure 7.2: Prototype 1: system diagram

7.3.2 Hardware

Figure 7.1 shows the prototype constructed herein, and Figure 7.2 shows a schematic diagram of the system. The system is based on a standard desk, the top surface of which is replaced by a 5 mm thick clear acrylic board. The devices are manipulated on the acrylic board, and their bottom surfaces are captured by a video camera placed below the board. The positions of the devices are detected from the captured images. A number of lights are placed below the board to light the blocks and improve color accuracy. Figure 7.3 shows a user manipulating the system.

The details of the implementation are as follows.

Acrylic board

A 660 mm×550 mm×5 mm (W × D × H) translucent acrylic board was used as the table top surface. The actual size of the input surface is 320 mm×240 mm.

Display

A 14-inch LCD display having a screen resolution of 800×600 pixels was placed on the table. Since the screen is separate from the input surface, this prototype is an indirect pointing system. The use of a projector to overlay the screen onto the input surface in order to achieve a direct pointing system was considered, but there was a problem in

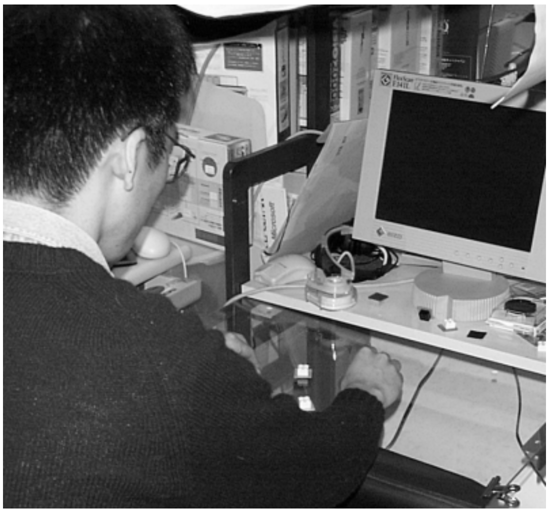


Figure 7.3: User manipulating the prototype

that in order to project the screen onto a surface, the screen must reflect the light from the projector. On the other hand, to track the blocks by the camera below the input surface, the screen must be translucent. A half-translucent board was tested, but the camera could not capture images of the blocks that had insufficient color saturation. In addition, the system sometimes incorrectly identified figures on the screen as blocks. Therefore, an indirect-physical device pointing system was not constructed.

Block

The system discriminates the blocks by colors. Each device has a different colored felt cloth attached to its bottom surface. The felt material was chosen for the smooth feel it provided when moved on the acrylic board. In addition, since the colored surface does not need to touch the acrylic board, the bottom surface can be a different color from the rest of the block.

The form of the block is not restricted, but the bottom surface of the block must be covered almost completely with felt in order to reduce measurement error. The shape of the bottom surface is not restricted, but the bottom surface must be larger than approximately 1 cm^2 for accuracy of measurement. For the prototype, 2 cm^2 blocks were used.

Video camera

A video camera is placed below the acrylic board facing upward. The camera is set in high-speed shutter mode so that it can capture images without motion-blur. If motion-blur occurs in an image, it is difficult to distinguish the colors of the blocks.

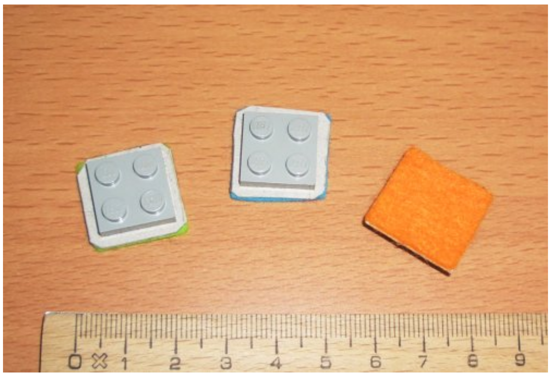


Figure 7.4: The examples of blocks.

Each block consists of a 2 cm^2 piece of cardboard to which colored felt and a LEGO block are attached. The block on the right is turned over to reveal the bottom surface.

The camera is connected to a video capture board installed in a computer. The analysis process of the captured image is described in Section 7.3.3.

Light

To improve the accuracy of color matching, the contrast of the image should be clear. Since the shutter speed of the camera is high and the camera captures the bottom surfaces of the blocks, the contrast is not sufficient. For this reason, a pair of lights is placed below the acrylic board. Daylight light bulbs were used because they are flickerless and have a wide spectrum.

Shield

A piece of black cardboard was placed above the input surface to obscure the ceiling from the view of the camera. In addition, in order to decrease the influence of changes in environmental light, shields were placed around the tables. These shield decreased reflections on the acrylic board from the floor.

7.3.3 Image analysis process

In this section, the process of image processing of position detection is described.

A PC with an AMD Athlon 600 Processor (600 MHz) was used, and the operating system was Linux.

An image captured by the video camera is sent to a capture board installed in the PC. The image is converted into a 320×240 pixel, 50,000 color digital image. Next, the system detects the bottom surfaces of the blocks by color matching. Before using the system, the colors of the bottom surfaces must be registered. Figure 7.5 shows an

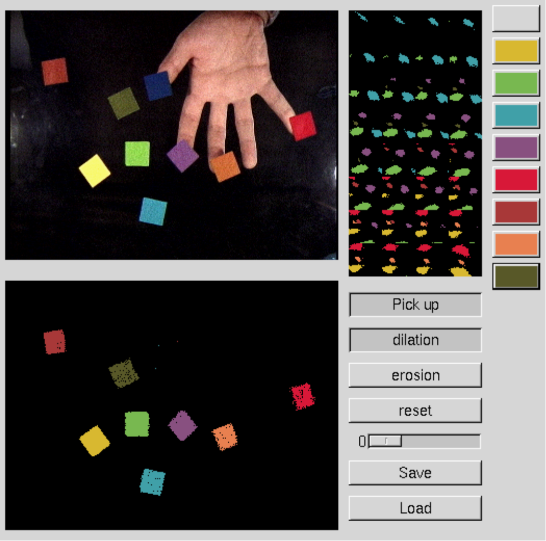


Figure 7.5: Registering colors of blocks

interface for registering the colors. The upper left area of the figure shows a captured image, and the color is registered when a pixel of the image is clicked. Pixels having colors that are registered as block colors are filtered and detected. At this time, the number of devices is limited to eight.

The next step is noise filtering. The noise filter looks for pixels that have eight neighboring pixels of the same color and eliminates the other pixels. Figure 7.6 shows the process of the noise filtering.

The position of each device is given by the average coordinates of all of the pixels, and the coordinates are expanded from 320×240 to 640×480. The accuracy of the coordinates was found to be sufficient if the number of pixels of the device was sufficiently large (approximately 400 pixels, as a rule of thumb).

The computing cost of these processes is relatively small, and as a result, the measurement of device positions is fast. The prototype system achieved 30 scans per second, which was only limited by the video capture speed.

The height of each block is estimated by measuring the change in the size of the bottom surface. In the color registering process, the number of pixels of the bottom surface is also registered. If the number of detected pixels is smaller than the registered value, the system estimates that the block is lifted off of the surface. Applications can use this parameter as an additional input. For this prototype, the threshold between the block touching the surface and the lifted block is 90% of the pixels, which is equivalent to a height of 2 cm from the input surface.

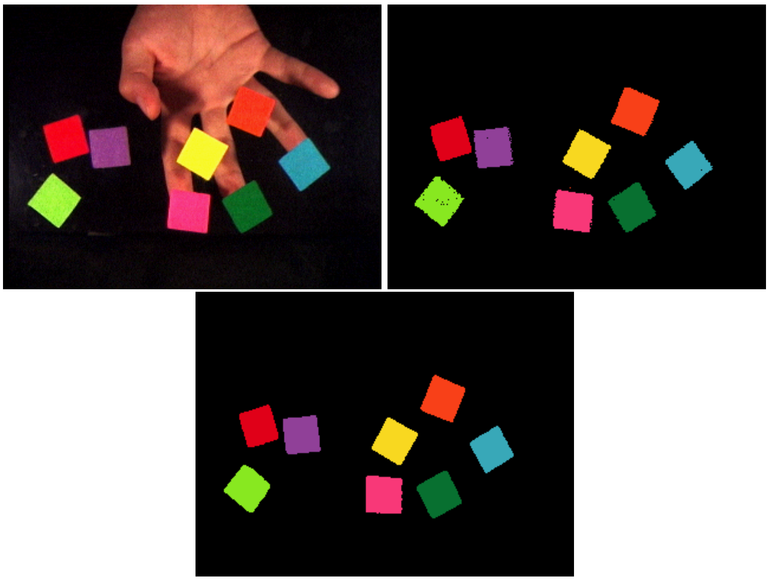


Figure 7.6: Individual noise filtering steps

The upper-left figure is a raw image. The upper-right figure shows the image after pixels filtering using registered color. The lower-center figure shows the image after noise filtering.

7.4 Multipoint input by prototype 1

Next, the multipoint input system of this prototype is described in detail.

7.4.1 Manipulation

The user placed a number of blocks on the acrylic board and manipulates them using primarily his fingers. The user can manipulate blocks as follows:

- Slide a block on the surface
- Pick up a block/put down a block
- Move a block above the surface

The system distinguishes whether a block is touching the surface or lifted off of the surface by applying a threshold to the number of pixels of the block, as described in Section 7.3.3. More precisely, the system cannot distinguish whether a block is grounded or lifted slightly. The system tracks the motion of a lifted block while it is detected.

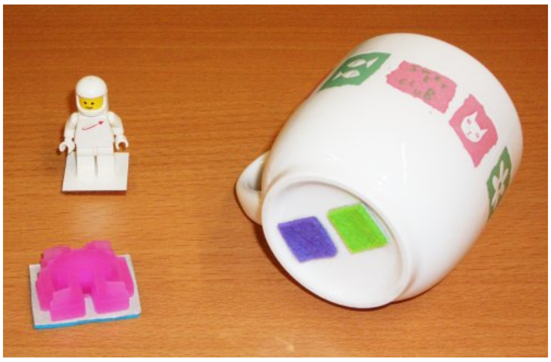


Figure 7.7: Examples of specific devices

The two devices on the left are toys and cardboard. If the cardboard is large enough to obscure the figure from the view of the camera, any objects can be mounted on it. The device on the right consists of a coffee cup and two pieces of cardboards, so that its position and direction can be detected simultaneously.

7.4.2 Specific device input

For a specific device input system, a registered block is bound to a corresponding component of the application. At present, the number of blocks is limited to eight, that is, the system allows concurrent manipulation of eight components. The shape of the blocks is not restricted, and so can be made to represent the corresponding components. Figure 7.7 shows some examples of specific devices.

On the screen, the system displays components according to the positions of the blocks.

7.4.3 Generic device input

For a generic device input, blocks having identical shapes are used, as shown in Figure 7.4. On the screen, the system displays pointers according to the positions of the blocks.

Binding

To control a component using a generic device, a binding operation between components is needed. Two binding operations are used herein.

The first is based on the lifting of a block. In order to attach a block to a component on the screen, the user first picks up a block. At this time, the pointer is displayed in half-tone to indicate that it is lifted. In this state, the pointer is moved onto a component and the block is placed back on the input surface, reattaching the block to the component. When the block is lifted, it is detached from the component. At present, this operation is not user-friendly. When the user lifts a block, because of the recognition process and natural unsteadiness of the user's hand, the position becomes unstable. In addition, the block must be lifted 2 cm above the input surface.

The second method is simpler. When a pointer touches a component, the pointer is attached to the component. This method is easier than the previous method, but the user should move blocks carefully in order to avoid attaching the pointer to another component. This results in a significant restriction on the manipulation of the blocks. In addition, there is no easy way to detach the pointer. At present, the user must pick up the block, or the system detaches them automatically when the binding is not needed. This is described in Section 7.6.

7.5 Applications

Here, a number of experimental applications on the prototype system are introduced.

7.5.1 Bezier curve editing

As a prototype of a drawing tool controlled by a multipoint input, an editing tool using a Bezier curve was developed. A Bezier curve is an interpolation algorithm used to draw a smooth curve from a set of control points. By this application, the user can move eight control points using eight blocks. Each block is bound to a control point, which means that the blocks are specific input devices. In Figure 7.8, the red line indicates connected control points. The black curve represents the interpolated curve.

This type of editing is used for various drawing tools and CAD software. Usually, the individual control points are moved one by one using a mouse. However, the shape of the Bezier curve around each control point is determined by two neighboring points. In order to make the curve that the user wants, a number of adjustment manipulations of the control points are required. On the other hand, using this multipoint input system, the user can manipulate multiple control points simultaneously. A Bezier curve can be edited relatively easily. However, since all of the blocks are bound to control points during editing, a block may sometimes move due to the carelessness of the user.

7.5.2 Parameters control of a physics simulation

As described in Chapter 2, a veteran operator often manipulates multiple components simultaneously. This sample application demonstrates this type of concurrent manipulation. The subject of the application is a physics simulation, and the interface allows the user to control multiple parameters concurrently. This simulation is designed to evaluate the efficiency of concurrent manipulation.

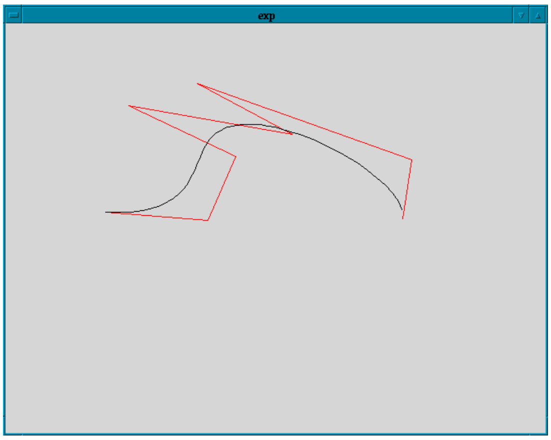
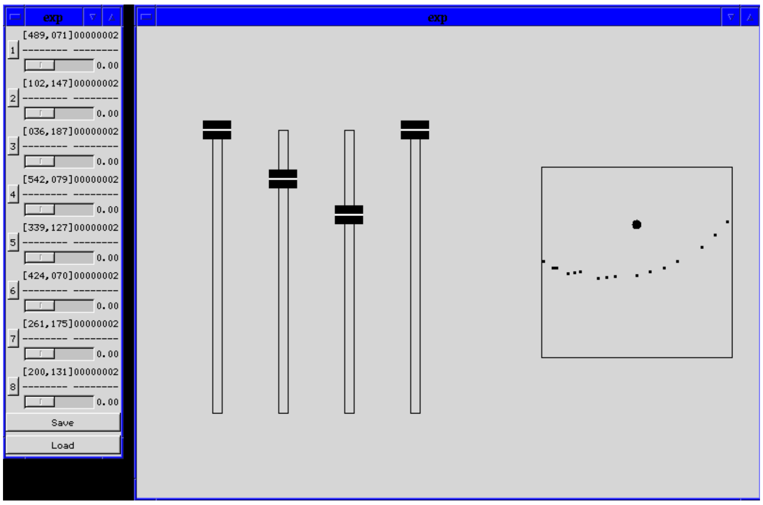


Figure 7.8: Concurrent manipulation of eight control points of a Bezier curve



```
[489,071]00000002
-------- --------
0.00
[102,147]00000002
-------- --------
0.00
[036,187]00000002
-------- --------
0.00
[542,079]00000002
-------- --------
0.00
[339,127]00000002
-------- --------
0.00
[424,070]00000002
-------- --------
0.00
[261,175]00000002
-------- --------
0.00
[200,131]00000002
-------- --------
0.00
```

Figure 7.9: Screenshot of physics simulation software

A visualization of a physics simulation is shown in the right rectangle. Mass points (black dots) are injected from the left and fly under the gravity of a star (black circle). The four sliders to the left of the figure control the angle and the power of the injection, and the position and the power of the gravity of the star.

Figure 7.9 shows the interface of the simulation. The physics simulation is a fictional and very simplified simulation. A star is located in the center of a space, and the mass points are injected sequentially toward the star. The goal of this application is to find an optimum parameter set with which to guide the mass points to an orbital or a certain target point. The simulation has four parameters: the power and the angle of the injection, and the power of the gravity and the vertical coordinate of the star.

We tested this simulation software and found that users rarely manipulate all four sliders simultaneously. In this simulator, the four parameters can be divided into two groups: two of them are for injection and the other two are for the star. The users tended to control either group and manipulate two sliders of the group manually.

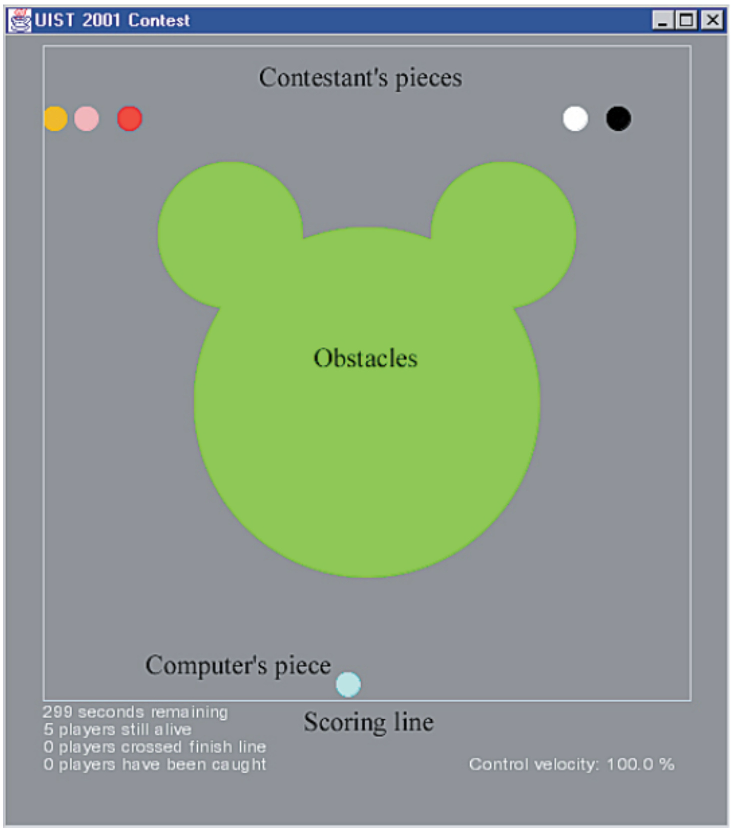
Buxton et al. reported that by-manual interaction is effective for concurrent control of multiple parameters[5]. Buxton's evaluation was confirmed, but the evaluation application includes one subject and two or three parameters, and the subject could focus on the subject. On the other hand, the present application has two objects (the injection and the star), and the user tends to focus on one of these objects, even if they are controllable simultaneously. This indicates that the ability of concurrent manipulation may be limited by the number of objects, and not by the internal parameters.

7.5.3 UIST'01 Interface design contest

At the international User Interface Software and Technology symposium in 2001, the Interface Design Contest, a contest to solve a given subject using UI technology was held[14]. The prototype system discussed in the present thesis and an application were developed as part of this contest.

Figure 7.10 shows the subject of the contest. The goal of the contestants is to move five pieces beyond the scoring line while avoiding capture by the computer's piece. In this game, the contestant should move as many pieces as possible in a limited time. The motion of each piece is sent to a game server from a client application, and the status of the latest game was relayed to the client. Motions are sent as vectors, but the speed of the pieces are limited in the game server. The computer's piece moves linearly toward the closest piece. None of the pieces can move across obstacles.

The multipoint input system was applied to the subject. The simplest design uses five specific devices to manipulate the pieces, but the speed of the piece is limited. Therefore, the devices were used as the targets of the pieces, so that the pieces moved toward a target that was controlled by a device. Five generic input devices were used. At the beginning of a game round, the devices were attached to the pieces according to the order of their X coordinates. By this automatic attachment, the user does not need to sort the devices before a game, but places them on the top of the input surface, while



299 seconds remaining
5 players still alive
0 players crossed finish line
0 players have been caught

Control velocity: 100.0 %

Figure 7.10: UIST'01 Interface Design Contest: specifications

The five small circles are the pieces of the contestant. The task of each contestant is to manipulate these pieces beyond the scoring line while avoiding capture by the computer's piece. The green object at the center of the field is an obstacle.

specific devices must be sorted according to the order of the pieces on the screen.

The above-described system won second prize in the Single User Interface category. Since the contestants' applications were based on various technologies, such as vocal or gesture inputs, and the results depended highly on the maneuvering strategy, these results are not particularly meaningful. Various findings from the contest are described below.

An easy way to manipulate the pieces is to divide the pieces into two groups and lead them in two directions using both hands. In addition, a left-right symmetrical manipulation of the pieces was easy, because the game fields were symmetric and symmetrical manipulation is ergonomically natural for humans. On the other hand, when it was necessary to move the pieces asymmetrically in order to avoid the computer's piece, corresponding blocks were often first moved so as to move some of the blocks far from the computer's piece toward the goal line, and then released. The other blocks were then touched to manipulate the rest of the blocks so as to avoid the computer's piece with both hands.

A weakness of indirect multipoint input was found. As described above, the user frequently releases one set of input devices in order to touch the other set of devices. At this time, the user looks at the input surface to find the blocks and does not see the display. In particular, in this case, the motion of the game pieces had to be followed because the pieces could not be manipulated directly. This presented a disadvantage in that the state of the game could be lost.

7.6 Evaluation

An experiment was conducted to investigate the efficiency of multipoint input compared to single-point input for concurrent user interactions. In addition, specific devices were compared to generic devices.

7.6.1 Method

The experimental application is a sorting task that represents the common drag-and-drop operation used in desktop applications. Figure 7.11 shows a screenshot of the experimental application.

There are eight numbered cards on the lower part of the screen, and the subjects were asked to move the cards into the target squares displayed near the middle of the screen in a designated order. The order is shown inside each square, and at the upper part of the screen. The size of each part is shown in Figure 7.12.

The task was performed using three input types: a single-point input with a generic

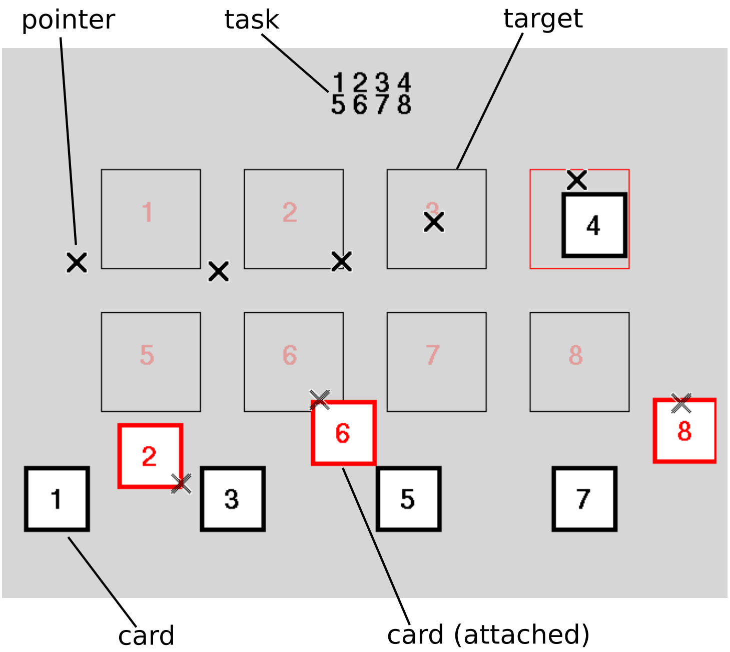


Figure 7.11: Screen showing the experimental application

device, a multipoint input with some generic devices, and a multipoint input with eight specific devices. In the experiment of the multipoint generic devices, the numbers of devices were two, four and eight. The subject performed the sorting task five times. Each task had six problems, and the total time required to solve the problems was recorded. Finally, the data for each task were compared and the five input forms were evaluated.

We used specific devices numbered from 1 through 8, as shown in Figure 7.14. The problems are shown in Figure 7.13. Problem 1, 2, and 3 were intended to have fairly regular ordering of numbers, while the orderings for problems 4, 5, and 6 were made to be random. On the system using generic devices, when all of the cards were moved into their designated squares, all of the bindings were canceled, and the cards were moved back to their initial positions and the next problem was shown.

Seven subjects participated in this experiment. Two were undergraduate seniors and five were graduate students. All of the subjects had sufficient experience in GUI operation using a mouse. The subjects were given explanations of the prototype system and had a training time of approximately two minutes, including the manipulation of multiple devices.

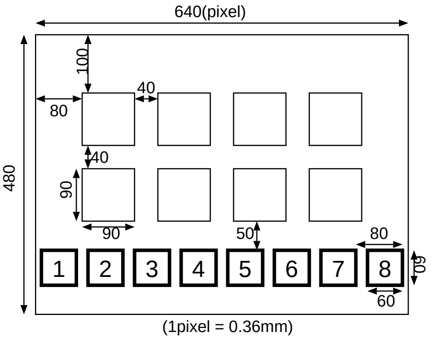
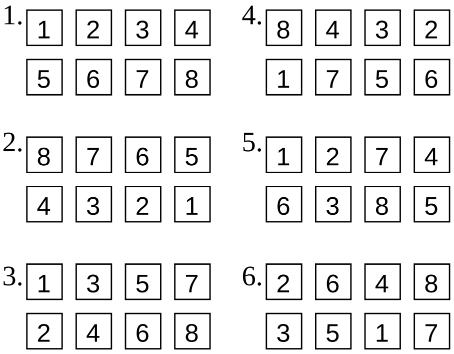


Figure 7.12: Sizes of components



1. 1 2 3 4 / 5 6 7 8

2. 8 7 6 5 / 4 3 2 1

3. 1 3 5 7 / 2 4 6 8

4. 8 4 3 2 / 1 7 5 6

5. 1 2 7 4 / 6 3 8 5

6. 2 6 4 8 / 3 5 1 7

Figure 7.13: Problems

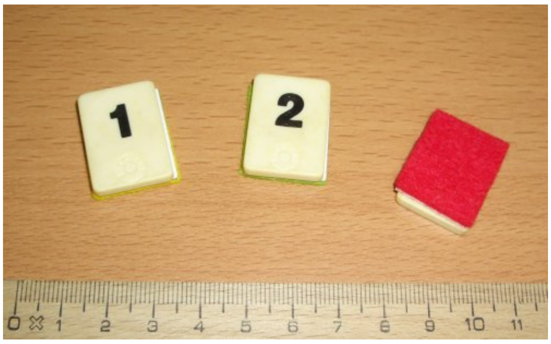


Figure 7.14: Specific devices

Tiles are numbered from 1 to 8. The bottom surface of the right-most tile is shown.

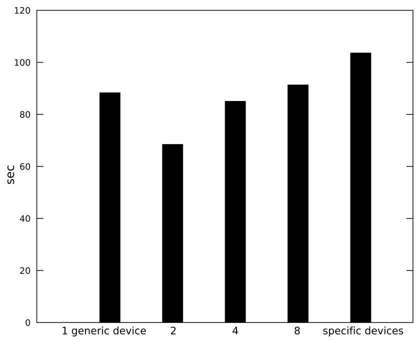


Figure 7.15: Results of the experiment (average total time)

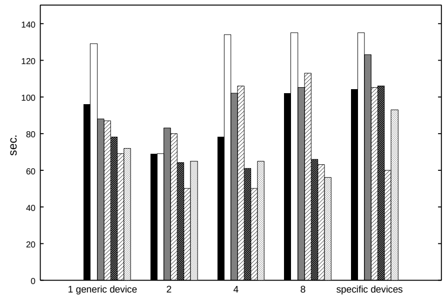


Figure 7.16: Results from each subject (total time)

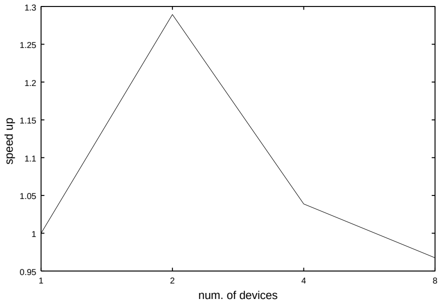


Figure 7.17: Results of the experiment (average speed up)

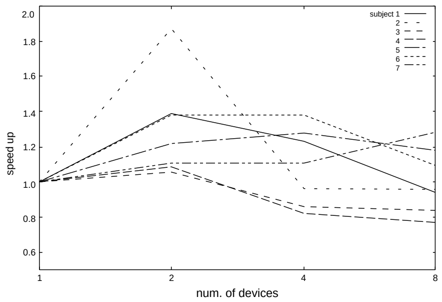


Figure 7.18: Results from each subject (speed up)

7.6.2 Attaching and detaching

On the system using generic devices, pointers are displayed on the screen. When the subject touches a card with the pointer, the card is attached to the device that corresponds to the pointer. The card is then moved by the device directly. When the card is moved into the designated square, the card is detached automatically, and the card is fixed in the square. When the card is moved into the wrong square, the card is not detached and the binding is maintained. This operation is identical for both single-point and multipoint inputs with generic devices.

In addition, two or more cards can be piled atop one another. The card having the larger number is displayed first. When a device is placed on the pile of cards, the displayed card is attached to the device.

7.6.3 Results

Figure 7.15 shows average total time for each input form. With one generic device, the task required 88 seconds on average, and using two generic devices this was reduced to

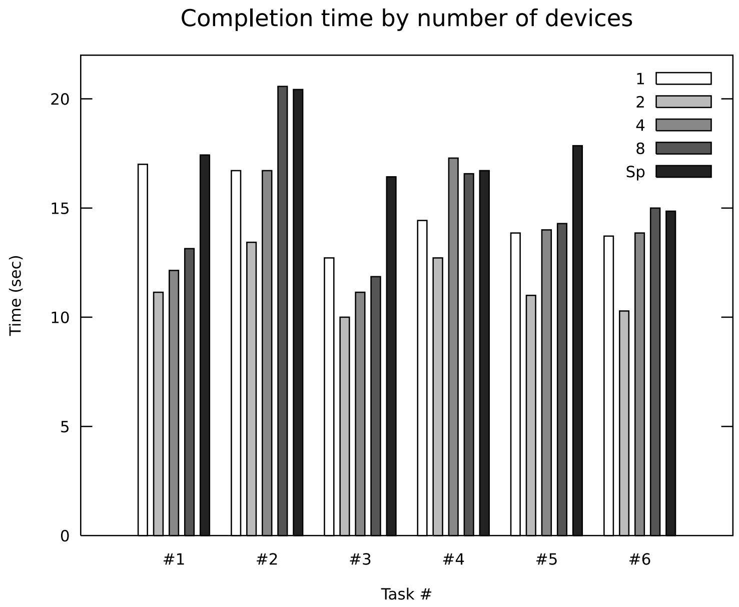


Figure 7.19: The results from each problem (average time)

68 seconds. However, using four devices the time increased (85 seconds) compared to that using two devices, and with eight devices, the time further increased to 91 seconds. With eight specific devices, the task consumed 103 seconds.

On average, the use of more than four devices was not effective for performing the task. On the other hand, using eight specific devices was worse than using eight generic devices because moving the devices took time. In the generic devices test, every card is reset to its initial position and the subject moves the input devices for attachment. In such cases, since the user can choose the closest card to a block, the total cost of moving the blocks can be decreased. On the other hand, this is not possible when using specific devices. Moreover, some subjects used two to four generic devices, even if the subject could use eight devices, whereas the subject had to use eight devices when using specific devices.

Figure 7.17 shows the speed up for the number of generic devices. When two generic devices were used, the resulting speed up was 130%. Using four or eight devices either caused no speed up, or caused a decrease in the score, compared to the use of two devices. Figure 7.18 shows the results for each subject. All of the subjects improved their scores with two generic devices, but with four devices, six of the subjects took longer than using two devices. In particular, three of the subjects had worse scores than using one device. Only one subject had a better score using eight devices, while four subjects resulted in speed-ups of less than 100%.

Figure 7.19 shows the average time for each problem.

7.7 Discussion

The efficiency of two-handed input has been demonstrated in several previous studies. The results of the present experiment confirmed these findings, as the use of two blocks improved the speed by a factor of 1.3. Whereas the use of four or eight blocks was disadvantageous compared to the use of one block, without substantial training.

One possible reason for this is thought to be the locomotive difficulty of moving multiple blocks effectively. When moving blocks by a hand, sliding them simultaneously in the same direction is easy, while moving them somewhat randomly is difficult. In this experiment, subjects had to consider combinations of cards and translate them to concurrent movements of the fingers in an effective, strategic manner. Many subjects began the sorting task without any strategy and bound the blocks to objects at random. These subjects were then confused by the time they got to the phase of moving the blocks. In addition, when exchanging the positions of two blocks, some of the subjects tried to move them using their fingers, which caused them to cross their fingers unnaturally, which wasted their time. This does not apply when moving only two blocks, because only simple locomotive functions of two hands are required.

Another possible reason is the physical conflict between the blocks themselves. When a subject exchanged the positions of two or more blocks, they often released non-related blocks in order to clear the way for the exchange to take place. In such a situation, the released block would typically be moved over to clear the way, or alternatively, some systematic, cyclic exchange is performed by the user, again to avoid physical block conflicts.

Moreover, blocks that were released because they were already detached from a card also sometimes interfered with the other blocks. Such often unexpected and unplanned movements of blocks seemed to cause the user to lose context of the exact locations of the blocks that he is conceptually manipulating. In the experiment, subjects often looked at the blocks to confirm their positions. Although the user was initially predicted to be able to know the positions of the blocks from the positions of pointers on the screen, according to our observations, this was not the case, as many subjects often stopped their hands and looked at the blocks.

The slowdown possibly caused by physical device conflicts was also observed using specific devices. For this experiment, the interaction emulated the typical scenario, where specific devices must be stationary in order to maintain the cards inside correct squares. In some cases, because the moving blocks conflicted with an already released block, the user (almost deliberately) bumped the released block out of the way in order to make room to situate the block that he is currently manipulating.

In order to move multiple blocks in the same direction simultaneously, some subjects did not use their fingers, but rather their palm, and suppressed the blocks and moved them, or used an edge of the hand to gathered the blocks. These types of manipulation are a good strategy for moving multiple blocks simultaneously, but some adjustment of the blocks was required after the manipulation, which sometimes took more time.

7.8 Conclusion

We constructed a prototype system of a multipoint input system, implementing an indirect-physical device input. The experiment was performed in order to investigate the properties of multipoint input with generic devices and specific devices, where two primary properties were observed.

The experiment showed the two chief properties. One is that the efficiency of concurrent control on multipoint input is low when multiple objects are controlled independently. The constraint on the locomotive aspects of human hands could limit various independent controls of the devices. In this experiment, there were no relationships among the cards and required independent manipulation by the subjects. The other is physical conflicts between physical devices, which is practically unavoidable in a physical device system. In addition, using specific devices was initially though to be advantageous over using generic devices due to the costs of attaching operations for generic devices. However, conflicts strongly affected the efficiency of both systems, actually making them less efficient compared to single pointing in the present experiment.

As such, it remains an open question as to whether specific devices will be advantageous in various interactions. Since there is inherent cost due to the nature of physical specialization (such as not being able to switch applications quickly), more research needs to be done to identify situations in which general devices will be advantageous over physical devices, and vice versa.

Chapter 8

Prototype 2: Concurrent Manipulation with Human-body Sensor

8.1 Overview

In the previous section, the first prototype was described using physical devices and the conflicts between the devices were shown to be a problem for multipoint input. A finger-pointing input system was developed to solve this problem.

A flat human-body sensor was employed as an input surface, and the user touches the surface using his fingers or hands. This sensor does not require any special devices and can be used in various situations, while a vision-based input system has some restrictions. The sensor can be overlaid on the surface of a computer screen to construct a direct input system.

To construct a multipoint input system, some known vision processing techniques were applied to data from the sensor to detect fingertips on the surface, and a cost minimization analysis was used to track their motions. As a result, the system can track the motions of fingertips in 20 scans per second, which is same as the scan rate of the sensor.

In addition, by using the recognized shape of the hand, we enabled gathering manipulation by the hands. This proved that a concurrent manipulation without a multipoint input is possible for an input system without physical devices.

This input system was evaluated, and the system can track multiple finger motions stably and was found to work effectively for concurrent manipulation. This system was compared with previous prototypes and its advantages and disadvantages were analyzed.

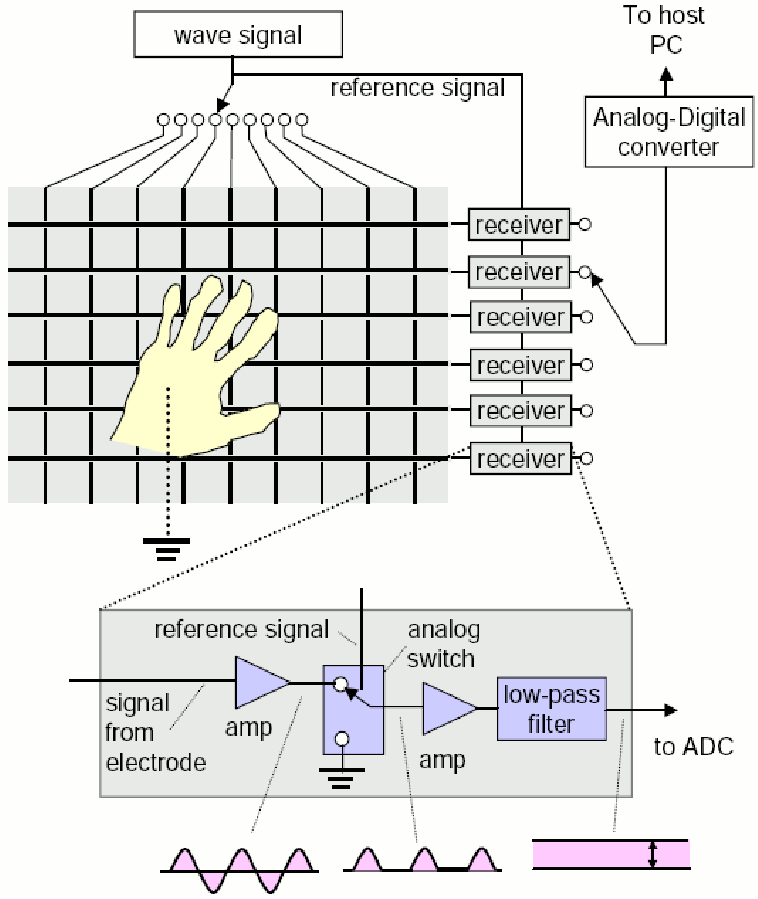


Figure 8.1: SmartSkin sensor configuration

8.2 Body shape sensor: SmartSkin

SmartSkin[51] is a human-body sensing device based on capacitive sensing. The sensor recognizes multiple hand or finger positions and shapes, and calculates the distance between the hand and the surface.

8.2.1 Sensor architecture

SmartSkin consists of grid-shaped electrodes, transmitters, receivers and micro controllers. Figure 8.1 shows the principle of operation of SmartSkin. The vertical wires are transmitter electrodes, and the horizontal wires are receiver electrodes. When one of the transmitters is excited by a wave signal (of typically several hundred kilohertz), the receiver receives this wave signal because each crossing point (transmitter/receiver pairs) acts as a (very weak) capacitor. The magnitude of the received signal is proportional to the frequency and voltage of the transmitted signal, as well as to the capacitance between the two electrodes. When a conductive and grounded object approaches a crossing point, it capacitively couples with the electrodes, and drains the wave signal. As a result, the received signal amplitude becomes weak. By measuring this effect, it is possible to detect the proximity of a conductive object, such as a human hand.

The time-dividing transmitting signal is sent to each of the vertical electrodes, and the system independently measures values from each of the receiver electrodes. These values are integrated to form two-dimensional sensor values. As a result, the system can recognize multiple objects (e.g., hands). If the granularity of the mesh is dense, the system can recognize the shape of the objects.

The received signal may contain noise from nearby electric circuits. To accurately measure signals only from the transmitter electrode, a technique called the lock-in amplifier is used. This technique uses an analogue switch as a phase-sensitive detector. The transmitter signal is used as a reference signal for switching this analog switch, to enable the system to select signals that have a synchronized frequency and the phase of the transmitted signal. Normally, a control signal needs to be created by phase-locking the incoming signal, however, in the present case, the system can simply use the transmitted signal, because the transmitter and the receiver are both on the same circuit board. This feature greatly simplifies the entire sensor design.

The integrated sensor values (see the upper-right of Figure 8.6) is similar to a two-dimensional image. The distance between the human body and the sensor corresponds to the luminance of a pixel. Once these values are obtained, algorithms for image processing can be applied to them to recognize gestures.

8.2.2 SmartSkin Prototypes

There are two SmartSkin prototypes. The first prototype is a table-size system that has 8×9 grid cells on a table top, and each cell is 10×10 cm. The resolution of the sensor data is 10 bits (0 – 1,023) and the scan rate is 10 scans per second. A piece of thin white plywood was placed on the grid, and a computer screen was overlaid on it using a projector set above the table. Figure 8.2 shows the system. This system provides a touch-panel-like input surface, but can recognize multiple hands of users.

The second prototype is a tablet-size system that has 32×24 grid cells, and each cell is 9×9 mm. This prototype consists of three parts: a transmitter board, a receiver and controller board, and a grid board. The transmitter board has a microprocessor, Atmel's AVR AT90S8535, and transmits a rectangular wave using its digital-out. The receiver board has four AT90S8535, and each AT90S8535 receives eight signals amplified by a lock-in amplifier. Received signals are digitized and then transmitted to the PC via USB. FTDI's FT232AM is used as a USB controller. The sensing range is about 5mm

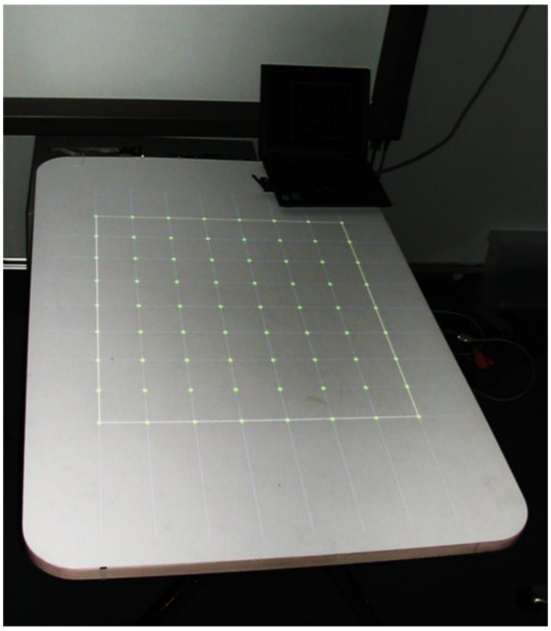


Figure 8.2: Table-size SmartSkin

An overview of the table-size SmartSkin. The grid on the table is an overlaid image from a projector above the table. Its pitch and position is adjusted to the electrode grid behind the table top.

above the grid. The electrode grid was covered with a sheet of plastic insulating film, and a touch-panel-like input system, which the user touches directly, was constructed. Its scan rate was 20 scans per second. Figure 8.3 and Figure 8.4 show overviews of the system.

8.3 Multipoint input on SmartSkin

A multipoint input system was implemented on SmartSkin. With this system, the user touches an input surface by his fingers directly. In order to detect the fingers on the sensor, a tablet-sized SmartSkin was employed for its finer resolution. Its sensing range is 5 mm, which means that the system can distinguish whether or not a finger is touching the surface. As a result, a touch-panel-like multipoint input surface was developed.

This system 1. detects multiple fingertips from the sensor value, and 2. tracks the motions of the fingertips.

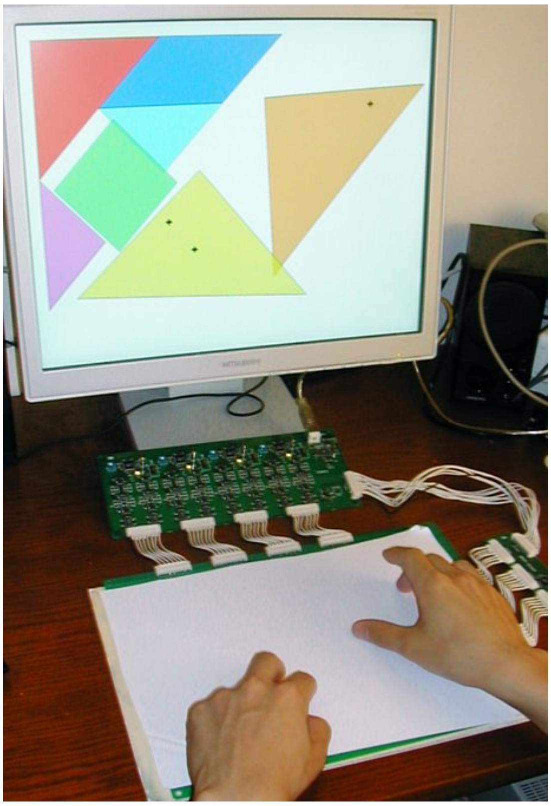


Figure 8.3: Tablet-size SmartSkin with a LCD display

Installation as an indirect-input system.

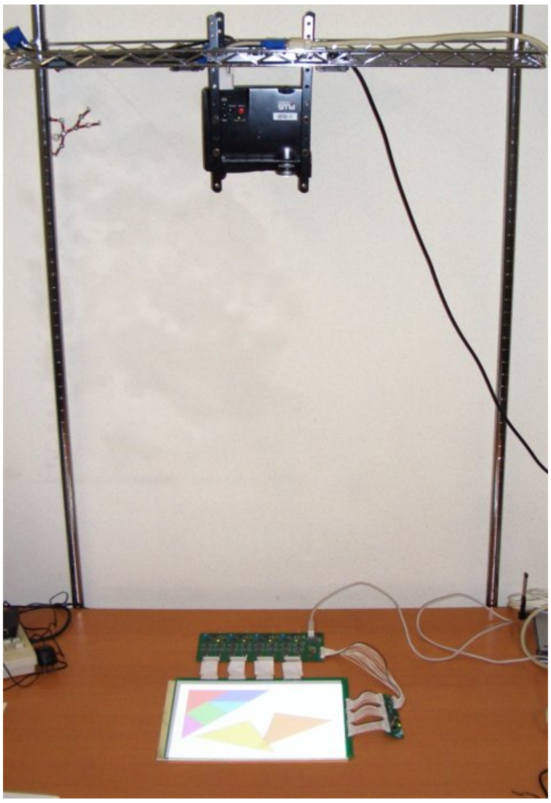


Figure 8.4: Tablet-size SmartSkin with an overlaid screen

Installation as a direct-input system.

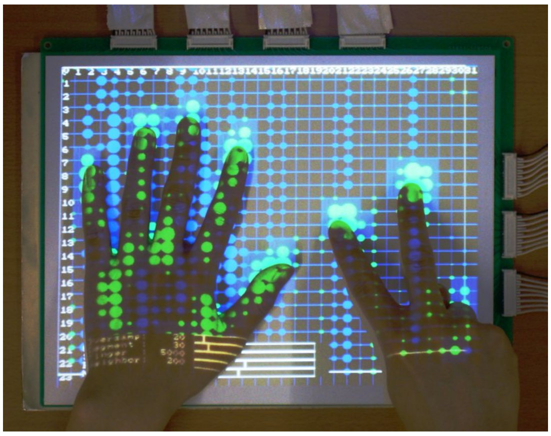


Figure 8.5: Gestures and corresponding sensor values.

Sensor values are shown on the surface of the system shown in Figure 8.4. Each green disc represents a sensor value on a crossing point. The diameter represents the amplitude of the sensor value.

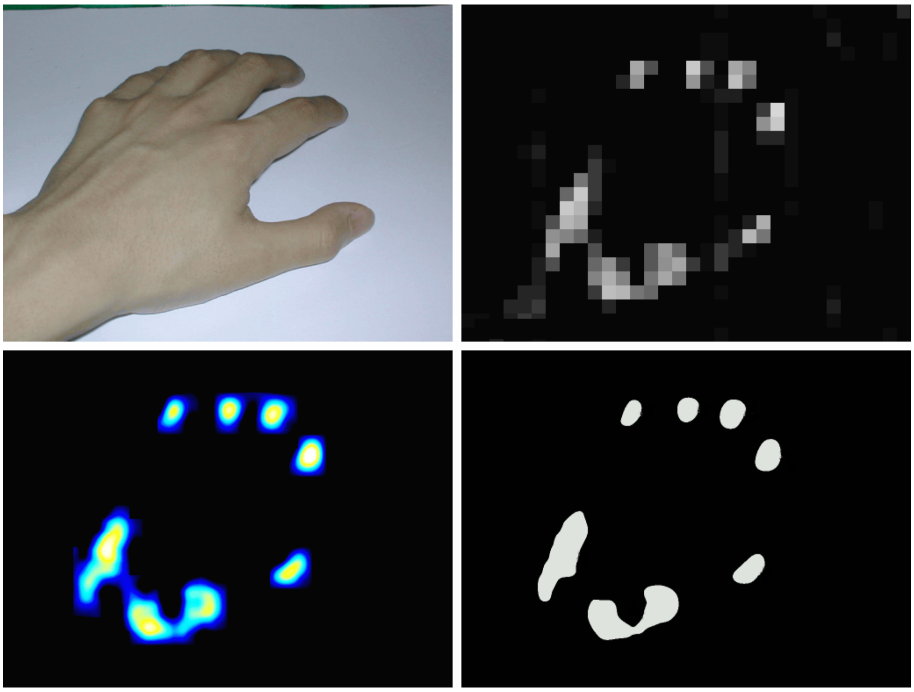


Figure 8.6: Step of fingertip detection: A hand on the SmartSkin (top left), sensor values (top right), interpolated values (bottom left), after the segmentation process (bottom right).

8.3.1 Fingertip detection

First, the system interpolates the sensor values. In this implementation, the 32×24 sensor values are scaled up (e.g., 1,024×768). The bicubic convolution interpolation algorithm was used. This algorithm interpolates two values by using an additional two neighboring values. As shown in Figure 8.6, fingertips can clearly be seen in the interpolated image (bottom right). In order to decrease the computational cost of interpolation, the scale factor is fixed during runtime, and the system makes a lookup table of the functions of the bicubic interpolation. Moreover, if the sensor values of four corners of a grid cell are lower than a threshold, the cell is not interpolated.

The system then detects the fingertips from the interpolated values. First, the values below a threshold are omitted to detect only contact or very close presence to the input surface parts. The bottom right section of Figure 8.6 shows a filtered image.

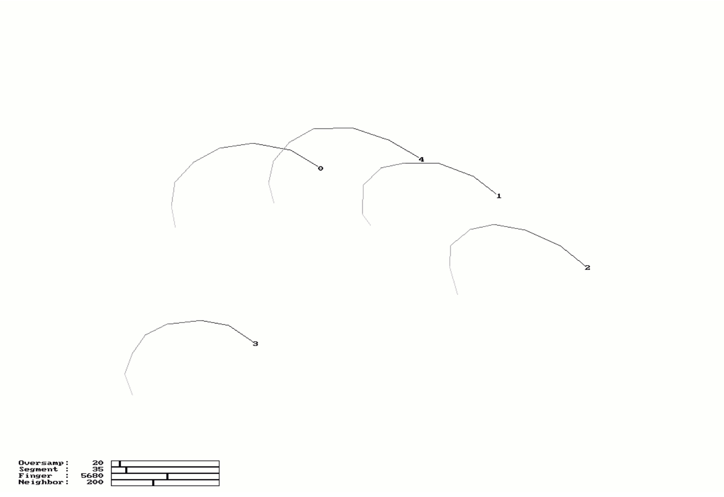
Next, the system divides the values into several continuous regions by using the segmentation algorithm. If the number of pixels of a region is within a certain range, the system recognizes the region as a fingertip and calculates its centroid as the position of the fingertip. This multipoint input system simply ignores regions larger than the range, whereas the system described in Section 8.5 uses them.

When the distance between two fingertips on the input surface is shorter than the pitch of the grid, corresponding regions join each other and the system cannot distinguish them. This has not occurred often, but in order to achieve more accurate recognition, it is necessary to shrink the pitch of the grid and improve the resolution.

8.3.2 Motion tracking

SmartSkin cannot identify which finger of whose hand is on the grid. Therefore, in order to track the motions of the fingers, the system has to estimate the motions by comparing the current positions of fingers and their previous positions. A cost minimization analysis technique was employed for motion tracking. The details are as follows.

In frame t, let $F_{i,t}(i = 1, 2, \ldots, n_t)$ be positions of detected fingertips. When $F_{j,t+1}(j = 1, 2, \ldots, n_{t+1})$ is given as their positions in the next frame $t+1$, the system calculates a *candidate set* of corresponding points $S_{i,t}$ for every $F_{i,t}$: when a cut-off value R is given, $S_{i,t}$ consists of points of $F_{j,t+1}$ which satisfies $|F_{i,t} - F_{j,t+1}| < R$. In the same way, the system calculates $S_{j,t+1}$ for $F_{j,t+1}$, which consists of points of $F_{i,t}$, which satisfies $|F_{i,t} - F_{j,t+1}| < R$. If $F_{i,t}$ has no candidate ($S_{i,t} = \emptyset$), it is considered that the corresponding finger is released, and it is eliminated from the remaining process. . If $F_{j,t}$ has no candidate ($S_{j,t+1} = \emptyset$), it is considered that the finger is newly detected, and it is eliminated as well.



```
Oversamp:   20
Segment :   35
Finger  : 5680
Neighbor:  200
```

Figure 8.7: Motion tracking of fingertips

Five fingertips are detected and their motions are tracked. The user touched the surface with his right hand and moved clockwise.

Next, in order to determine a corresponding point for the rest of $F_{i,t}$, the system checks all of the combinations. Let a matrix T be a combination of $F_{i,t}$ and $F_{j,t+1}$. Its cost is calculated by following equation:

$$\sum_{i,j} T_{i,j}|F_{i,t} - F_{j,t+1}|^2$$

Then, the system compares all of the costs of T and choose the minimum cost. If the number of combinations is too large, the system aborts the tracking and treats all of fingers as being released. In this implementation, the cut-off value R is 200 pixels, and the limit of the number of combinations is 10^6.

Finally, the system uses a Kalman filter to absorb small shaking. However, because the scan rate of SmartSkin is as slow as per 20 scan/sec, its response became worse when the filter was applied naively. In order to avoid this, the system does not apply the filter when the motion length is longer than five pixels. In addition, the system does not use a Kalman filter to estimate the motion.

Figure 8.8 shows an increase in computational time for finger tracking. In the experiment, the user moved his fingers on the input surface while increasing the number of fingers touching the surface to ten. The fingers were moved as shown in Figure 8.7. The computational time is averaged over frames of every number of fingers. The light gray area represents the time required to read the sensor values and interpolate them. The dark gray are represents the time required to detect and track the fingers from the interpolated data. As shown in the graph, the current implementation is sufficiently

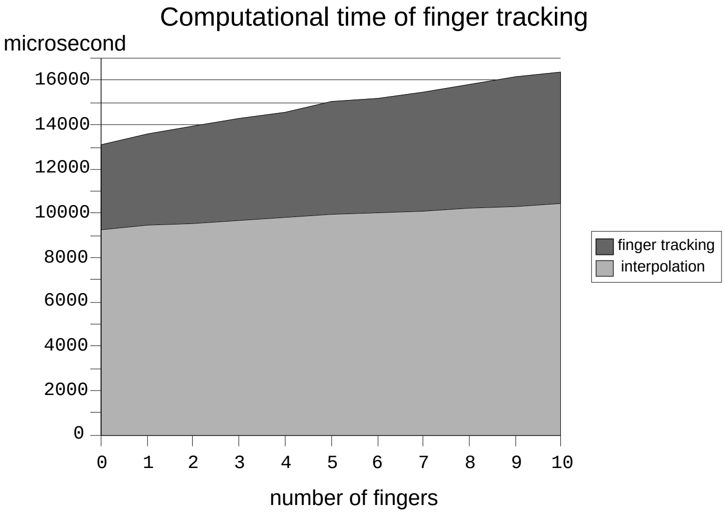


Figure 8.8: Average computational time of finger tracking

Each time is for the processing of one frame. The light gray area represents the time required to read the sensor values and interpolate them, while the dark gray area represents the time to detect and track the fingers from the interpolated data. This measurement was performed on a PentiumM 1.6 GHz machine.

fast for one or two users for real-time processing. However, this processing time can be increased according to the motion or spatial relationships of the fingers.

8.3.3 Specifications

The resolution of finger position is a maximum of 992×744, and the scan rate is 20 scans per second, which is same as the scan rate of SmartSkin. At present, the number of fingers is limited to 50, but the practical limit should be around 20 based on the size of the sensor board. This is sufficient for normal use by a single user.

The stability of the input is described in Section 8.6.1.

8.4 Applications

A number of experimental applications wee developed for the multipoint input system. The goal was to investigate what types of interaction techniques could be implemented on a multipoint input system.

8.4.1 Map viewer

The application shown in Figure 8.9 allows the user to pan, rotate and scale a map with two fingers. When the user points to a position on the map with his finger, the application displays information for that location. When the user touches the map with

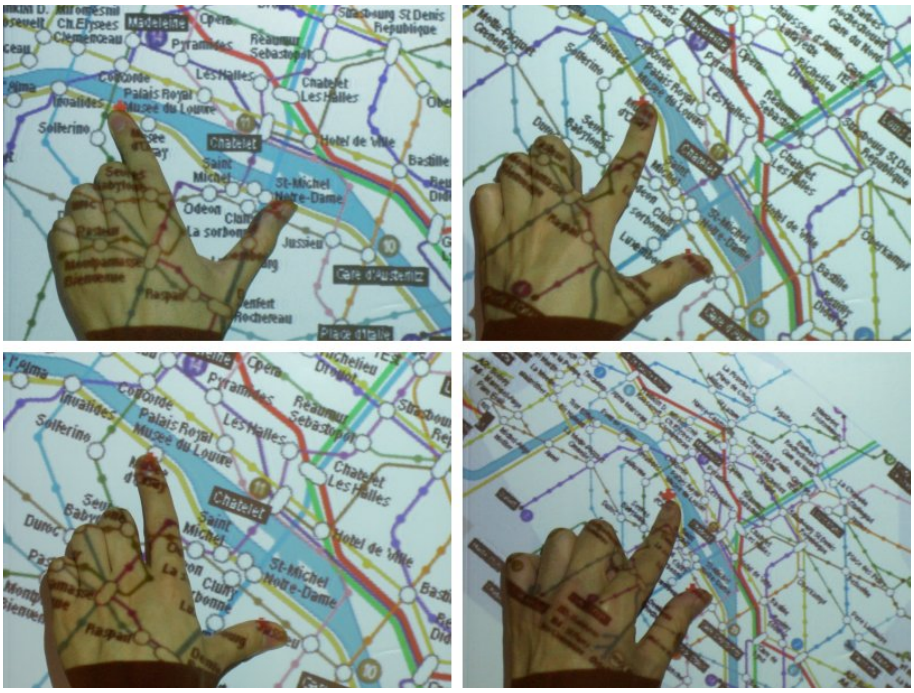


Figure 8.9: Map viewer

The top right image is a rotation of the top left map image. The map at bottom left is scaled up by extending the distance between the two fingers, whereas the map at bottom right is scaled down.

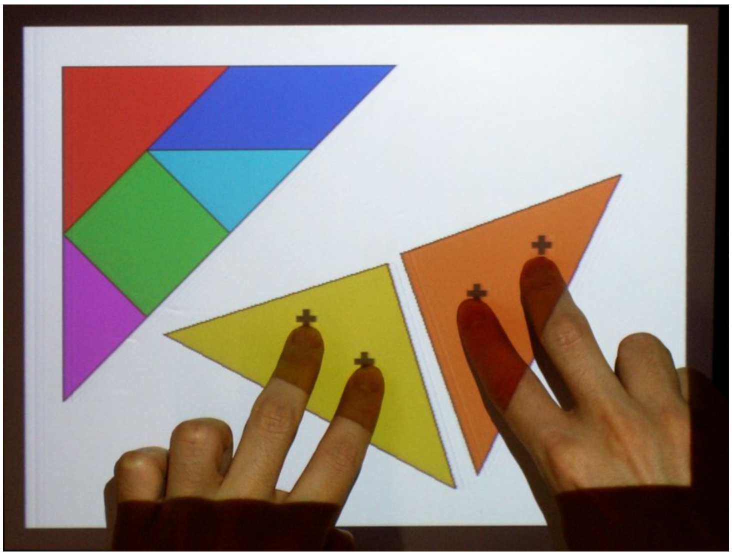


Figure 8.10: Tangram editing

The user moves two Tangram pieces simultaneously. Each piece is moved by two fingers so that the user can move and rotate the piece.

two fingers and moves them, the map is panned or rotated as if the map is stuck to his fingers. When the user changes the distance between the fingers, the scale of the map is changed. In every manipulation, the positions of the fingers on the map are not changed.

The same application is implemented on metaDESK[58], but since the application is controlled by Phicons corresponding to buildings on the map, the user cannot manipulate the map by pointing to any place other than the buildings. Moreover, another Phicon is needed in order to see information on the map. In contrast, in the proposed application, to change the operation mode, the user simply changes the number of fingers on the input surface.

8.4.2 Tangram

As a simplified study of graphic editor, a Tangram application was developed (Figure 8.10). The Tangram is a puzzle that consists of seven triangles and quadrangles. Typically the pieces are made from paper or wood. The objective is to form a specific shape using seven pieces. The shape must contain all of the pieces, which may not overlap.

In this application, the user can move a piece using a finger. By using two fingers, the user can move and rotate a piece as if the piece is stuck to the fingers. However, if the distance between the fingers is changed, the finger that touched the piece first does not change its relative position on the piece.

The number of users is not limited. Anyone can move any number of pieces concurrently. However, the size of the input surface is not sufficiently large to be used by more than three users.

Conventional graphic editors employ icons or keyboard shortcuts to change the operation mode. In contrast, the present application is intuitive and does not require any additional costs to change the mode. Chen et al. reported a similar multipoint input interaction techniques[68][29]. Chen's graphic editor allows the user to rotate a figure by pointing to it with the left and right hands. In the present application the user can rotate a piece with two fingers of one hand.

Scarlatos et al. introduced a tangible Tangram application with physical Tangram pieces with ID tags. The position and orientation of a piece is recognized using a video camera. The computer displays information according to the current status of the pieces. This is a multipoint input system with specific physical devices, and it provides natural and intuitive interaction, but the computer cannot move the pieces. In contrast, the present application enables saving/loading of the status or change the size of the pieces dynamically. The present approach is suited for applications that are based on the advantages of computer software.

Preliminary experiment

A preliminary experiment was performed using the Tangram application and a set of Tangram pieces to determine the difference of manipulation of them.

The Tangram puzzle, Mini Tangoes, by Rex Games, Inc. was used in the present study. The sizes of the pieces were 1.5 – 3 inches square. Subjects were asked to form the shape for which the silhouette was printed on a card. When the subjects finished, or after one minute, the solution was presented to the subjects and they wee asked to form the same shape using the Tangram software application.

Subjects often used both hands to concatenate two or more pieces, as follows. While one hand was used to hold one piece, the other hand moved another piece and concatenated them. Moreover, some subjects used both hands to move multiple pieces while maintaining the shape of the pieces. In such cases, the pieces were pressed somewhat strongly by the fingers so that the pieces were moved while maintaining their relative positions. In addition, some subjects slid or threw pieces. These manipulations are not implemented in the present application. In addition, conflicts between pieces were not calculated. Some subjects tried to rotate two pieces by pointing to them with two fingers.

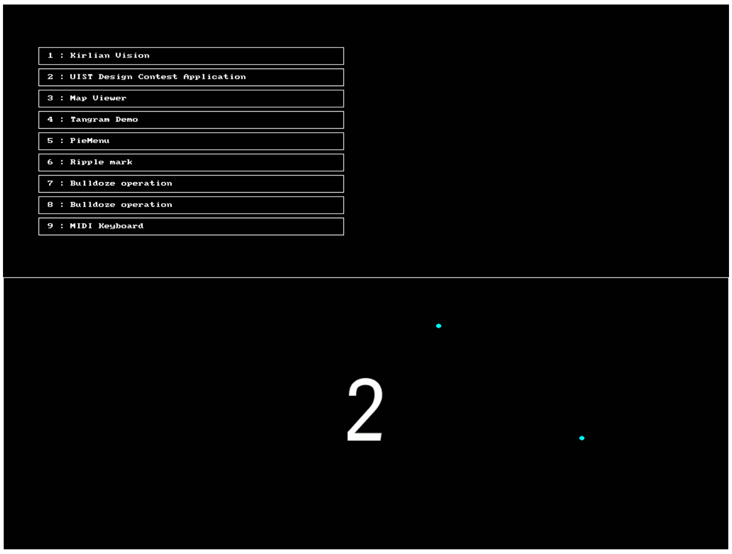


Figure 8.11: Screenshot of Touch Counter

The lower half is the Touch Counter region. A common menu interface is displayed at upper left. This screenshot shows the user inputing 2 by touching with two fingers.

8.4.3 Touch Counter

A demo application of this input system was developed, and the demo application consists of several sub-applications, as described in previous sections. In order to select a sub-application, a menu interface was first provided. To choose an item from the menu, the user touches the item by a finger, as in a common GUI. However, the user must look at the screen and the overlaid menu to select an item, and during a demonstration, the presenter was asked to face to the audience for as long as possible. In addition, selecting an item takes time [1]. This menu interface is not sufficient for demonstration purposes.

Therefore, a new menu selection interface, Touch Counter, was developed (Figure 8.11). Touch Counter has a region occupying half of the screen, and counts the number of fingers touching the region. For example, when a user touches the region with two fingers, 2 is the input number. In addition, when the region is touched within 300 milliseconds after the fingers are released, the number of fingers is added to the previously input number. For example, when the user touches the region with three fingers twice, the input number is 6. In order to select an item, the user inputs the number assigned to the item.

Any place in the region can be touched, so that input is easy, even if the user is not

[1] This will take some time when the item is small (Fitt's law). In contrast, if the menu is too large, it will take a time to move a finger to the item.

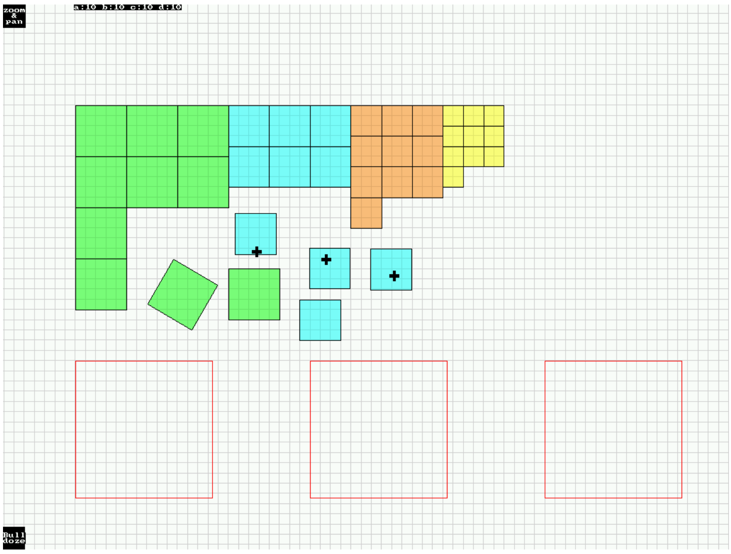


Figure 8.12: Application for the UIST'02 Interface Design Contest

The goal of this application is to move squares at the top of the screen into the frames at the bottom. This screenshot shows that three squares are moved simultaneously. The button at the top left is used to change the operation mode: when pressed, the mode is changed to the panning/scaling mode, and when released the mode is changed to the editing mode.

looking at the screen. Moreover, if there are not so many items in the menu, operation selection can be done in a very short time. On the other hand, the Touch Counter region occupies a large part of the screen.

8.4.4 UIST'02 Interface Design Contest

An Interface Design Contest was held at the ACM symposium, User Interface Software and Technology (UIST), in 2002[56]. The contestants designed and implemented interfaces to solve a given problem. In this case, the problem was a packing problem. The contestants were given squares and that were to be divided into three lots, the squares in each lot were to be packed into the smallest square bin that would hold them. The score of the layout is the sum of the lengths of the square bins. The contestants' goal was to make a layout for which the score was smaller than any competitors.

The multipoint input system on SmartSkin was applied to the problem. It was hypothesized that concurrent manipulation was suited to trial-and-error. The application constructed allows the user to manipulate squares concurrently. The user can move and rotate squares as with the Tangram editor. Moreover, the user can pan and scale the entire field as with the Map viewer. The operation modes of editing mode or panning/scaling mode is changed by pressing a button on the screen. The application was tested using 20 squares and was observed to work sufficiently. Figure 8.12 shows a screenshot of the application.

However, the problem of the contest involved over 100 squares, and the proposed application did not scale for such problem. The criteria of the contest could not be met.

In the contest, most of the contestants employed a semi-automatic packing process as a preprocess. This strategy cut down the initial packing cost. A great deal of time was lost for this initial packing. In addition, the proposed application did not scale for 100 squares because it became difficult to manipulate the squares because the squares were too small when zooming out to see all of the squares, while fewer squares were seen on the screen when zooming in to make the squares larger.

8.4.5 Shape manipulation of 2D graphics

Igarashi et al. developed a shape deformation algorithm[24]. Igarashi and Moscovich applied a multipoint input on SmartSkin to the algorithm.

The algorithm provides an interactive system that lets the user move and deform a two-dimensional shape without manually establishing a skeleton or freeform deformation domain beforehand.

To deform a shape by this algorithm, the user moves some points on the shape. A multipoint input system allows the user to point to the shape by his fingers directly, as shown in Figure 8.13. This approach provides an intuitive interface to make an animation, and even beginners can easily move and deform shapes at will.

8.5 Bulldozer manipulation

As described in the previous sections, a pointing input system was developed for concurrent manipulation. In this section, an interaction technique is introduced for concurrent manipulation without pointing manipulation. This technique can be used with the multipoint input system.

Fitzmaurice et al. reported that when they provided a pile of colored LEGO blocks and asked subjects to separate them by color, the subjects slid the blocks simultaneously rather than of picking up and dropping them, using their hands and arms like a ‘bulldozer’[13]. A similar manipulation for moving multiple blocks was observed during the experiment on prototype 1. (See Section 7.7.)

Bulldozer manipulation can be performed by physical contact between the objects and the user’s hand, and this kind of manipulation was not possible on the input system without physical devices.

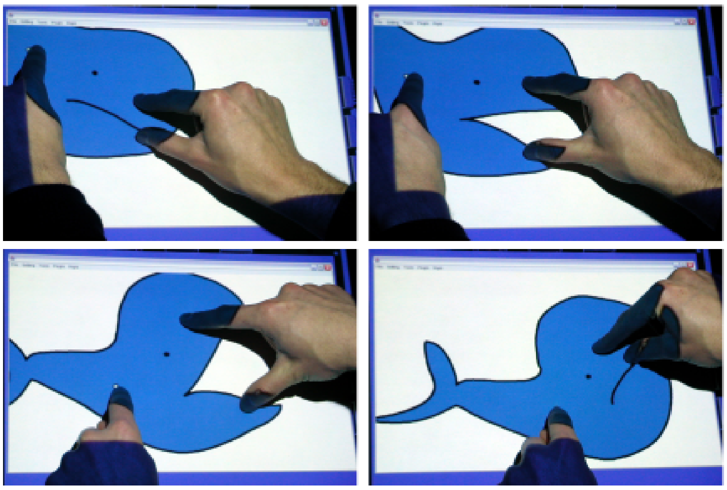


Figure 8.13: Shape manipulation by using SmartSkin

The user can move and deform the shape using his fingers. The shape is deformed according to the positions touched by the fingers. The relative positions of the fingers do not change during the deformation. This figure is taken from [24].

In order to simulate the bulldozer manipulation on SmartSkin, the fact that SmartSkin can capture the contact surface between the input surface and the user's hands is considered. By capturing the contact surface the system can track the posture of the hands on the input surface so that it can calculate collisions between virtual components and the hands. However, it is difficult to track a hand from a contact surface. There are many known techniques to estimate the posture of a hand from video images, but it is not possible to apply these techniques to contact surface data for the following reasons. First, the shape of the contact surface changes dynamically, but since the shape data from SmartSkin has fewer features it is difficult to estimate the motion between the frames. Second, the contact surface represents very limited posture information, and even distinguishing the left and right hand from a contact surface is difficult when the hand only partially touches the input surface.

For these reasons, rather than estimating the posture of the hands, bulldozer manipulation was implemented by the following two methods. The first method treats the contact surface as a physical object and simulates collisions between components and the contact surface using the potential field of the contact surface. The second method estimates the motion of the contact surface using an optical flow tracking.

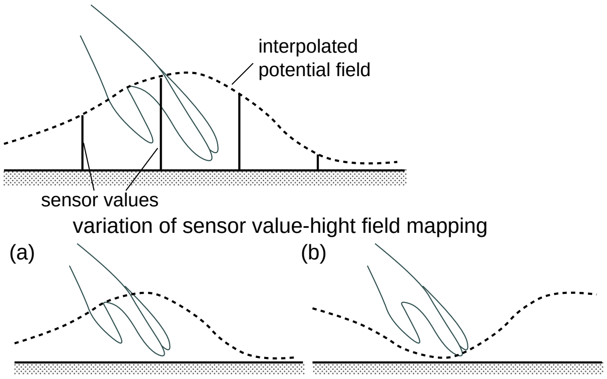


Figure 8.14: Two methods of creating a potential field

Creating a potential field from the sensor values. There are two methods: (a) When a hand is close to the surface, the potential of that point is high. In this case, the object on the potential field moves away from the hand. (b) When a hand is close to the surface, the potential is low. In this case, the object is drawn toward the hand.

8.5.1 Using a potential field

When a user touches the input surface with his hands, the contact surface is recognized, as shown in Figure 8.6. The contact surface is smoothed using a bicubic convolution interpolation, as described in Section 8.3.1. This smoothed surface was used as a potential field and the dynamics of objects on the potential field were calculated so that the objects move on the field according to the position and shape of the hands simultaneously. Each object, which is treated as a point mass, rolls on the potential field toward the lower position.

There are two method by which to create a potential field from the sensor values. Figure 8.14 shows these methods. Method (a) makes the potential high when a hand is close to the surface, which means that an object on the potential field moves away from the hand. In contrast, method (b) makes the potential low when a hand is close to the surface, which means that an object is drawn to the hand. Both methods were tested for bulldozer manipulation, and method (a) was found to be better for this purpose.

By using the potential field, the field does not reflect the speed of the hand, therefore, when the hand is moved quickly, it often passes through the objects.

In this case, the objects are forced by the potential field in the opposite direction. To avoid this problem, the force from the potential field could be made stronger, but when the user touches the objects they would move too quickly, even if the hand is still.

The bulldozer manipulation technique was applied to an entertainment application described in Section 10.1, but this approach is not suitable for desktop application because of the abovementioned problems.

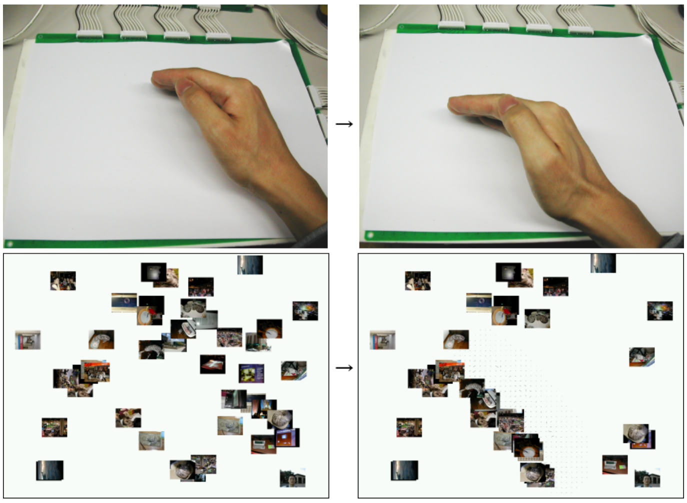


Figure 8.15: Example motion of bulldozer manipulation

The user moves his hand on the input surface from right to left (top left to top right), and icons on the screen are moved from right to left concurrently.

8.5.2 Using optical flow

Implementing bulldozer manipulation

In this method, optical flow is used to track the motions of the hands.

First, the image of the contact surface is scaled down to 64×48 and the system applies optical flow analysis. An optical flow routine of Intel's Open Computer Vision Library was used. The Horn & Schunck method was selected. Note that the sensor value represents a capacitance, but the value was set as luminance for the optical flow routine, which worked sufficiently.

The optical flow routine outputs a vector field V. When an object is on the points $p_i (p = 1, 2, \ldots, n)$ of the V, a force to the object is represented as $\sum_{0<i\leq n} V_{p_i}$. By calculating the dynamics of objects, multiple objects are moved concurrently according to the motion of the hands.

This method is superior to the previous method for a number of reasons. Objects are not forced in the direction opposite of the user's will, and the user can place his hand on an object. The hand may pass through the objects, but the user can perform the same manipulation repeatedly, just like brushing a desk to remove eraser dust.

Figure 8.15 shows a sample motion of bulldozer manipulation. There are several icons on a desktop. When a multipoint input is used to move them concurrently, at most, 10 icons can be moved. In contrast, the bulldozer manipulation allows the user to move more icons.

Integration with multipoint input

As stated previously, a small part of the contact surface is recognized as a fingertip, and the rest of the surface is ignored in the multipoint input system. Here, the larger part of the contact surface is used for the bulldozer manipulation, so that both interaction methods are integrated for concurrent manipulation. The user can use both methods by touching the input surface in different manners. Moreover, the user can perform both manipulations at the same time.

8.5.3 Applications

An application using the bulldozer operation will be described in Section 10.1 in the next chapter.

8.5.4 Discussion

The current two implementations do not track the hand motion and the collisions between the hand and the components completely. As such, a component can sometimes pass through a hand, or the bulldozer manipulation must be repeated in order to complete a task. In order to avoid this problem, it is necessary to track the motion of a hand with a high degree of certainty. The condensation method can be used to track the motion of the contact surface. This method is known as a robust method to recognize the contour of an object in a video image and track its motion. Using this method, interpolation of the motion of corresponding contact surfaces between frames, as well as the calculation of collisions, is possible.

During a bulldozer manipulation, components were often gathered into a heap, and it was often necessary to move components by pointing input in order to break up the heap to spread out the components. This could be avoided by introducing a collision solver between components. This solver was introduced and worked well to solve this problem, as described in Section 10.3. However, because the motion of a component is restricted by the solver, the user is unable to move some components freely. This causes a gap between manipulations by the user and the result from the application. The system should provide feedback to the user that the user loosing control. Because this input system gives the feeling that the user is touching components virtually (direct manipulation), it would be reasonable to provide a tactile feedback from the input surface to the user's hand.

8.6 Evaluation

Three experiments were performed in order to evaluate the multipoint input system on SmartSkin.

8.6.1 Stability test of finger pointing

Goal

The goal of this experiment is to evaluate the stability of the implementation of the multipoint input on SmartSkin. The stability was evaluated by measuring the motion of the detected finger position during the duration of touching.

Method

A sheet with 25 numbered squares was printed on the input surface as a guide, and subjects were asked to touch the squares in the order of the corresponding numbers. Figure 8.16 shows the environment for this experiment.

The length of the sides of the squares is 18.375 mm, and the squares are placed at 18.375 mm intervals, in order to make the relative positions of the centers of a squares on the SmartSkin grid differ from each other. The squares are formed in a 5×5 grid, and the numbers are placed from bottom left to top right (see Figure 8.17).

On the top of the input surface, a message window is overlaid by a projector and it displays instructions for the subject. The application asks the subject to first touch a number. When the system detects a touch, its position is measured. The application then asks the subject to touch a number continuously for a short time. The application then measures the average position of the finger. The application drops the first 10 frames (0.5 seconds) and measures the position of the finger for 20 frames, and then asks the subject to release his finger. When the finger is released, the application stores the last recognized position. The application repeats this process 25 times.

The measured data was analyzed as follows. The average position over 20 frames for each square is the central value of the square. Then, the accumulated error from the distance between the central value and the position of the finger on each frame was calculated. An error on touching is the distance between the central value and the position of the first touch. An error on releasing is the distance between the central value and the last recognized position.

Nine subjects participated in this experiment.

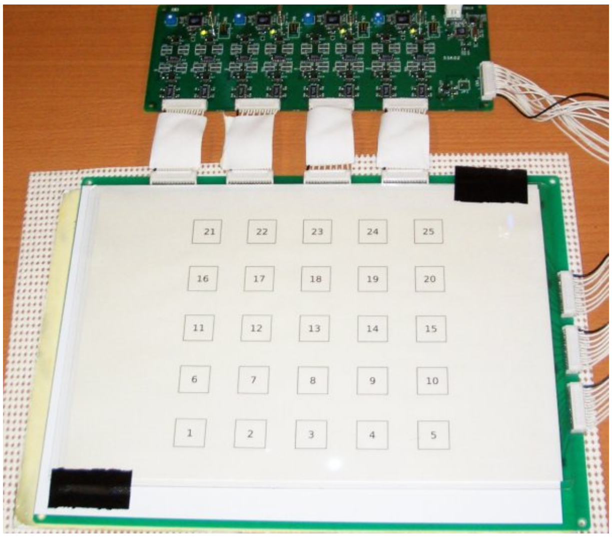


Figure 8.16: Environment of the stability test

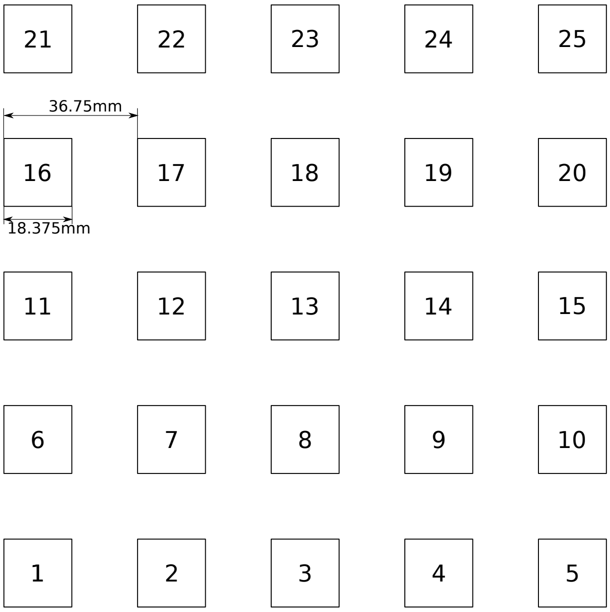


Figure 8.17: Sizes of the guide squares

Table 8.1: Results of the stability test

Action	Average	Standard deviation	Minimum	Maximum
Press	5.23	5.46	0.00	25.3
Touch	1.93	1.63	0.00	8.04
Release	0.976	1.41	0.00	9.33

Results

The results for nine subjects are shown in Table 8.1. The data were averaged on 225 squares (25 squares for nine subjects). The same result is represented as a graph in Figure 8.18. The graph shows the minimum error, first quartile, median, third quartile and maximum error of the accumulated error. In addition, one of the touching errors was 210 pixels, which seems impossible. After the experiment, this was found to be caused by an error in the program of the application. Therefore, this data was removed from the experimental data set and the final results were calculated.

The average accumulated error while touching the input surface was 5.23 pixels. When averaged over 20 frames, the average accumulated error was 0.26 pixels per frame. The first quartile was 0. The third quartile was 9.6 pixels, and the averaged result was 0.48 per frame. The input was concluded to be sufficiently stable.

The average touching error was 1.93 pixels, and the maximum was 8 pixels. One reason for the error is that when a fingertip is touched to the input surface, the edge of the fingertip is recognized. However, the finger is soon pressed to the surface, and the middle of the fingertip is recognized. The recognized position thus moves towards the root of the finger.

The releasing error was less than 3 pixels until the third quartiles, but the maximum error was 9.33 pixels.

Discussion

The error while pressing the input surface (0.26 pixel) is stable. However, the touching and releasing error may cause a problem for real applications because it can change the status of the corresponding component upon touching, or can break the adjusted parameters on releasing. To avoid this problem, the system should drop some frames upon touching and releasing. In addition, let us consider the physical input device. The traction and inertia of the device prevent sudden motion so as to avoid this type of error. This frictional force is added to the components and to solve this problem.

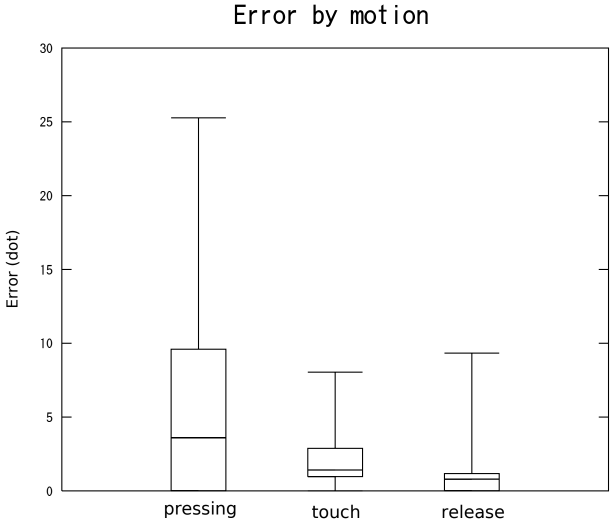


Figure 8.18: Results of the stability test

8.6.2 Supplementary evaluation of sorting test

Goal

In this experiment, the difference between the physical device input and direct touching input is considered. As such, a supplementary experiment of a card sorting task was conducted on prototype 1, as described in Section 7.6. This type of multipoint input was not sufficient for concurrent manipulation, which requires independent manipulation of the components. Therefore, this supplementary experiment is not suitable for evaluating the concurrent manipulation, but comparing the two evaluation is a good way to observe the differences between them.

Method

The details of the experiment are described in Section 7.6. In this experiment, because of the difference in the implementation, a few of the details of the method were changed. First, a card is attached when touched by a finger and detached when the finger is released. The automated detaching introduced for prototype 1 is not implemented in this experiment.

Figure 8.19 shows a screenshot of the experimental application. The size of each component is different than in the previous experiment, but the proportions are the same. The size of the screen in the previous experiment was 320×240 mm, while in this experiment, it was 285×214 mm.

In this experiment, a subject solved the six problems using one finger, and then solved the same problems using any number of fingers of both hands. Each subject

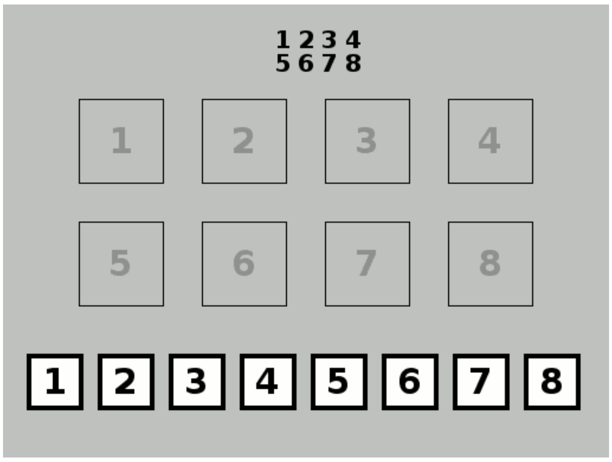


Figure 8.19: Screenshot of the supplementary experiment application

Table 8.2: Average time to finish each problem

Problem #	Single pointing (seconds)	Multiple pointing (seconds)	Speed up (%)
1	13.598	10.389	23.60
2	19.778	17.547	11.28
3	18.317	17.849	2.55
4	17.168	16.916	1.47
5	17.719	16.670	5.92
6	16.639	16.355	1.70
Total	103.219	95.727	7.259

was told to solve the problems as quickly as possible. No instructions or manipulation strategies were provided.

The nine subjects who participated in the experiment described in the previous section also participated in this experiment. None of them participated in the experiment on prototype 1.

Results

The results of the experiment are shown in Table 8.2. The table shows the averaged time required to finish each problem. Figure 8.20 shows the results in a graph.

As shown in the table, the score of the multipoint input was better than that of single-point input, but the difference is small. In problem #1, the score of the multipoint input was improved 23%, but it improved 1 – 11 % in the other problems, and the total improvement was 7%.

In addition, the ratio of time between the numbers of cards manipulated at the same time was determined. Figure 8.21 shows the result.

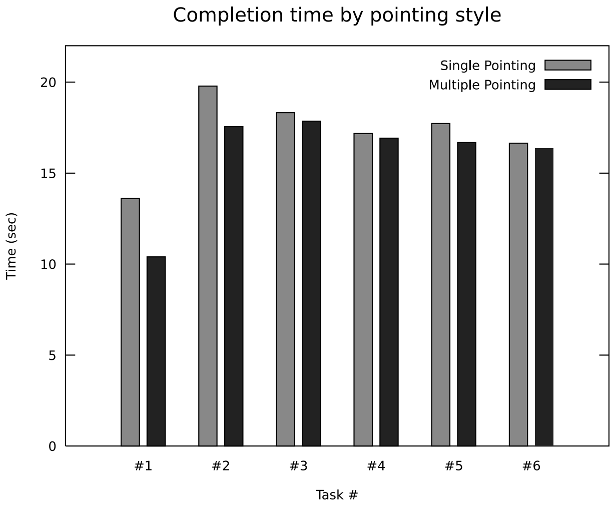


Figure 8.20: Average time to finish each problem

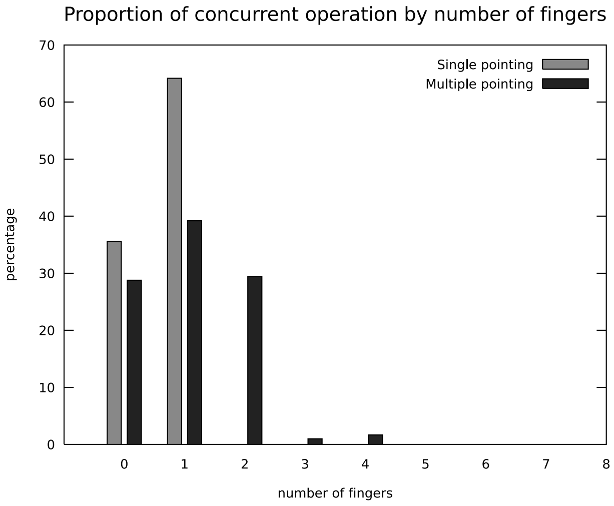


Figure 8.21: Ratio of total time and the number of cards manipulated concurrently

Discussion

The reason why the score of problem #1 was improved with the multipoint input is that concurrent manipulation is suited to move the cards from the initial positions to the target squares, as shown in Figure 8.19. In addition, for problems #1 to #2, the order of the cards should be reversed, but this operation can be performed in an orderly manner so that the concurrent manipulation was efficient. The times to finish problems #3 – #6 were hardly improved by concurrent manipulation because in these problems it was difficult to manipulate the cards concurrently. This tendency was observed in the previous experiment, as shown in Figure 7.19.

In the previous experiment, the score was improved 30% with two devices, but the score became worse when eight devices were used. In this experiment, since the subjects were told to use their fingers freely, the results cannot be compared with the previous results in a straightforward manner. However, most of the manipulation was performed using two fingers, as shown in Figure 8.21. Therefore, the results of this experiment should be compared with the results obtained using two devices. From these results, the speed up by multipoint input was 7%, which was lower than the result from prototype 1. As shown in Figure 8.21, 40% of the manipulation was performed with one finger and concurrent manipulation was 30% of the total. This indicates that concurrent manipulation was not used effectively.

One reason for this is that the experimental system did not detach a card automatically, whereas the previous system detaches a card when it is in an appropriate target. Many subjects adjusted cards by placing them on the targets. This could be the reason why the system was manipulated with one finger. Moreover, the system does not reset the positions of the cards at the beginning of each problem. Therefore, at the beginning of each problem, some of the subjects first moved the cards out of the target square. In contrast, the previous experiment system reset the positions.

Some subjects reported that using both hands covered the input surface so that it was difficult to see the components on the screen. In this system, the greater part of the input surface can be covered with both hands, and this problem may have hindered concurrent manipulation.

8.6.3 Evaluation of continuous concurrent manipulation

Goal

In the experiment described in the previous section, when a component was adjusted to an appropriate place, it did not need to be manipulated until its target was changed. In a real application, this means that the user knows where the component should be

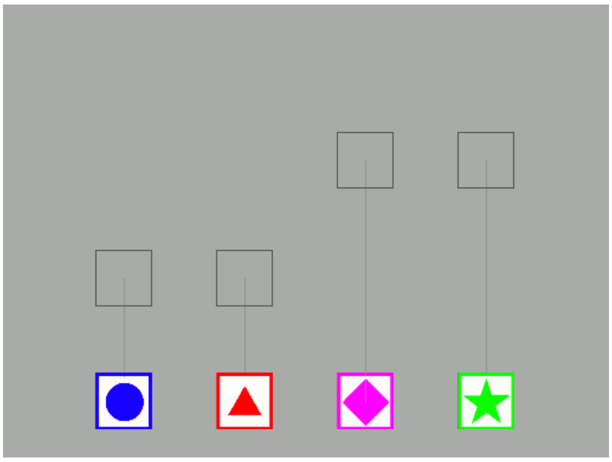


Figure 8.22: Screenshot of the application of the experiment

moved to before the operation. For this reason, the components were moved and set down repeatedly. In other words, most of the manipulation was performed discretely.

However, in the field of the target of the present research, the parameters of an application must be manipulated continuously. For example, slide volumes are manipulated continuously to change the volumes of the sounds slowly and smoothly. The previous experiment did not clarify the efficiency of the multipoint input system on such applications.

For this reason, an experiment was conducted to evaluate the multipoint input system on continuous concurrent manipulation. In this experiment, a subject tracks multiple continuously moving targets by manipulating multiple components concurrently.

Method

Figure 8.22 shows the screen used for the experimental application. There are four cards on the screen, each of which has a unique color and symbol. Each card has a corresponding target, which is indicated by a black square, and the targets are connected by a black line. The targets move continuously, and the task of the experiment is to keep the cards on the target as accurately as possible.

Figure 8.23 shows the motions of the targets. The task consists of four phases. In the first phase, the targets move up and down horizontally in 2-pixel steps. In the second phase, the two targets on the left move as in the first phase, and the two targets on the right move up and down twice in 4-pixel steps. In the third phase, the targets move in 2-pixel steps along the Y axis and 0.5-pixel steps along the X axis. The two targets on the left first move to the upper left, and then move to the upper right. When they reach the top of the screen, they move to the lower left and then to the lower right.

The two targets on the right move symmetrically. In the fourth phase, the cards move horizontally and repeatedly in 1-, 2-, 3- and 4-pixel steps from the right to the left. Each phase is 300 steps. One step is 0.05 seconds, which means that each phase is 15 seconds. The total duration of the task is one minute.

Subjects were given instructions to manipulate the cards, and practice trials (first and second phases) with both hands were conducted. The subjects first performed the task with one finger, and then performed the task with multiple fingers. Moreover, as a supplemental experiment, subjects were asked to manipulate one card and to track its target with a finger.

Nine subjects who participated in the previous experiment participated in this experiment. However, one of the subjects did not perform the supplemental experiment.

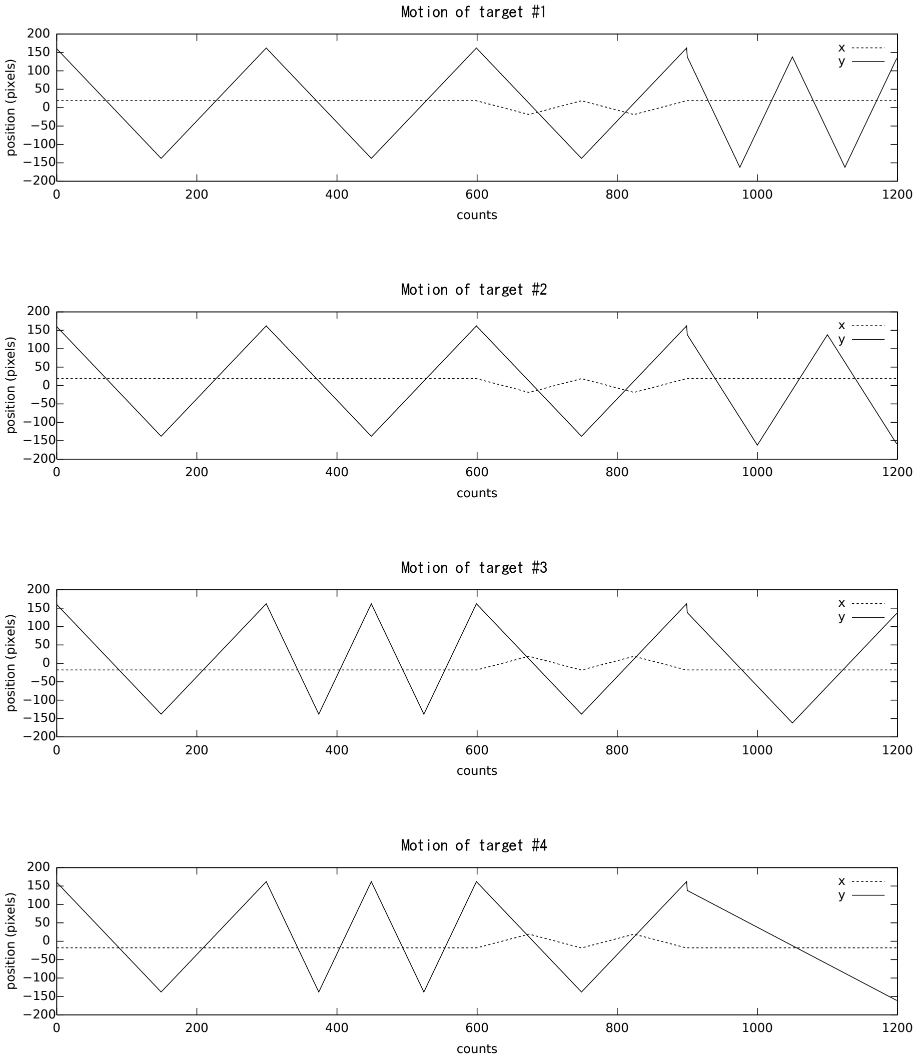


Figure 8.23: Motions of targets

The coordinates are relative positions from the center of the average of the maximum and minimum values.

Table 8.3: Total amount of error in each phase

Phase	Single pointing	Multiple pointing	Single tracking
1	48329	10649	1530
2	66654	17284	1746
3	54415	14637	1789
4	63845	65481	2403
Total	233233	108051	7468

unit = pixel

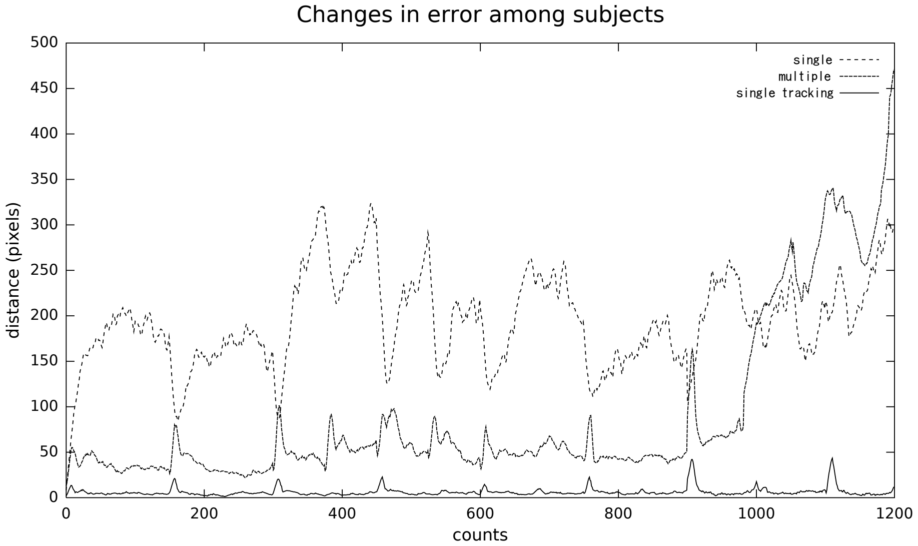


Figure 8.24: Changes in the average error

Results

The results are shown in Figure 8.24. The graphs show the changes in the average score for the subjects at each step. The score is the sum of the distance between a card and its target. The line denoted by 'single' represents the results for single-point input, and the line denoted by 'multiple' represents the results for multipoint input. The line denoted by 'single tracking' represents the results for the supplemental experiment. Table 8.3 shows the total error for every phase.

Analysis

As shown in the graph, from phase 1 to phase 3, the multipoint input improved the score compared to the single-point input in most cases. From the total score of each phase, the accuracy of the tracking was improved four-fold with the multipoint input in phases 1–3. However, in phase 4, the errors of both input methods were equivalent.

During phases 1–3, the subjects could continuously track the target without confu-

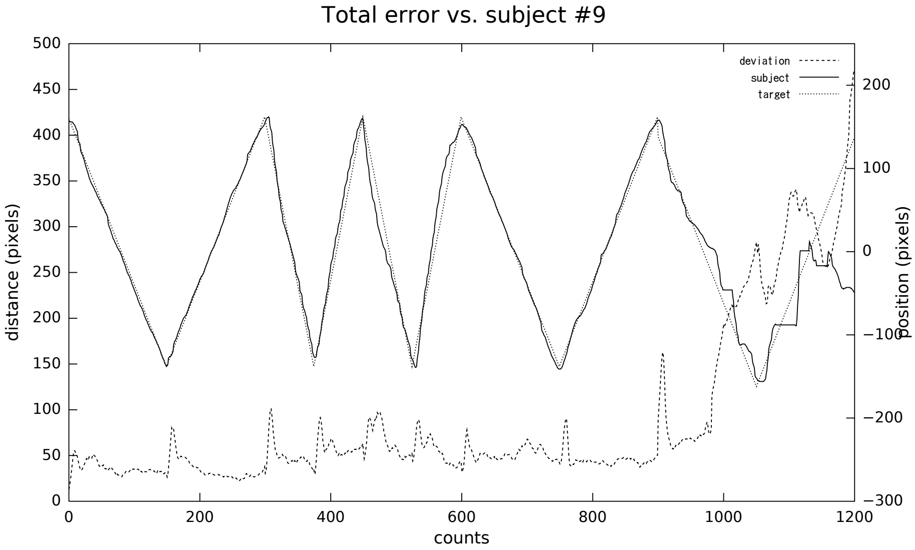
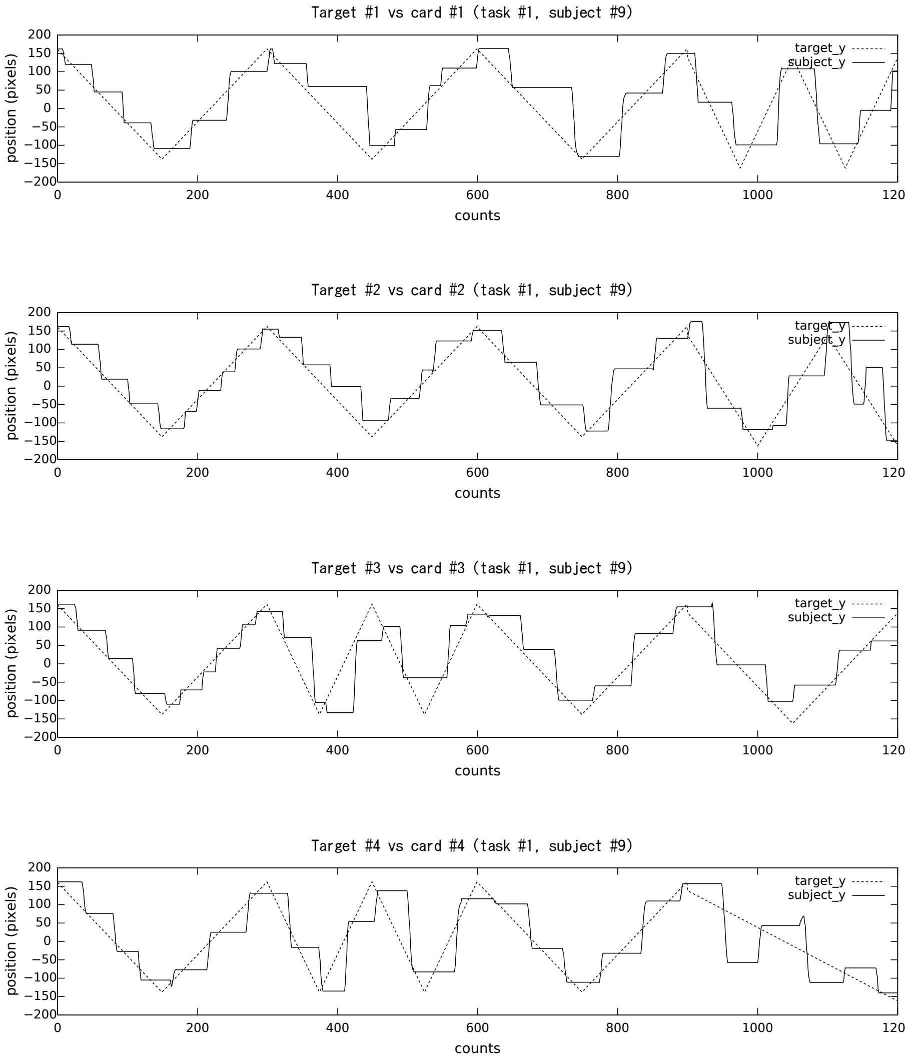


Figure 8.25: Comparison of the average error with the motion log of the card and the target

sion, so that the accuracy was maintained high. The four-fold improvement of concurrent manipulation of the four cards in the total error indicates this. On the other hand, in phase 4, the targets moved in a disorderly fashion, making concurrent manipulation difficult and causing the total error to become worse. The error increased sharply in phase 4. The reason for this is that the subjects were trying to manipulate the cards awkwardly, which increased unnecessary manipulation. In addition, because the targets moved independently, the subjects had to track the targets by eye with precision. However, the hands covered the input surface, and it was difficult to see the targets. This is confirmed by the results for the single-point input. In these results, the error increased slightly in phase 4, indicating that the subjects were not so confused. Figure 8.26 represents the Y axis motions of the cards and their targets from the results of a subject who manipulated the cards with a finger. It is confirmed that the cards were moved toward the targets intermittently. On the other hand, Figure 8.27 shows the results for the same subject manipulating the cards with multiple fingers. As shown in the figure, some cards were not manipulated during phase 4, and some cards were moved in the wrong direction. This indicates that the subject was confused, which caused problem with respect to the manipulation. In addition, while two cards were manipulated by two fingers of one hand, when the cards were distant from each other, the system failed to track the fingers. Usually, the angle between the finger and the input surface is narrow. However, when the cards were distant, the subject split his fingers and the body side of the fingers touched the input surface with a wide angle. In such cases, the contact surface of the finger became small and was filtered out, and the system recognized that the finger was released.

In Figure 8.24, the score of the single-point input sometimes increased, while the score of the multipoint input decreased. In some cases, the score of the single-point input was better than that of the multipoint input. This happened when the targets changed directions at the end of the screen, as shown in Figure 8.25. The reason for the increase in the single-point error is that when a card remains still, its target moves away from the card at first but soon turns and approaches the card, and the total error becomes low. On the other hand, when a subject is manipulating the cards with his fingers, he cannot catch up to the targets when they turn and the total error increases sharply. In this experiment, the targets of the components move independently of the user's intention. This causes a tracking error, but in real applications the target of the component is set by the user and so this error does not occur.

From the results obtained for the supplemental experiment, the total error of the multipoint tracking was 10 times greater than the error of single-point tracking. The reason why the ratio was larger than four, which is the number the cars, was not clarified by this experiment. However, these results indicate that the subjects distributed their attention to multiple targets, which caused the error. A typical example is shown in Figure 8.28. As shown in the motion on the X axis, in phase 3, two cards on the left hand (#1 ,#2) moved linearly and were similar to the motion of their targets, but the motion of the two cards of the right hand (#3, #4) were rounded. Moreover, in Figure 8.27, at the beginning of phase 2, the speed of the two right-hand targets increased, but both the two right-hand cards and the two left-hand cards were moved faster. These examples indicate that the subject paid attention to only one side, causing the error of the other side to increase.



Target #1 vs card #1 (task #1, subject #9)

Target #2 vs card #2 (task #1, subject #9)

Target #3 vs card #3 (task #1, subject #9)

Target #4 vs card #4 (task #1, subject #9)

Figure 8.26: Comparison of motions: Single pointing vs. targets (subject #9)

The coordinates indicate the relative position from the center of the average of the maximum and minimum values.

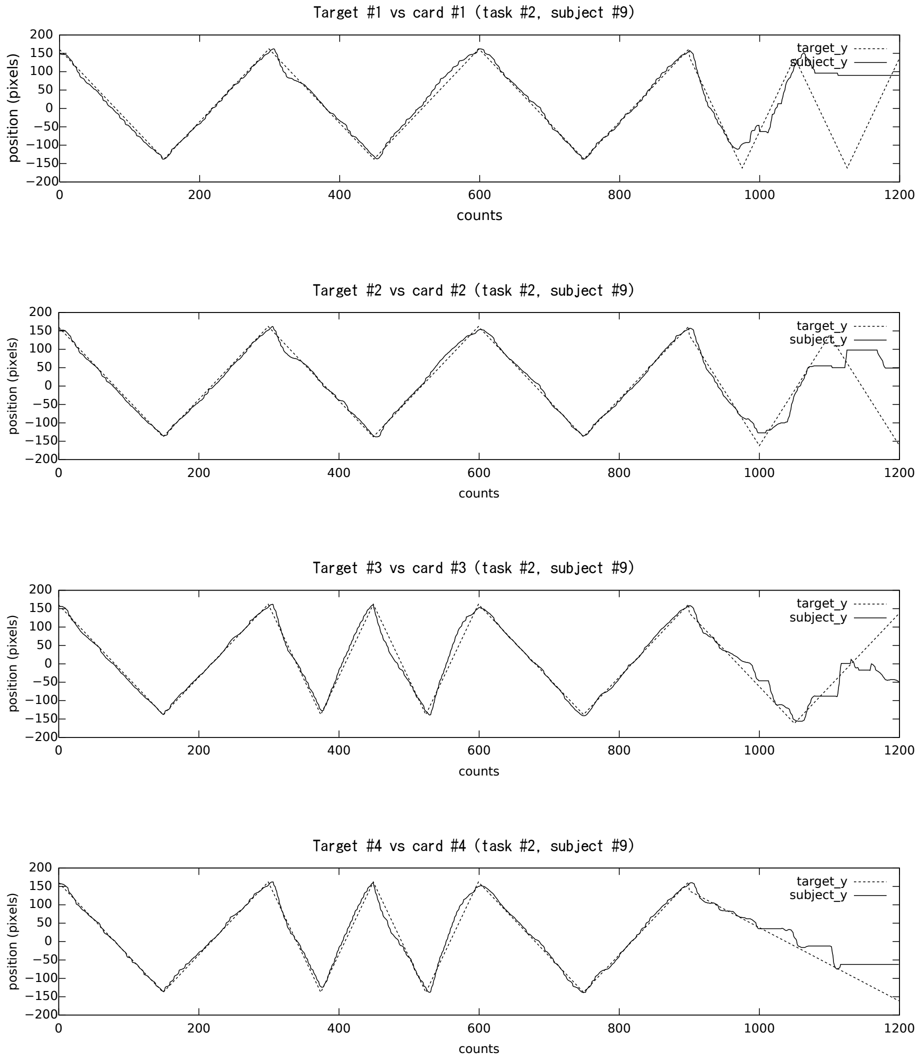
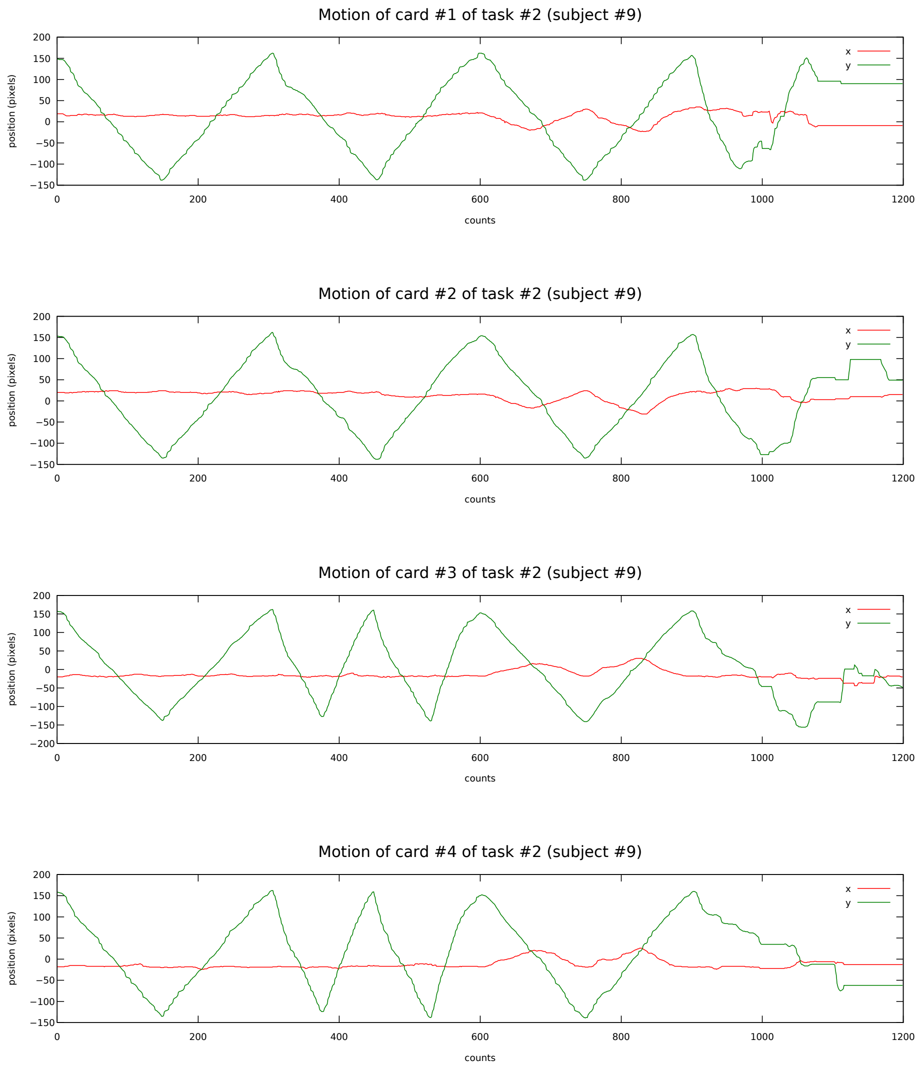


Figure 8.27: Comparison of motions: multiple pointing vs. targets (subject #9)

The coordinates indicate the relative position from the center of the average of the maximum and minimum values.



Motion of card #1 of task #2 (subject #9)

Motion of card #2 of task #2 (subject #9)

Motion of card #3 of task #2 (subject #9)

Motion of card #4 of task #2 (subject #9)

Figure 8.28: Motion of objects under multiple pointing input (subject #9)

The coordinates indicate the relative position from the center of the average of the maximum and minimum values.

8.7 Discussion

The experiments indicated that concurrent manipulation was efficient.

First, based on the results of the card sorting test (Section 8.6.2), when the components were to moved in independent directions, concurrent manual manipulation was not efficient. This conclusion is based on the results of the tracking test (Section 8.6.3). In contrast, concurrent manipulation of multipoint input improves the performance when the components are to be moved in the same direction. In addition, when the components should be moved continuously, concurrent manipulation is efficient. Moreover, in the experiment, the error on the multipoint input was observed to increase when the direction of the targets changed and the subject missed tracking, or when the motion of the right-hand side affected that of the left-hand side. The former problem does not occur in real applications because the target of the component is set by the user himself. A test that allows the subject to control the components at will should be conducted.

In the stability test described in Section 8.6.1, we observed a blur in a recognized position when the finger is released. This type of error has been reported for touch panels and pen tablets. Two techniques were proposed to avoid this problem: drop a number of frames at release, or introduce traction or inertia to the components to resist the blur. However, both techniques cause additional latency in the manipulation. In particular, since the scan rate of SmartSkin is low, if the number of latency frames increases, the response time increases significantly. The key issue is the improvement of the scan rate.

When the angle between a finger and the input surface is wide, the system fails to track the finger. In such cases, the system recognizes that the finger is released from the surface. Moreover, some subjects did not notice that the tracking of the finger was aborted and continued the manipulation, to find only later that the corresponding component was not moved as expected. In order to move the component, in some cases, the subject had to release the other components. In summary, in addition to the limitation of finger tracking, the user is not able to realize immediately that the component was released from the finger. On the other hand, the input system with physical devices in Chapter 7 has a robust tracking mechanism and the restriction of the positional relationship between the finger and the device is low. Moreover, a device gives a physical feedback to the user so that the user can quickly realize when the device is released. In order to improve the multipoint manipulation of SmartSkin, the advantages of physical devices should be introduced to the finger input. In the finger detection process, a fingertip is currently distinguished by the size of the contact surface, but if the resolution and the range of the sensor are improved, the system would be able to distinguish them by the shape of the contact surface and the distance between the sensor and the observed object. In addition, tactile feedback can be added to the input surface by placing a vibrator (a piezoelectric vibrator would be sufficient) on the surface to feedback to the fingertip directly.

In the Tangram test, the subjects were confused by the difference in behavior between the real Tangram pieces and the application software on multipoint input. This indicates that if the goal of an application is to simulate a real system, it must simulate its dynamics and behavior as closely as possible. For the Tangram application, a constraint solver or dynamics simulation could be implemented to simulate conflicts between pieces. On the other hand, the present application allows the pieces to be moved freely without any physical conflicts. This is an advantage of the software application. The answer to the question "Which is better?" may depend on the application. Interestingly, however, when the subjects manipulated piece with their fingers, most of the subjects did not try to use the physical behavior of pieces. Usually, graphical components on a computer screen are not assumed to behave as they would if they were real objects. However, when the subjects used a multipoint input, they tried (and failed) to manipulate pieces using conflicts. The reason why the subjects changed their mind when the system allowed multipoint input was that the system provided an intuitive interaction, and the subjects then assumed that the system allowed any manipulations that are possible in the real world. This indicates that an application on a multipoint input system should accept various manipulations on physical behavior, or clearly inform the user as to which manipulations are not allowed. The design of components may be important.

8.8 Conclusion

A finger-pointing multipoint input system was developed. From the results of evaluation tests, the system was confirmed to have achieved stable finger-pointing, and the multipoint input was confirmed to have performed effectively for continuous concurrent manipulation. On the other hand, some disadvantages of finger-pointing multipoint input over physical devices-based multipoint input were also discovered. First, when a finger is released from the input surface, the system recognizes a blur in the recognized position. Second, the system fails to track a finger when it touches the surface in an unusual way. A number of techniques were proposed to avoid the first problem, namely, dropping a number of frames at the time of release, or introducing traction and inertia to components. For the second problem, the improvement of resolution is required in order to acquire a fine image of the contact surface.

Bulldozer manipulation was introduced for concurrent manipulation on SmartSkin.

Two implementations of the bulldozer manipulation were developed: one is based on creating a potential field from the sensor values and the other employs an optical-flow analysis. They have not yet been evaluated, but the differences between these two implementations were examined.

An input system on a human body sensor, which does not use physical devices, is suitable for concurrent manipulation. Concurrent manipulation with a multipoint input system performs efficiently when the user needs to move the components in the same direction or when the components are close to each other. Therefore, this system is suitable for applications that involve few tasks but that include many components. On the other hand, bulldozer manipulation is suitable for applications that include many subjects. Consequently, the proposed system, which implements both multipoint input and bulldozer manipulation, is effective for various applications.

Chapter 9

Prototype 3: Laser Pointer Tracking System

9.1 Overview

The prototype systems described in the previous chapters are designed for desktop use. In this chapter, a third prototype, designed for a large screen, such as 10 m×10 m, is described. Previously, a computer graphics technique was developed for improvised visual performance based on real-time video effects[16][15], and the present prototype is designed to create improvised computer graphics by multi-pointing.

9.1.1 Background

Interactive techniques to improvise live visual images are an important challenge with respect to stage performance. For example, recent live performances of musicians or DJs often employ background visuals as a part of stage performances.

Computers are commonly used onstage to generate real-time rendered computer graphics for this purpose, and most of the systems are controlled via mice or keyboards. This causes some problems. One of which is that a mouse or a keyboard limit the amount of information flowing from the performer to the computer, as described in Chapter 1. The other problem is that using such interactive devices in front of a large screen on the stage lacks the feeling of direct manipulation by the performer. In addition, it is difficult for the audience to recognize the relationship between the body action of the performer and the result that appears on the screen. Moreover, it is not very entertaining to see a performer manipulating a mouse or tapping a keyboard. The goal of the present research is to clarify the relationship between performer and screen for both the performer and the audience. This goal can be achieved using the proposed laser pointer tracking system.

For live performances, the musician's movement is an integral part of the entertainment. The audience can easily recognize the relationship between the movement of the musician and the sound. The musician may even exaggerate his movements for this reason. On the other hand, some contemporary musicians use a laptop computer on a stage, but using a mouse on the stage is not so entertaining. This is one of the most important problems with computer-generated sound performance. Recently, an international computer conference, New Interfaces for Musical Expression (NIME) [1], at which scientists, artists and engineers gather and discuss such problems, has been held annually.

9.1.2 Input system with laser pointers

A multipoint input system has been developed that uses laser pointers to construct a live visual performance tool, solving the problem described in the previous section.

By using a laser pointer, the performer can point to a position on a screen remotely, which gives the feeling of direct manipulation. In addition, by using a smoke machine, the beam of the laser pointer and the manipulation of the performer can be better visualized. This clarifies the relationship between the performer's action and the auditory result to the audience.

A video camera was used to track laser spots on the screen. There have been a few related studies using a laser pointer as a pointing device, but the previous systems are not suitable for use in stage performance for reasons described in the next section. This prototype solved the above-mentioned problems and has been used in a number of performances.

9.2 Related works

Kirstein and Müller presented a simple framework of laser pointer tracking [28]. The system uses a video camera to track a laser spot. The system continually calculates a reference image by averaging a certain number of previous frames, and a pattern recognition is applied to the difference between the current frame and the reference image to detect the laser spot.

Sukthankar et al. described the application of a laser pointing system in a computer-based presentation application[53]. The system assumes that the environmental light is sufficiently dark, and uses the image of a presentation slide as a reference image.

Olsen and Nielsen presented a laser pointer-based input system that can recognize the gesture motion of the laser pointer[41]. They reported that in order to avoid false

[1] http://www.nime.org/

off signals in laser spot detection, the system sums the on or off signals of the laser pointer for the previous five frames using a camera. This causes a latency of a gesture recognition.

Myers et al. reported the results of an evaluation test of laser pointing[35]. The result of a point-and-click style input by a laser pointer was reported to be insufficient compared with direct screen tapping, the mouse and other hand-held device. Oh and Stuerzlinger also reported the result of evaluation tests based on the ISO 9241 standard[39] and described that the throughput of the laser pointer is below that of the mouse on the ISO-based pointing and selecting interaction.

LumiPoint[7] is an input system developed for multi-user interaction with a wall-sized display. The display consists of four rear-mounted projectors and screens, and four rear-mounted video cameras cover the screens. Each camera captures an image of the screen, and a Kalman filter is used to estimate the pointer path. This system is not able to distinguish between laser pointers. However, Oh and Stuerzlinger introduced a technique to identify different laser pointers that blink in different patterns.

As reported in related studies, a key issue in laser pointer tracking is the detection of laser spots from captured images. In addition, previous studies assumed that each user used one laser pointer.

9.3 Multipoint input with laser pointers

9.3.1 System requirements

An effective laser pointer tracking system for visual performance must satisfy various requirements that differ from requirements for desktop applications or interactive presentation systems.

low latency, high scan rate The visual on the screen must be synchronized to the motion of the performer's action. In order to reduce the latency of input, the scan rate should be high.

high resolution The resolution should be as high as possible. The system should be usable for various purposes, not only for pointing to a GUI widget.

robustness The system must track lasers accurately and robustly during a performance.

no restriction on the visual and the motion The system should be applicable to any type of visual presentation without restrictions on color or image. The system should not restrict the motion of the laser pointer.

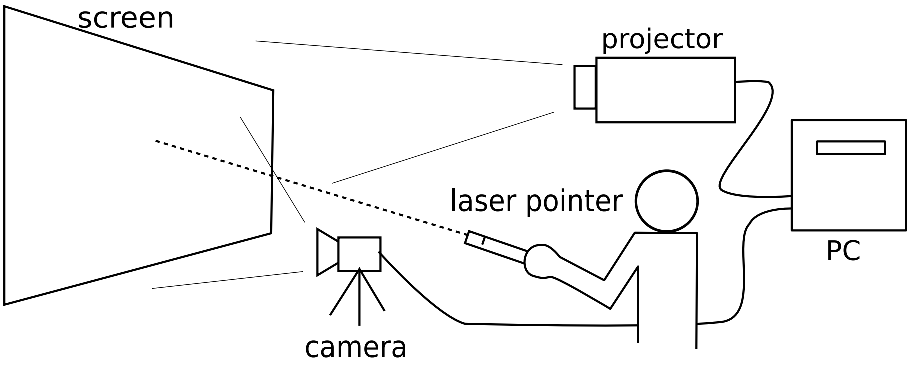


Figure 9.1: System overview of the laser pointer tracker

Because of these requirements, the previous approaches were not adopted. Background subtraction is insufficient when the visual on the screen moves quickly. Color matching techniques are not versatile because the color of the laser spot cannot be used in the visual. Using a visual output such as a reference image is problematic when the frame rate of the visual is higher than the scan rate of the camera. In addition, synchronization of the shutter of the camera to that of the projector is required. Pattern matching techniques are not robust if the laser spots move quickly or if the visual contains a similar figure.

9.3.2 Implementation

Figure 9.1 shows an overview of the proposed system. The basic approach is the same as that of previous systems: the computer is connected to a projector and a camera that observes the screen. An IEEE1394 digital camera (Fire-i, Unibrain S.A.) that can deliver uncompressed 640×480-pixel images at 30 frames per second was used. The projector was a standard 2000–2500 ANSI lumen XGA projector.

As described in the previous section, background subtraction or pattern/color matching techniques cannot be used to detect a laser spot from the camera image. In order to bypass expensive image processing techniques for laser detection, very bright green laser pointers (532 nm wave length, class 3a, 5 mW)[2] were used, and an ND-4 or ND-8 filter was attached, which decreased the power of incoming rays to 1/4 or 1/8 their original power. Using one of these filters, environmental light and the image on the screen could be completely eliminated from the view of the camera because the luminosity contrast is very high[3].

Figure 9.3 shows a typical difference between the two settings. The left-hand figure

[2] GPL-105, Leadlight Technology, Inc.

[3] Instead of the ND filter, a bandpass filter that passed only 532 nm light was used.

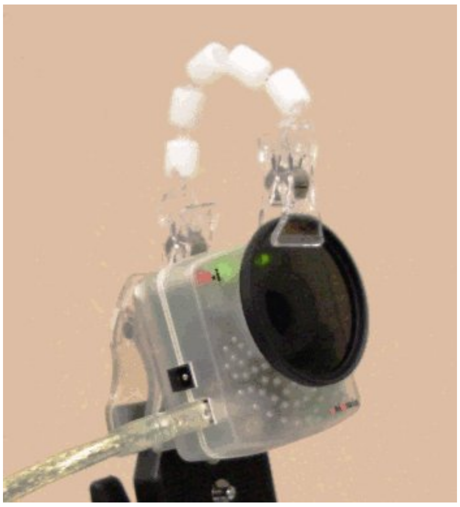


Figure 9.2: IEEE1394 digital camera and ND filter

is an image captured without an ND filter. There is a laser spot in the center of the marked area, but the color of the spot is not so different from its background. In addition, the color of the spot is not green, because the brightness of the laser pointer is so high that a false color problem occurs. The right-hand figure is an image captured with the filter. Only the laser spot is seen in the image and most of the background is cancelled. By using the filter, laser detection becomes easy and the computational cost is reduced.

All of the automatic parameter controls (brightness, white balance, and exposure) of the camera were turned off to avoid unexpected parameter shifts during performance. The exposure was set to approximately 1/30 of a second, which is the same as the scan rate of the camera. This caused the image of the laser spot to moved quickly so as to become a blurred and slightly dimmed trail (Figure 9.4). This is considered to be less suitable for laser spot tracking[39], but using the laser trails is thought to be justified because the laser motion generated by the performer is captured as trails. This approach is discussed in Section 9.4.

The camera provides the images in YUV 4:1:1 format. Only the Y value (luminance) was used for laser tracking. In addition, the image is converted to RGB format because the RGB format is better for image processing, such as blending. This is described in Section 9.4.

9.3.3 Calibration

The proposed laser tracking system can be applied in various situations. In particular, the size of the screen, the distance between the camera and the screen, and the angle

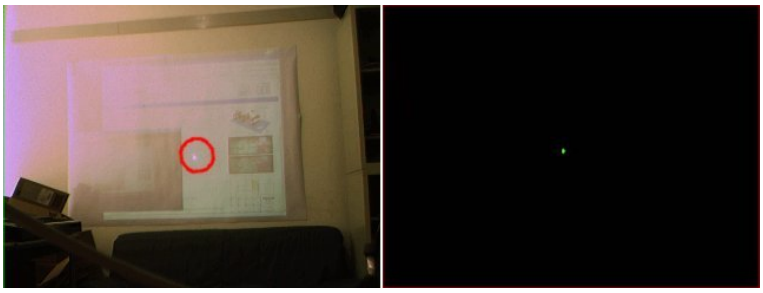


Figure 9.3: Captured images. Left: without an ND filter. Right: with an ND filter.

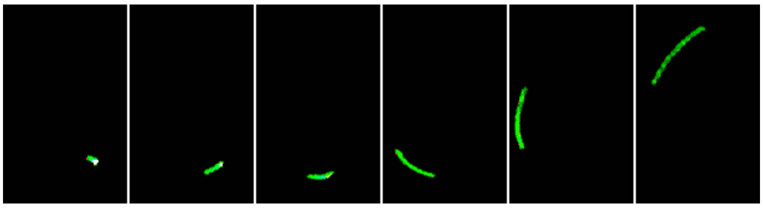


Figure 9.4: Sequential shots of a fast moving laser trail

of the camera can be varied at every stage. As a result, a calibration process is needed before a performance is possible.

This process calibrates the camera to map the detected positions and shapes of laser trails in the camera image to the corresponding positions and shapes on the screen. A simple solution that uses a **perspective transformation** is used.

At the beginning of the calibration process, the user points to four corners of the screen with a laser pointer for one second (30 frames) for every corner. The system measures the centroids of the laser spots in the camera image and chooses a *mode* position from among 30 recognized positions. The reason for choosing the mode is that, generally, the user first points to a position distant from the corner and then moves the laser to the correct position. This error affects the median or mean of the calibration.

After the four corner positions are given, a three-dimensional transformation matrix is obtained by using standard linear algebra techniques. Letting T be the transformation matrix, a point on the camera coordinate system $p = (x, y)$ is transformed to (X, Y) in the screen coordinate system by following equations:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = T \begin{pmatrix} x \\ y \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} \frac{x'}{z'} \\ \frac{y'}{z'} \end{pmatrix}$$

T^{-1} will be used to transform an image from the camera onto the screen coordinate system.

9.3.4 Laser spot tracking

The process of laser spot tracking consists of four steps. First, the system eliminates pixels darker than a Y value threshold. Second, the system scans pixels and divides the image into several segments of consecutive pixels. At this time, segments smaller than 10 pixels are eliminated in order to remove noise pixels around a laser spot [4]. Third, the system calculates the position of each segment and then finally tracks their motions. In this prototype, the centroid of a segment as its position is used.

With this in mind, for the motion tracking process, since the system cannot identify the laser spots, the laser motions must be to estimated from the results of the third step, using the same estimation technique as that used to track the motions of fingers described in Section 8.3.2.

The proposed laser tracking system was tested with three laser pointers, and the system was found to track them sufficiently in real time. The system has not yet been tested with four or more lasers, but based on the results of the finger tracking, it is thought that approximately ten lasers may be tracked in real time.

However, unlike the case of finger tracking, this technique has a problem in the case of laser tracking. First, two or more laser spots can cross over each other while fingers cannot. In such a case, the system cannot track them correctly. Moreover, even if the laser spots do not cross each other, when the distance between the spots is too short, tracking may fail.

9.4 Using laser trails

A multipoint input with laser pointers was described in the previous sections. Here, a concurrent manipulation that which uses the shape of a laser trail is introduced.

9.4.1 Accuracy of multipoint input

Most of the previous systems use the centroid of a laser spot to estimate its position and track its motion. This strategy was also employed in the present multipoint input system. In several cases, it is sufficient to track the motion of the laser for a presentation or a desktop application. However, a laser trail can become large and skewed when the laser pointer is moved quickly.

[4] Although it is necessary to change the threshold according to the distance from the camera to the screen, approximately 10 pixels is generally suitable.

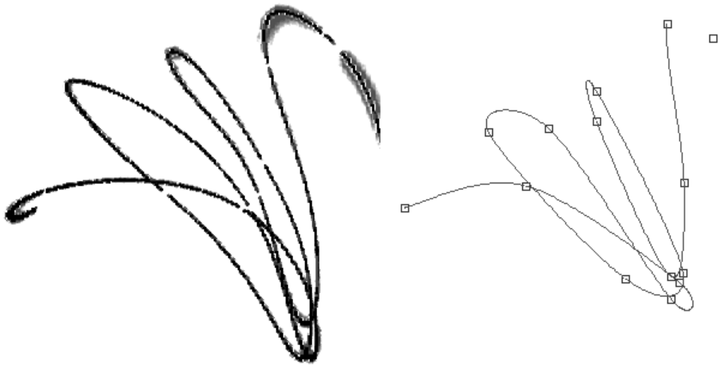


Figure 9.5: Left: image-based drawing. Right: an interpolated curve.

Figure 9.4 shows the difference between the actual motion and the estimated motion of a laser pointer[5]. The left-hand figure shows a composite image of laser trails. In the right-hand figure, each small rectangle represents the centroid of a laser trail, and the curve is an interpolated motion using a Catmull-Rom spline function on the centroids. Their shapes are very different.

As shown in the figure, when a laser pointer is moved quickly, its trail becomes larger and the accuracy of position estimation becomes worse. The distance between centroids becomes longer, and the estimated motion becomes different from the actual motion.

In conclusion, a set of the centroids of the trails provides a rough estimation of the motion of the laser pointer, which is sometimes undesirable for visual performances. In addition, a motion estimation process using an interpolation function, such as a spline curve, causes an additional delay. In order to calculate the parameters of a control point, the positions of the neighboring control points should be known. This means that the parameters of the latest control point cannot be calculated, which causes a one-frame delay. This does not satisfy the present requirements. (See Section 9.3.1.)

The proposed approach to solve this problem uses a bitmap image of the laser trail. If tracking positions are not needed by the application, the use of a bitmap has various advantages over position tracking. First, the use of a bitmap allows the entire information of a laser trail to be used, some of which is omitted by the position tracking process. Second, delays due to the position tracking process can be avoided.

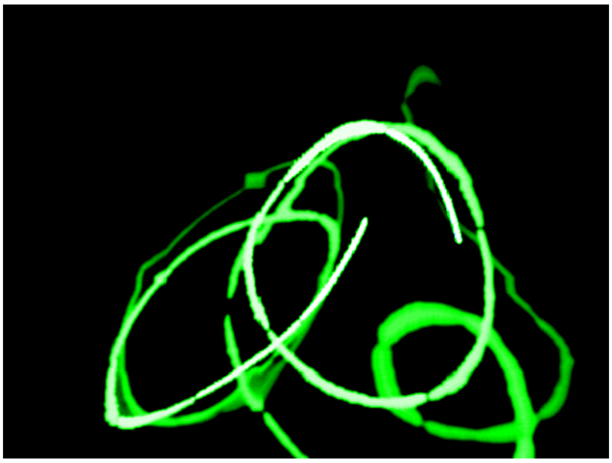


Figure 9.6: An example of drawing application based on laser trail input

9.4.2 Trail-based drawing

As we described in Section 9.3.2, the proposed approach allows us to obtain a bitmap image of a laser trail. In addition, the image is transformed into screen coordinates so that the user can paint the screen by a laser pointer directly. This is sufficient for a painting application. In order to obtain a whole stroke of the laser pointer, the system converts the image from YUV to RGB, and then eliminates dark pixels to reduce the noise. Finally, the system composes all of the bitmap images of the trails by adding RGB values of pixels with saturation. In order to obtain a consequent stroke, the shutter speed of the camera is set to 1/30 of a second, which means that the shutter is kept open during performance. An example of the composed stroke is shown in the left of Figure 9.5. Figure 9.6 shows another example of a drawing application. In this application, the luminance of each pixel is reduced in every frame, which causes a fading of the strokes. As shown in this example, this trail-based drawing technique can be applied to existing real-time image processing techniques and enables a novel visual presentation.

9.4.3 Using laser trails for a GUI

The technique using laser trails described above does not provide position information. However, conventional GUI widgets and interaction techniques (e.g., buttons or sliders, click or drag & drop mechanisms) require a position-based input system. To extend the trail based approach to the conventional GUI, an experimental GUI button was implemented for bitmap image-based interaction.

In Figure 9.7, the rectangle and circular frames represent buttons, and the solid

[5] This figure represents 13 frames of motion, which corresponds to 13/30 of a second.

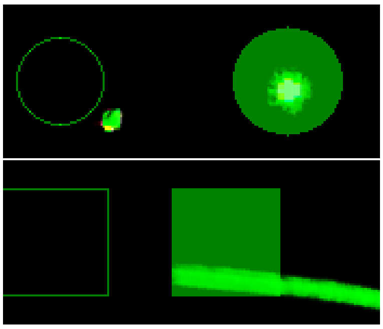


Figure 9.7: Button widgets for bitmap image-based interaction.

rectangle and circle represent *pressed* buttons. When a trail image is captured, every button on the screen scans the region inside its shape (rectangle or circle) and counts the number of pixels of laser trails. If this number is not zero, the button is pressed, otherwise the button is *released*. The button does not lose touch with the laser trail, even if the laser moves quickly, as shown in the figure. At this time, if a button is touched with two or more laser pointers simultaneously, it is not able to detect the laser pointers, but there is a chance of detect them by counting the increased number of pixels.

This approach enables the control of GUI widgets as if they were painted by a brush. By applying the techniques of Don't Click, Paint![3] (see Section 5.4.1), it is possible to control multiple components in real-time.

9.5 Discussion

This interaction technique has not been evaluated precisely, but the laser tracking system has been used for visual performances and the impact of this approach on performance was observed.

This approach satisfied the present design goal in presenting a clear relationship between the actions of the performer and the visual result. In particular, in clubs, the room is filled with smoky air, but the beams of the laser pointers were clearly visible, and the audience understood that the visuals on the screen were controlled directly by the lasers. Some members of the audience were given laser pointers to use, enabling them to interact directly with the screen.

Both position tracking-based and image-based applications were used. These two approaches have different advantages. Position-based interaction is good for making geometrical graphics or manipulating graphical objects on a screen by laser interaction, whereas the image-based approach produced an 'action painting-like' performance. In a performance, delayed visual presentation should be avoided at all costs. The proposed system consistently causes a 1/30 of a second delay because of the camera interface. If there is an additional one frame delay, the total delay is approximately 0.1 seconds, which is not acceptable for musical performances.

When the distance between the performer and the screen is great, movements by the performer may cause a great deal of blurring on the screen. This blurring has been reported in previous studies, and a number of studies proposed the use of a Kalman filter as a solution. However, this solution cannot be applied to the proposed painting application based on laser trail capture.

The present implementation requires a calibration process before a performance by using a laser pointer to explicitly point to the corners of the screen. A number of previous studies employed an automatic calibration by projecting a test pattern onto the screen and calibrating the parameters according to the pattern. In contrast, the proposed approach is not automated and requires some time. The use of a test pattern is not applicable to the proposed system because an ND filter is mounted on the camera and the pattern cannot be captured. However, it is not acceptable to remove the filter before the calibration process because this may affect the position or direction of the camera. Therefore, the use of a camera that has a variable aperture instead an ND filter, or the addition of small guide lights beside the screen, is proposed for calibration.

9.6 Conclusion

A multipoint input system using laser pointers was introduced. In contrast to the previous laser tracking system, the advantages of the proposed system are low latency, high scan rate, robust tracking and few restrictions on the visual on the screen. A novel input technique using laser trails, which is introduced precise tracking of the motion of laser pointers, was proposed. This approach is useful for visual performance.

On the other hand, as a multipoint input system, the proposed system has a few problems. The system fails to track the motion when two or more laser trails cross over each other. In addition, the performer can manually manipulate two laser pointers, but this is not sufficient to simultaneously control a number of components.

Chapter 10

Applications

In this chapter, we describe three applications based on the concurrent manipulation techniques and prototype systems described in the previous chapters.

10.1 Marble Market

10.1.1 Overview

A video game application, Marble Market, was developed using the non-pointing input system described in Section 8.5. Marble Market uses a table-sized SmartSkin to track the motion of the players' arms. Two to four players can play the game, and the players control characters displayed on the table using their arms.

10.1.2 System architecture

Figure 10.1 shows the Marble Market game and players. The sensor is a table-sized SmartSkin, described in Chapter 8, that is installed on a table. A projector is located above the table, and the projector displays the game screen onto the tabletop. Players stand around the table and touch it with their arms.

10.1.3 Game rules

The game screen is shown in Figure 10.2. The goal of this game is to bring as many small balls, called *Marbles*, as possible to the player's *Basket*. A *Marble Machine* produces marbles continuously. The Marble Machine sometimes produces large red balls called *Crushers*. Crushers break Marbles. When into a Basket, the Basket looses a quarter of the Marbles already collected. Therefore, players should not collect Crushers, but rather send them toward his opponents' Basket. Marble Machines sometimes provide *Gold Marbles*, which have the value of 50 Marbles, or *Special Marbles*, which bring about some special event.

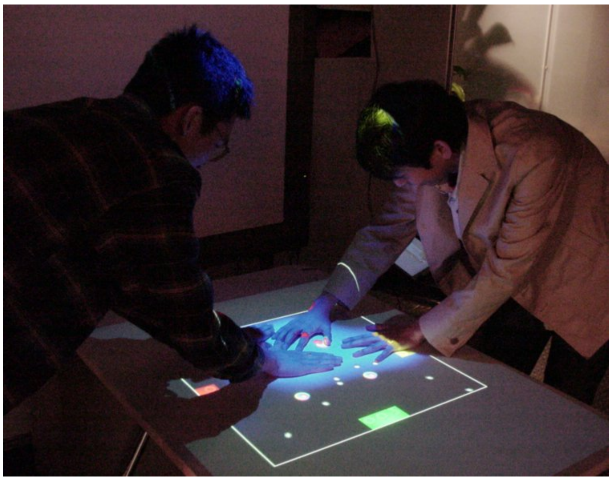


Figure 10.1: Marble Market

The player who collected the most Marbles in two minutes wins the game.

10.1.4 Bulldozer operation in Marble Market

In the game screen, many Marbles are rolling simultaneously in real time, and the players have to control these Marbles concurrently. For concurrent manipulation, bulldozer manipulation was employed based on the potential field technique described in Section 8.5.1. That is, the system forms a potential field from the value of the SmartSkin sensor. Marbles and Crushers are forced by this potential field toward the lower potential. By controlling the potential field, players can move multiple Marbles concurrently. There is no friction between the field and a Marble, but the speed of the Marbles is limited so that players can follow them with their eyes.

As described in Section 8.5.1, there are two methods by which to create a potential field from sensor values: increase the potential when a player's hand approaches the table ((a) in Figure 8.14), or decrease the potential ((b) in Figure 8.14). In the former method, Marbles move away from the hand, and in the later method, Marbles are drawn toward the hand. The later method may be more intuitive because the potential escalates according to the distance between the hand and the table. However, after implementing and comparing these two methods, the former method was found to provide a better gaming experience. The differences between the two methods are discussed in the next section.

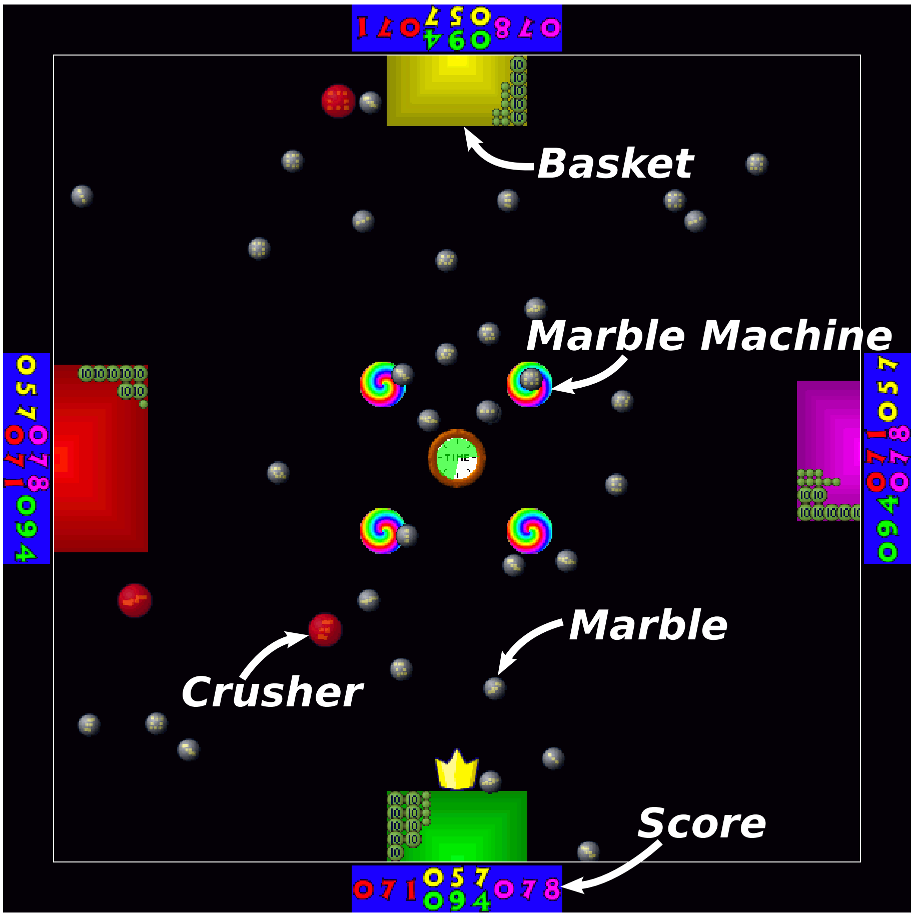


Figure 10.2: Marble Market: an illustration of the game field

10.1.5 Concurrent manipulation of many Marbles

Marble Market was exhibited at Sony ExploraScience in Tokyo. In addition, Marble Market was exhibited in a number of open laboratory events, and several visitors played this video game. At both exhibitions, the basic rules of the game were described and the method of moving the Marbles by hand were described. However, a detailed description of the techniques of concurrent manipulation was not given. Despite this, the players discovered various techniques by themselves. Through these exhibitions, a number of typical techniques of concurrent manipulation of Marbles were observed.

In order to gather Marbles, the player basically places their of both hands on the table and moves slowly toward his Basket. Many players were observed to place their hands beside a group of Marbles and make them roll on the potential field towards their Basket, following the Marbles with their hands to adjust their direction. Some players formed a funnel in front of their Basket so as to guide Marbles into their basket.

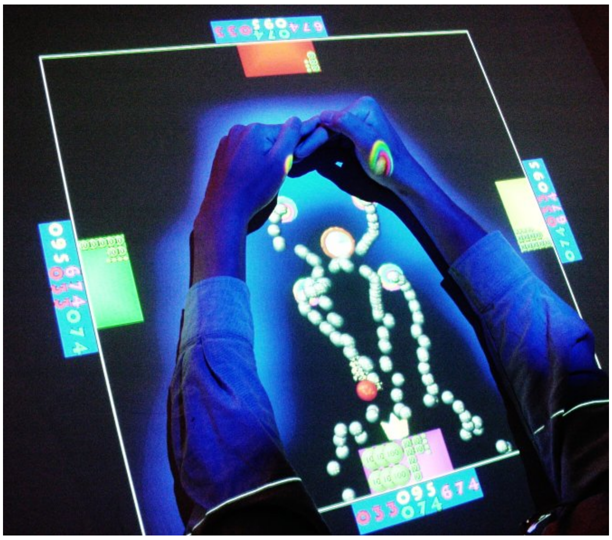


Figure 10.3: Marble Market: gathering marbles by arms

This SmartSkin detected not only hands but also arms. When players discovered this, they gathered Marbles more aggressively. Several of them moved their entire arms in the manner of a bulldozer blade. Some players used both arms. Moreover, some players closed off the Marble Machines and Baskets using their arms, as shown in Figure 10.3. In this way, all of the Marbles from the Marble Machines were guided into their Basket.

These manipulation techniques can be used when a potential increases under a hand. In contrast, when the potential decreases under a hand, the following techniques are effective. One technique is simply placing a hand over a Basket to form a potential hole over it, so as to pull in the Marbles rolling near the Basket. The other is putting a hand on a Marble Machine for a while in order to collect the Marbles in the hand, and then to bring them to a Basket by moving the hand slowly.

As mentioned above, techniques by which to gather Marbles differed between the two different potential field. These methods must be evaluated in order to discuss their advantages, but the former method was employed for this application from the observation of the user tests. The former method enables intuitive manipulation, and allows various manipulation, that avoids monotony and makes the game more exciting.

10.1.6 Discussion

From the observation in the exhibitions, the bulldozer manipulation of Marble Market is easy to learn and effective for concurrent manipulation of many Marbles. As an example of ease of learning, early elementary school children soon learned to control the Marbles using their arms.

On the other hand, in some cases, the behavior of the Marbles was not so intuitive for the players. First, most players moved their hands from the Marble Machines to the Baskets too quickly at first. When the hand is moved quickly, it overtakes the Marbles and the Marbles are accelerated to the opposite side, as described in Section 8.5.1. In the current implementation, the hand should be moved at the same speed as the Marbles, or placed as shown in Figure 10.3. This indicates that the potential field method limits bulldozer manipulation. Second, some players pushed the table top surface strongly to block Crushers from entering their Baskets. Since SmartSkin does not sense pressure, this motion is meaningless. However, this observation suggests the use of a pressure sensor for a more intuitive game interface.

10.1.7 Conclusion

A video game application, Marble Market, was developed that allows bulldozer manipulation to control several characters concurrently. The application uses a SmartSkin sensor to create a potential field from the position and shape data of players' arms and hands, and then the game characters are forced from the field. Marble Market has been exhibited, and various techniques were observed for concurrent manipulation of the characters. From these observations, the bulldozer manipulation is judged to be easy to learn and effective for concurrent manipulation of multiple components.

10.2 Multi-track Scratch Player

A musical instrument application was developed with a pair of tablet-sized SmartSkins. This application is a type of software turntable systems for scratching, a DJ or turntablist technique. The conventional turntable systems consists of two turntables (record players) and an audio mixer, but the proposed system requires a computer and a tablet-sized SmartSkin, so that it is smaller and portable. Moreover, the proposed system enables various novel scratching techniques that are difficult or impossible to perform on existing systems.

10.2.1 Background

Scratching on a turntable is a popular sound generation technique in today's music scene. Scratching usually involves the use of two turntables and an audio mixer for scratching, and a player (known as *DJ* or *turntablist*) manipulates them with his hands (Figure 10.4). A scratching sound is generated by rotating a record on the turntable back and forth by hand, while the other hand moves a cross fader or volume slider.

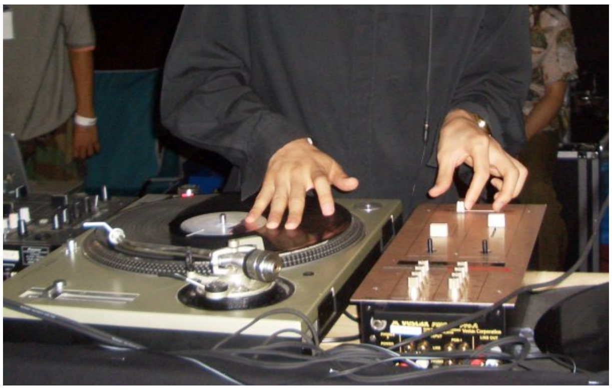


Figure 10.4: Scratching using a turntable

Therefore, the records cannot be changed during a scratching motion and scratching two or more records is not possible. This limitation is considered to be is an important problem in scratching.

A simple solution to this problem is the use of more turntables, but turntables are too large (at least 12 inches square) to manipulate multiple turntables by hand. There are a number of software-based scratching applications, but these employ conventional pointing devices and do not allow scratching of multiple sound tracks.

Audiopad[45] allows multitrack audio mixing by Phicons on a table with a sensor (see Section 5.1.8). By moving Phicons, the user can control the volumes of sound tracks concurrently, but scratching is not possible.

KORG KAOSS PAD[30] is a commercial product that allows scratching by moving a finger on an embedded touch sensor. The sensor does not detect multiple touches so that it is not possible to scratch multiple sound tracks.

10.2.2 System architecture

This system uses a pair of tablet-sized SmartSkin sensors to create a larger input surface that is 47 cm×31 cm. The resolution of the input surface is 1504×992 pixels. Figure 10.5 shows an overview of the system.

10.2.3 Interface

Figure 10.6 shows a screenshot of the screen that is overlaid on the input surface, as shown in Figure 10.5. Current implementation enables five sound tracks to be played, and the sound wave from each track is shown horizontally. Each sound track is the same length as the PCM wave data.

When a track is not touched by a finger, the track is never played. When it is

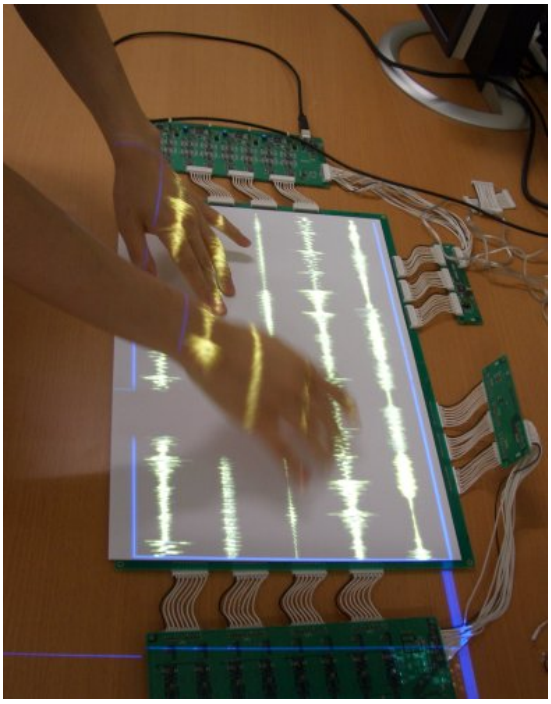


Figure 10.5: Multi-track Scratch Player

touched by a finger, the application plays the waveform that the finger passes over, as if the finger is a needle and the sound track is a groove of a vinyl record. The vertical motion of the finger controls the volume of the sound.

This manipulation can be performed in parallel on all sound tracks, and the result is mixed down to the computer's audio output. Even when a sound track is touched by two or more fingers, it acts in the same manner.

10.2.4 Audio processing

The motion of a finger is used to make a scratching sound. When a new position of a finger is acquired, the application plays the waveform that the finger passes over within the interval of scanning. In order to play the waveform within a certain time, the waveform data must be resampled. At present, the application resamples the data discretely so that it degrades the quality of the sound. Several techniques by which to avoid quality loss are known, such as using Fourier transformation for frequency analysis. However, these techniques were not tested.

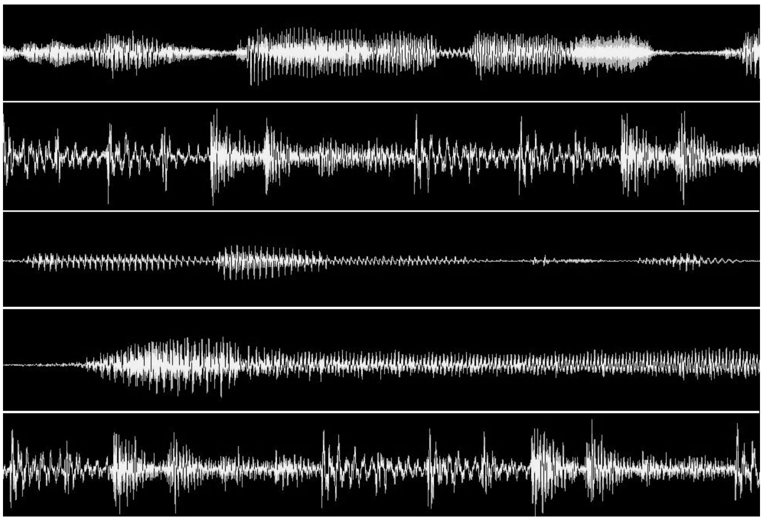


Figure 10.6: Screenshot of the Multi-track Scratch Player

10.2.5 Simulation of known scratch techniques

There are several scratching techniques for conventional turntables and audio mixers. A scratching technique consists of a pattern of motion of a turntable and a cross fader. DJs and turntablists use these techniques sequentially in a performance.

The proposed scratching system is designed to allow novel scratching techniques that cannot be performed on conventional systems, while enabling known scratching techniques.

Figure 10.7 shows two major scratching techniques on a turntable and the corresponding procedures for the proposed system. The technique shown in the left of the figure is known as "baby scratch", the simplest type of scratching, in which the player rotates the turntable back and forth repeatedly. This scratching can be performed by repeatedly moving a finger on a track horizontally. The right side of the figure shows the two-click flare technique, which requires concurrent manipulation of the turntable and cross fader. The player rotates the turntable back and forth using one hand, while the other hand quickly slides the cross fader between min and max twice. By this fading motion, the player can cut the sound to make beats. In order to simulate this scratching technique in the proposed system, the player must release the finger from the input surface twice while moving his finger repeatedly. However, it is difficult to perform this motion quickly. To perform a rapid click, the player can play the first click by one hand then the second click by the other hand.

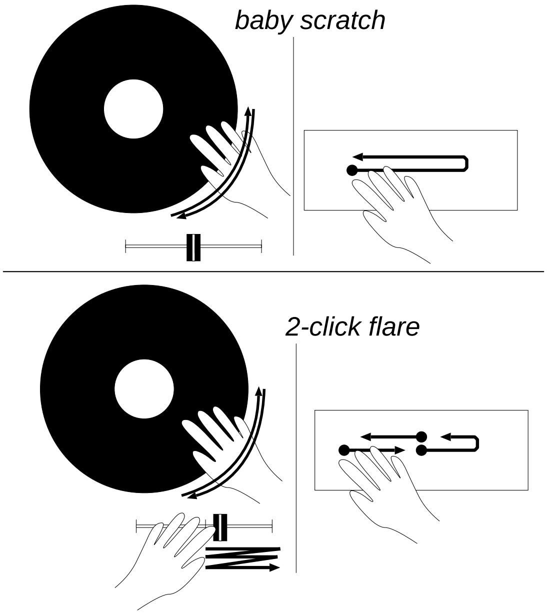


Figure 10.7: Scratching techniques on the Multi-track Scratch Player

10.2.6 Discussion

This system was tested and the delay of the sound was found to be a significant problem. The SmartSkin sensor scans at 20 Hz, but because the application plays a sound track between current and previous positions of a finger, a 50-milliseconds playback delay is generated. In music performance, this delay is unacceptable.

Generally, a turntable rotates at a constant speed when a player does not touch it. The speed can be controlled by a slide volume. Moreover, it can be slowed down by touching the surface of the record softly, or increased by pushing it in the direction of rotation. At this time, the proposed system does support this kind of manipulation. Some DJs have reported that these manipulations are very important for scratching. Therefore, in order to realize these existing scratching techniques, these manipulations should supported.

Scratching over multiple sound tracks is unprecedented, so that it is difficult to discuss its potential performance. Some DJs commented that the multi-scratching would be welcomed for use in the "no-tricks" style of scratching performance whereby a player performs music by only scratching. Generally, no-tricks play often requires a needle to be moved on a record or records to be changed in order change sound tracks for scratching. During this change, the DJ has to stop the scratching performance. Since the proposed approach allows such pauses to be avoided, it improves the performance of the DJ. In addition, although the DJs reported that they were unable to imagine the performance of multi-scratching, they commented that it would be a worthy challenge. However, no practical tests were conducted for multi-scratching.

10.2.7 Conclusion

We developed a software scratching application with a pair of tablet-sized SmartSkins, which enabled concurrent scratching of multiple sound tracks. At this time, the proposed implementation causes a 50-milliseconds delay, which is unacceptable for musical performance.

The proposed approach was well-received by DJs. Therefore, the scan rate of the multi-touch input device should be improved and the delay should be decreased. Finally, new scratching techniques should be developed for the system.

10.3 Graph editing on a Web Community Browser

10.3.1 Overview

Toyoda et al. proposed a clustering technique of Web pages to create a *web community chart*[57]. A web community is a collection of web pages created by individuals or associations that have a common interest on a specific topic. Toyoda's technique is based on a related page algorithm that assigns related pages to a given page using only link analysis. The resulting chart includes web communities and paths between related communities.

Over 10 million web pages in the .jp domain were collected by running a web crawler and extracted a web community chart from them to evaluate this algorithm, but it was difficult to analyze the results by looking at the resulting data. Therefore, the **Web Community Browser** was developed. The **Web Community Browser** visualizes a sub graph of a web community chart and allows the chart to be edited and browsed. This tool was built using a standard GUI toolkit and a desktop UI system (X Windows System and GTK+). The user manipulates a chart in the application using a mouse. However, this interface is not sufficient for manipulating a large chart that contains over 100 communities.

To improve the interface of Web Community Browser, a concurrent manipulation support with SmartSkin was added. This new interface allows concurrent manipulation of the chart by multipoint input and bulldozer manipulation.

10.3.2 Web Community Chart

In this section, Toyoda's algorithm is described briefly.

First, a web crawler collects web pages from the Internet and stores them in a database. It extracts the hyperlink structures from among the pages and divides them into web communities, where each community includes 2 - 1,500 pages. Moreover, the web crawler outputs weighted directed edges between communities that represent the relationships between the communities. This directed graph of web communities is called *Web Community Chart*. In the present study, a web community chart that was based on pages collected in February 2003 was used. The chart consists of 180,000 communities and 1.3 millions edges.

Most web communities calculated by Toyoda's algorithm contain web pages that are created by individuals or other associations having a common interest in a specific topic. Moreover, when groups of web pages have different aspects, such as fan pages of a baseball team or official pages of baseball teams, they are separated into different communities.

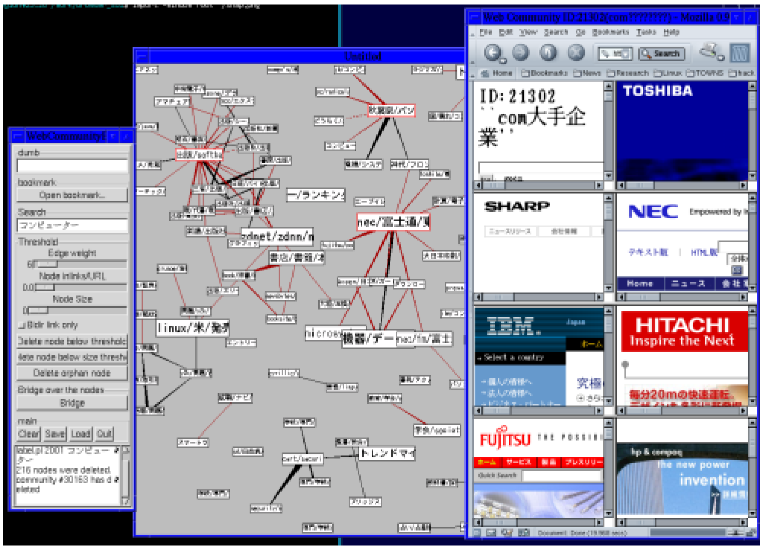
An edge between two communities is a directed edge for the following reason. Suppose there exist community A and community B, and some pages that have links to pages in A or B. If the pages that have links to A often have links to B, while the pages that have links to B rarely have links to A, the algorithm gives a relationship edge from A to B, but not from B to A.

10.3.3 Web Community Browser

The Web Community Browser[17] is a visualization and browsing tool of the Web Community Chart. Using this tool, the user can see part of the Web Community Chart that includes communities that he is interested in and can browse the communities near them or pages inside the communities. This tool was developed to examine the precision of the Web Community Chart and to explore the hidden structure of web communities. Figure 10.8 shows a screenshot of the browser.

The Web Community Chart is a huge directed graph, and its structure is a very complicated hyperlink. Generally, it is difficult to layout and display a hyperlink graph with many links. Moreover, it is impossible to display a graph with over 100 thousands nodes, because of limitations in the resolution of displays. Therefore, the Web Community Browser displays a sub graph of the Web Community Chart that the user has selected, and allows the user to edit the sub graph dynamically, for example by adding related communities or eliminating communities.

Currently, the browser can display 2,000 nodes of the Web Community Chart simultaneously. As shown in Figure 10.9, the browser displays a 1500×1500 pixel window on a wall-sized display and enables browsing of a large graph structure.



コンピューター

label.pl 2001 コンピューター
216 nodes were deleted.
community #30163 has deleted

Figure 10.8: Web Community Browser

Screenshot of a Web Community Browser on a desktop GUI system. In the center of the display, a sub graph of Web Community Chart is visualized. On the left, a console window is shown, and the Web browser on the right shows Web pages that are included in the selected Web Community.

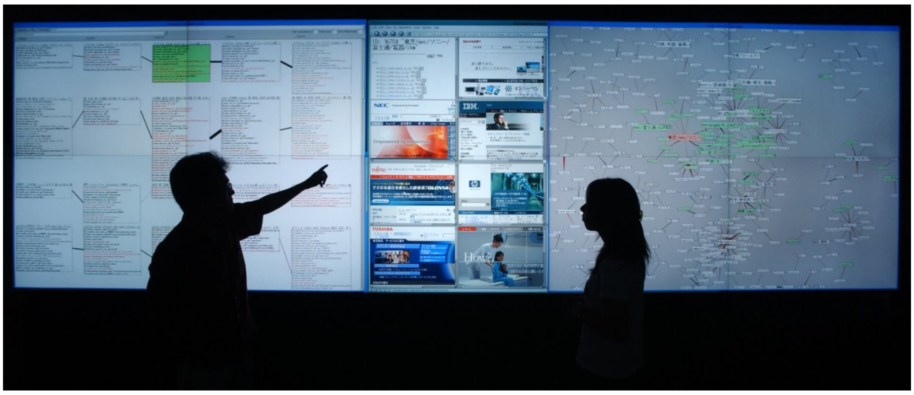


Figure 10.9: Web Community Browser on a display wall system

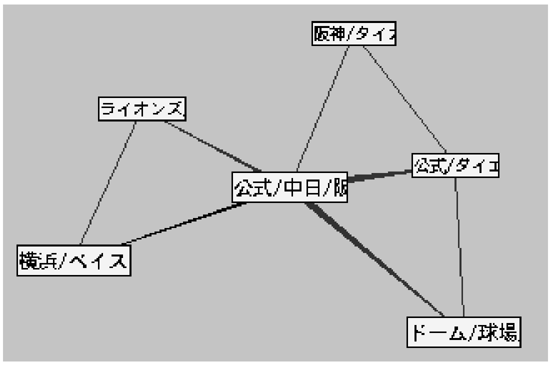


Figure 10.10: Visualization of communities and edges

Visualization of communities and edges

In the browser, each community is displayed as a labeled rectangle (Figure 10.10). In order to emphasize dominant communities, the size of a community is proportional to its score, which is calculated from a number of in-links to the community. The label of a community is generated from keywords in the community, which are collected from link texts to the community (*anchor text*).

In a space crowded with communities, rectangles cover the edges between them, making it difficult to see their structures. In such cases, the browser displays communities as 5-pixels squares, and omit the labels, as described in detail in Section 10.3.3.

An edge is displayed as an isosceles trapezoid. Commonly a directed edge is visualized as an arrow. However, during an analysis of a chart, the focus is often on a *hub* community, which has many in-links. When arrows are displayed around a hub community, the graphic becomes complicated and it is difficult to see the direction of the links. In addition, an edge between community A and community B has two parameters: the weights of a link from A to B and a the weights of a link from B to A. These two weights are unbalanced. In order to visualize this asymmetric relationship, the browser displays edges as isosceles trapezoids (Figure 10.10). The center of a base of an isosceles trapezoid is positioned at the center of community A, and the length of the base is determined by the weight of the link from B to A. Likewise, the other base is positioned at the center of community B, and its length is determined by the weight of the link from A to B. In addition, the browser draws an edge with a different color according to whether the edge has two-way links (bidirectional) or only one link.

The length of an edge is determined by the lower of the scores of the communities at both ends, in order to maintain high-score communities distant from each other to reduce local densities of the graph.

Graph layout

The browser uses Kamada's spring model[26] to layout the displayed graph automatically. That is, all communities exert repulsive force on each other, and each edge acts as a spring to maintain two related communities at a distance of the length of the edge. This layout optimization process runs constantly during runtime, and every step of this process is displayed in the window. In addition, the speed of each community is limited to avoid rapid change of the graph layout by editing the graph by the user, so that the user can follow the change with his eyes.

Graph editing

The user can edit the graph to obtain a desired graph or optimize the graph layout manually. The technique of Henry et al.[21] was referenced for this dynamic graph editing. To edit a graph, the user can eliminate communities having scores that are below a threshold or that have no edges. The user can also temporarily eliminate edges by a threshold. To change the layout of the graph, the user can move communities by mouse. In addition, a community can be secured, or 'stuck', so that it is not moved by repulsive force or tension.

In order to browse the structure of a graph clearly, often it is necessary to stick some hub communities separately to stretch the graph. For example, Figure 10.11 shows a graph of communities related to the stock market. This graph is obtained by using keyword search with the word 'stock' in Japanese, and then eliminating the isolated communities after thresholding the edges. There are three hub communities: major stock brokers, securities dealers associations, and stock exchanges.

These hub communities were stuck manually to stretch the graph. As Shown in the figure, by sticking hub communities and separating them, the structure of the graph becomes clearer.

Fisheye view

When a graph that includes many communities and edges is displayed, it is difficult to see the structure of the graph because the labels of the communities occlude the edges. This can be avoided by omitting the labels. However, when the browser omits all of the labels and allows a label to be seen only by pointing to it, it becomes difficult to determine what types of communities are present.

In order to solve this problem, a fisheye view[18]-based graph drawing technique was introduced. That is, the browser displays the labels of communities that are related to a community on which the user is focused. In this technique, the browser calculates

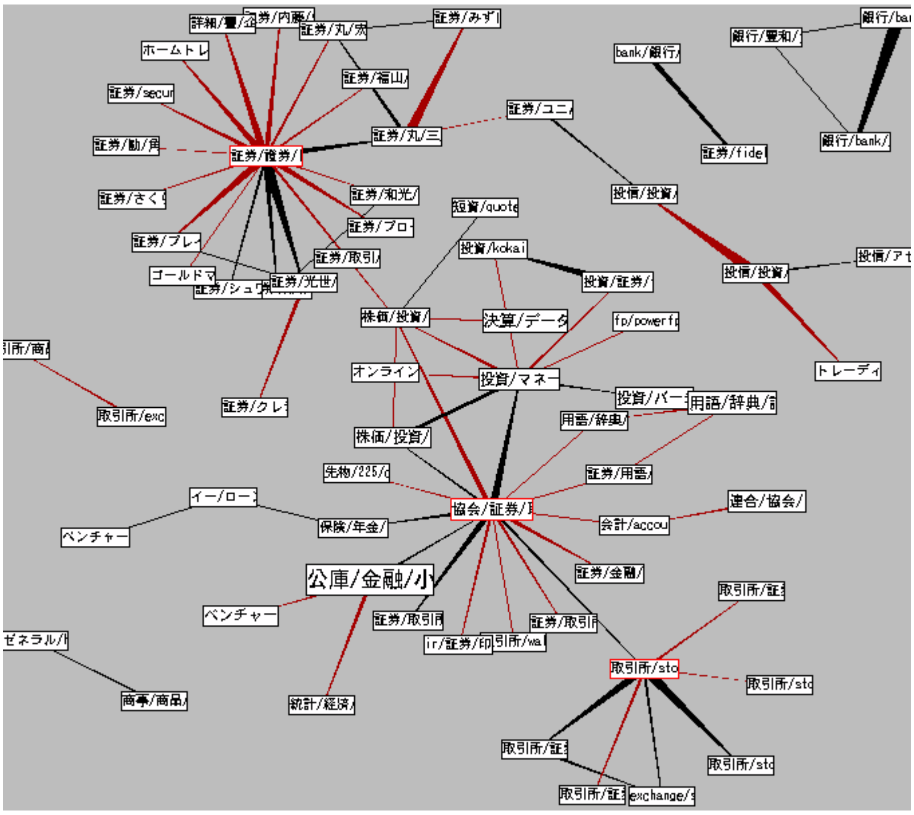


Figure 10.11: A chart of communities related to the stock market.

the degree of interest (**DOI**) for every community, and then displays the labels of communities having a DOI that is higher than a threshold. Currently, the user can point to a community to focus on it and see related communities.

Here, the algorithm of the calculation of DOI is described. First, the browser gives 100 points of DOI to the community v of interest. Then, the DOI of a community (u) that has an edge with v is calculated by the following algorithm.

$$
\begin{aligned}
DOI_u &= DOI_v \frac{w(u,v) + w(v,u)}{d + w(u,v) + w(v,u)} \qquad (10.1)\\
& \quad w(u,v)\text{represents the weight of the link from } u \text{ to } v\\
& \quad d(\text{rate of decrease})
\end{aligned}
$$

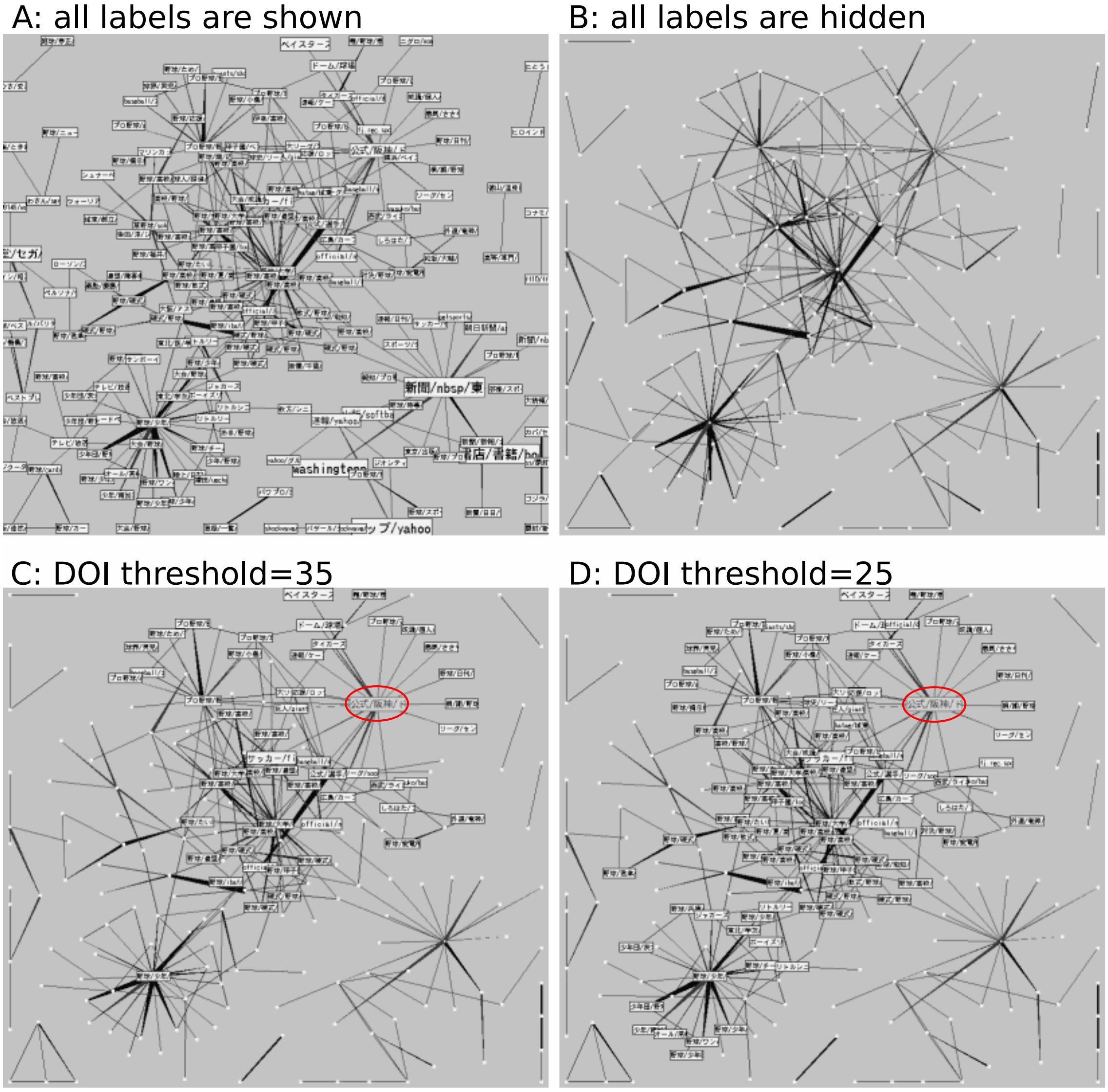
The browser then calculates the related communities of u recursively. If the calculated DOI is lower than the threshold, it stops the traversal calculation.

Figure 10.12 shows an example of a fisheye view of the Web Community Chart that consists of communities that contains the keyword ‘baseball’. Figure 10.12-A shows a chart in which all of the labels are shown, and Figure 10.12-B shows A chart in which all of the labels are hidden. Figures 10.12-C and 10.12-D show the results of fisheye view, where the focus is a community of official baseball teams. The threshold of DOI is set at 35 in Figure 10.12-C and 25 in Figure 10.12-D. By comparing Figures 10.12-A and 10.12-B, the labels are found to occlude the structure of the chart. By Figures 10.12-C and 10.12-D, the user can recognize both the structure of the chart and detailed information of the related communities. The differences in the thresholds can be seen in the lower left-hand corner. There is a small hub community of major children’s baseball clubs, and in Figure 10.12-D there are a number of related communities that appear around this hub, while they are hidden in Figure 10.12-C.

10.3.4 Problems

Currently, the lengths of edges are fixed during runtime, and the length of each edge is calculated from the score of the communities. This calculation of edge length is arbitrary and does not guarantee an ideal graph layout. Moreover, when there are too many edges, the graph tends to appear shrunken. In addition, this layout algorithm does not avoid the crossing of edges; sometimes a graph falls into a stable layout with crossing edges and the user must resolve these crossings manually. In the present study, this manual process to achieve a better layout, is referred to as *disentangling*.

Disentangling is performed by sticking and dragging communities using a mouse. This operation costs time during an analysis.



A: all labels are shown

B: all labels are hidden

C: DOI threshold=35

D: DOI threshold=25

Figure 10.12: Example of fisheye view

In addition, sometimes the user wants to move or rotate an entire sub-graph at once. Of course, special commands could be added to support these requirements, but this requires the addition of an edit mode.

10.3.5 Concurrent manipulation on the Web Community Browser

In order to solve the above problems, concurrent manipulation was introduced to the Web Community Browser. This proposal can be applied to the other graph editing applications.

In this solution, an application was developed that recognizes multipoint input and bulldozer operation on a SmartSkin and transmits these data to a Web Community Browser via UDP. The position of each finger is sent as a pair of 0–1 single-precision floating point numbers and the Web Community Browser scales the position to the size of its graph browsing window. For bulldozer operation, optical flow-based recognition was employed. The result is transmitted as an array of vectors to the browser. OpenSound Control[66] over UDP was used for communication.

A SmartSkin-based input system was used, as shown in Figure 8.4. To use the entire input surface for graph editing, the graph layout window of the Web Community Browser covers the input surface completely, and the other console of the browser is displayed beside the input surface. A mouse was used to manipulate the console.

10.3.6 Graph editing with concurrent manipulation

First, the Web Community Browser was extended to enable concurrent manipulation of communities. The user can move multiple communities simultaneously by pointing to them with his fingers. Each community attached to a finger is not moved by the graph layout optimization process, but rather follows the finger at all times.

Concurrent manipulation of multiple communities allows an intuitive disentangling and rotating manipulation. To disentangle a graph, the user points to two or more communities and stretches them to resolve the crossing of edges or place hub communities separately. Moreover, by rotating attached communities, the entire graph can be rotated, because connected communities are drawn by the tension force of the edges.

These manipulation can be performed by sticking and moving communities repeatedly, but this requires a number of manual operations. In particular, in rotating a graph, because the previous implementation of the browser allows single manipulation of a community, communities must be moved sequentially. However, when the user moves a community to rotate the graph, the community is drawn by the other communities when the community is released, making the rotation very difficult.

10.3.7 Graph editing with bulldozer operation

Next, bulldozer manipulation was enabled to move multiple communities simultaneously. The browser obtains a vector field that represents the motion of the hands on the input surface from the optical flow engine and applies the force from the engine to the communities at the process of graph layout optimization. Hence, the user can move communities of a sub-graph simultaneously by his hands. Previously, in order to move an entire graph, the user had to point to and drag one of the communities on the graph and then wait until the rest of the communities were drawn to that spot. In addition, there was no effective method to move multiple communities that were not linked to each other. In such cases, the communities must be moved sequentially.

By the bulldozer manipulation, multiple communities could be moved simultaneously whether they were linked or not. In particular, this method is useful for brushing off communities to the edge of the window in order to make space to open a new graph, which is useful for comparing two graphs.

10.3.8 Multi focus fisheye view

Since multipoint input allows the user to point to multiple communities, the fisheye view feature was extended to support multi-focuses.

Let $V = \{v_1, v_2, ..., v_n\}$ be a set of focused communities by a user. The browser gives the same initial DOI value to all communities in V, and then calculates DOI of linked communities by Equation 10.1. Let $D(u, v_i)$ be the DOI of community u propagated from community v_i. The DOI of u is calculated from the following equation:

$$D(u) = \max_{i=1}^{n} D(u, v_i) \tag{10.2}$$

This improvement in the multi-focus fisheye view enables users to point to multiple communities and find related communities around them. This facilitates the finding of bridging communities between hub communities.

10.3.9 Discussion

Disentangling and rotating of a graph by multipoint input, or concurrent movement by bulldozer manipulation could be possible by a conventional GUI with a mouse. For example, rotating a graph can be achieved by adding a rotating mode, as in most graphic editors, where the user selects communities by a rectangle and moves one of the corners of the rectangle. In the same way, communities could be moved simultaneously by rectangle selection.

However, this extension, which adds concurrent manipulation by multipoint input, enables rotation and concurrent movement without adding a special editing mode. Note that the extension does not implement a rotation mode or a concurrent movement mode directly. As described in Chapter 3, this implementation enables these operations by allowing concurrent manipulation of multiple components by fingers.

The Web Community Browser was based on a near physical simulation among the components, and the behaviors were calculated in parallel. This feature of the browser is though to have enabled intuitive rotation and concurrent movement of communities by concurrent manipulation, because users could perform common manipulation with objects seen in daily life. This result implies that if a GUI system simulates the physical behavior of components precisely and allows concurrent manipulation of these components, the user will be able to manipulate the components in a highly intuitive manner.

The multi-focus extension for fisheye view highlighted the bridging of communities between focused groups of communities. However, at this time, the browser displays a community only in two styles. Its label is displayed if its DOI exceeds the threshold, or its label is hidden if its DOI does not exceed the threshold. Therefore, it is not possible to see how far a community's DOI is from the threshold. In the single focus fisheye view, the focused community has the highest DOI. In contrast, in the multi-focus fisheye view, a number of bridging communities may have higher DOI than that of the focused communities. In order to emphasize these communities, it is necessary to visualize the community's DOI. Moreover, a different method of DOI propagation should be introduced, instead of Equation 10.2.

10.3.10 Conclusion

Concurrent manipulation was introduced to a Web Community Browser by using SmartSkin. This enabled multiple communities to be moved simultaneously by multipoint input or bulldozer manipulation and rotation of a graph by multipoint input. Disentangling or rotating a graph took time using a mouse, but the concurrent manipulation enabled rapid and intuitive manipulation of graphs without adding a new editing mode.

Simulating the physical behaviors of components in a display is thought to be important for intuitive and effective concurrent manipulation.

Chapter 11

Conclusion

11.1 Summary

In the present research, a generic input system was proposed to allow users to control multiple internal states simultaneously by manipulating multiple components on a screen concurrently. The design goal of the proposed input system is to build a generic, efficient and highly responsive system.

Conventional input systems of common computers allow the user to use a pointing device such as a mouse, and the user manipulates a component on a screen to transmit his intentions to the computer. However, an application usually has multiple internal parameters, and even if the user wants to change them at the same time, the current input system only allows the user to change the parameters one by one via corresponding components.

In the present study, the advantages of concurrent manipulation were analyzed. Applications were found to include multiple internal states and corresponding components. Based on this analysis, three typical situations in which concurrent manipulation performs effectively were introduced. One situation is that the application has multiple independent components that can be manipulated simultaneously. The second situation is that there is a task that includes multiple components that can be manipulated simultaneously. The third situation involves a system that allows collaborative work by multiple users.

Previous studies on concurrent manipulation were reviewed, and the previous approaches were discussed. A few systems were found to allow concurrent manipulation, but had not tested to evaluate differences in implementation.

Therefore, several approaches were developed to implement input systems for concurrent manipulation based upon knowledge of the previous discussion. Two basic approaches were proposed: a system that uses multiple physical devices to input multiple positional data, and a system that accepts direct pointing by fingers. The requirements of these systems were described, and two method were proposed to allow concurrent manipulation without pointing input: a shape-based input by hands and a system using a curve input device.

Three prototypes of concurrent manipulation were implemented in different manners. Evaluation tests were performed on two of the prototypes, and the effectiveness of concurrent manipulation was confirmed. In addition, a system of shape-based input without physical devices was implemented and was confirmed to be usable for concurrent manipulation.

Finally, a number of applications were built on these input systems for real uses. These applications are based on different input systems, for different fields. These applications were used and their effectiveness with respect to concurrent manipulation was confirmed through load tests.

11.2 Contributions

The following is a summary of the contributions of the present research.

- A taxonomy of concurrent manipulation was introduced
- A concurrent manipulation system with physical devices was developed and its effectiveness was confirmed.
- A non-device-input multipoint input system on a human-body sensor was developed.
- The system was evaluated and its effectiveness was confirmed.
- Bulldozer manipulation was proposed for concurrent manipulation without positional input, and bulldozer manipulation with a human-body sensor was developed.
- A number of applications were built on these input systems for real use and load tests were conducted, confirmed their efficiency.

11.3 Limitations and open issues

The limitations of concurrent manipulation and the proposed prototypes were described and a number of open issues were identified.

11.3.1 Large number of components

When there are many components on the screen that must be manipulated independently, although concurrent manipulation performs well, its effectiveness is limited. On a non-device-input system, the number of components a user can manipulate simultaneously is limited to the number of fingers, and an evaluation test indicated that users cannot use all of their fingers efficiently. An input system with physical devices allows concurrent manipulation of as many components as the number of input devices, but this is also limited by the user's skill.

11.3.2 application to existing GUIs

In order to introduce concurrent manipulation to existing GUI frameworks, it is desirable that the cost of installation is low. But existing GUIs seriously depend the presence of only one pointing device. For example, consider what would happen if "Yes" and "No" buttons were pressed simultaneously. It is impossible to enable all components to be manipulated concurrently; otherwise we would have to redesign the GUI framework from scratch.

11.3.3 Concurrent manipulation of independent components

Evaluation tests indicated that it is difficult to manipulate multiple components that are to be moved to different directions, because of the constraints of the locomotive aspects of the human hands. In the conclusion of Chapter 8, concurrent manipulation was proposed to be effective for components that are local and that are to be manipulated in the approximately the same directions, not for completely independent components. However, it is not clear how many components of applications can be parallelized.

11.3.4 Direct or indirect pointing

For intuitive interaction, direct pointing is better than indirect pointing for concurrent manipulation. However, in the evaluation of prototype 2, direct pointing caused an occlusion problem whereby the user's hands covered the input surface. This problem has been reported for pen-based PCs and is considered to be a severe problem. Indirect pointing could be a solution to this problem, but it is difficult to maintain input devices without seeing the input surface. In a previous study, TactaPad (see Section 5.2.6) provided a solution to this problem by overlaying the image of the hands onto the computer screen, but interference with the screen should be minimized.

11.4 Future work

The proposed concurrent manipulation system will be extended to various applications and their efficiencies will be evaluated precisely. If each application is required to be overhauled in order to apply concurrent manipulation, application programmers will not adopt this system.

Therefore, the following two frameworks will be investigated for various applications.

11.4.1 Extension for GUI toolkit

The GUI toolkit provides a set of GUI components and event handling procedures to develop GUI applications. Existing GUI toolkits will be extended for concurrent manipulation.

Naively, this can be achieved by allowing multiple events from multipoint input system for every component, but this causes the problem described in the section on limitations. Therefore, an additional attribute that enables or disables concurrent manipulation will be added. A tree of GUI components would be useful for managing the attributes.

11.4.2 Application to a visual data flow language

Concurrent manipulation will be applied to a controllable component of the visual data flow language introduced in Section 2.2.5. In this framework, every component virtually runs independently and accepts multiple signals at any time. This aspect is suited for concurrent manipulation.

Recently, this type of visual language is used for a rapid GUI builder so that it is able to apply concurrent manipulation to various applications using the visual language. In particular, visual languages are used by applications for art, which will solve the problems of applications used in the creation of art (see Section 2.4).

Bibliography

[1] Ravin Balakrishnan, George Fitzmaurice, Gordon Kurtenbach, and William Buxton. Digital Tape Drawing. In *Proceedings of UIST'99*, pages 161–169, 1999.

[2] Ravin Balakrishnan, George W. Fitzmaurice, Gordon Kurtenbach, and Karah Singh. Exploring interactive curve and surface manipulation using a bend and twist sensitive input strip. In *Proceedings of I3D 1999 – the ACM Symposium on Interactive 3D Graphics*, pages 111–118, 1999.

[3] Patrick Baudisch. Don't Click, Paint!: Using Toggle Maps to Manipulate Sets of Toggle Switches. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 65–66, 1998.

[4] Matthew Brand, Nuria Oliver, and Alex Pentland. Coupled hidden Markov models for complex action recognition. Technical Report 407, M.I.T. Media Laboratory Perceptual Computing / Learning and Common Sense, 1996.

[5] William Buxton and Brad Myers. A STUDY IN TWO-HANDED INPUT. In *Proceedings of CHI'86*, pages 321–326, 1986.

[6] XEROX Palo Alto Research Center. Alto user's handbook, 1979.

[7] James Davis and Xing Chen. LumiPoint: Multi-User Laser-Based Interaction on Large Tiled Displays. Technical report, Stanford University, 2001.

[8] Paul H. Dietz and Darren Leigh. DiamondTouch: A Multi-User Touch Technology. In *Proceedings of UIST'01*, pages 219–226, 2001.

[9] Douglas Engelbart. US Patent # 3,541,541: Computer mouse.

[10] Douglas C. Engelbart and William K. English. A Research Center for Augmenting Human Intellect. In *Proceedings of the 1968 Fall Joint Computer Conference*, pages 395–410, 1968.

[11] George Fitzmaurice and William Buxton. An Empirical Evaluation of Graspable User Interfaces: towards specialized, space-multiplexed input. In *Proceedings of CHI'97*, pages 43–50, 1997.

[12] George Fitzmaurice, Hiroshi Ishii, and William Buxton. Bricks: Laying the Foundations for Graspable User Interfaces. In *Proceedings of CHI'95*, pages 442–449, 1995.

[13] George W. Fitzmaurice. *Graspable User Interfaces*. PhD thesis, Dept. of Computer Science, University of Toronto, 1996.

[14] Marty Frenzel, Joe Marks, and Kathy Ryall. First UIST interface-design contest. *Interactions*, 9(5):57–62, 2002.

[15] Kentaro Fukuchi, Sam Mertens, and Ed Tannenbaum. EffecTV: A Real-Time Software Video Effect Processor for Entertainment. In Matthias Rauterberg, editor, *Entertainment Computing - ICEC2004*, volume 3166 of *Lecture Notes in Computer Science*, pages 602–605. Springer, 2004.

[16] Kentaro Fukuchi and Ed Tannenbaum. EffecTV: an application of DEMO techniques to realtime video effect. In *Proceedings of Entertainment Computing 2003*, pages 94–99, 2003.

[17] Kentaro Fukuchi, Masashi Toyoda, and Masaru Kitsuregawa. Web Community Browser: visualization of a large Web community chart. In *Proceedings of DEWS2002*, 2002.

[18] George W. Furnas. Generalized Fisheye Views. In *Proceedings of CHI'86*, pages 16–23, 1986.

[19] Gimp - The GNU Image Manipulation Program. `http://www.gimp.org/`.

[20] Saul Greenberg and Chester Fitchett. Phidgets: Easy Development of Physical Interfaces through Physical Widgets. In *Proceedings of UIST'01*, pages 209–218, 2001.

[21] Tyson R. Henry and Scott E. Hudson. Interactive Graph Layout. In *Proceedings of UIST'91*, pages 55–64, 1991.

[22] Ken Hinckley, Randy Pausch, John C. Goble, and Neal F. Kassell. Passive Real-World Interface Props for Neurosurgical Visualization. In *Proceeding of CHI'94*, pages 452–458, 1994.

[23] Ken Hinckley, Randy Pausch, Dennis Proffitt, James Patten, and Neal Kassell. Cooperative Bimanual Action. In *Proceedings of CHI'97*, pages 27–34, 1997.

[24] Takeo Iagarashi, Tomer Moscovich, and John F. Hughes. As-Rigid-As-Possible Shape Manipulation. In *ACM Transactions on Computer Graphics, Vol.24, No.3, ACM SIGGRAPH 2005*, pages 1134–1141, 2005.

[25] iGesturePad. `http://www.fingerworks.com/igesture.html`.

[26] T. Kamada and S. Kawai. An algorithm for drawing general indirect graphs. In *Information Processing Letters*, number 31, pages 7–15, 1989.

[27] Hirokazu Kato and Mark Billinghurst. Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System. In *Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99)*, 1999.

[28] Carsten Kirstein and Heinrich Müller. Interaction with a Projection Screen Using a Camera-tracked Laser Pointer. In *Proceedings of The International Conference on Multimedia Modeling*, pages 191–192, 1998.

[29] Hideki Koike, Xinlei Chen, Yasuto Nakanishi, Kenji Oka, and Yoichi Sato. Two-handed drawing on augmented desk. In *Proceedings of CHI2002*, pages 760–761, 2002.

[30] KORG KAOSS PAD. `http://www.korg.co.jp/Product/Dance/KPE1/`.

[31] Nobuyuki Matsushita, Yuji Ayatsuka, and Jun Rekimoto. Dual Touch: a New Interaction Technique for Pen-Based PDAs. In Michiaki Yasumura, editor, *Interactive Systems and Software VII*, pages 23–32. Kindai Kagakusya, 1999.

[32] Nobuyuki Matsushita and Jun Rekimoto. HoloWall: Designing a Finger, Hand, Body, and Object Sensitive Wall. In *Proceedings of UIST'97*, pages 209–210, 1997.

[33] Measurand Inc. ShapeTape. `http://www.measureand.com/`.

[34] Kobayashi Motoki and Koike Hideki. Enhanced Desk, Integrating Paper Documents and Digital Documents. In *Proceedings of APCHI'98*, pages 167–174, 1998.

[35] Brad A. Myers, Rishi Bhatnagar, Jeffrey Nichols, Choon Hong Peck, Dave Kong, Robert Miller, and A. Chris Long. Interacting at a Distance: Measuring the Performance of Laser Pointers and Other Devices. In *Proceeding of CHI2002*, pages 33–40, 2002.

[36] Satoshi Nakamura, Masahiko Tsukamoto, and Shojiro Nishio. Design and Implementation of Operating Mechanisms for Window System with Two Mice. *IPSJ SIG Notes*, 99(35):1–6, 1999.

[37] Satoshi Nakamura, Masahiko Tsukamoto, and Shojiro Nishio. Design and Implementation of the Double Mouse System for a Window Environment. In *Proceedings of Pacific Rim Conference on Communications, Computers and Signal Processing(PACRIM 2001)*, pages 204–207, 2001.

[38] Michael Nielsen, Moritz Störring, Thomas B. Moeslund, and Erik Granum. A Procedure for Developing Intuitive and Ergonomic Gesture Interfaces for HCI. In *Gesture-Based Communication in Human-Computer Interaction: 5th International GestureWorkshop*, volume 2915/2004 of *Lecture Notes in Computer Science*, pages 409–420. Springer Verlag, 2004.

[39] Ji Young Oh and Wolfgang Stuerzlinger. Laser Pointers as Collaborative Pointing Devices. In *Proceedings of Graphics Interface 2002*, pages 141–149, 2002.

[40] Kenji Oka, Yoichi Sato, and Hideki Koike. Real-time fingertip tracking and gesture recognition. *IEEE Computer Graphics and Applications*, 22(6):64–71, November/December 2002.

[41] Dan R. Olsen Jr. and Travis Nielsen. Laser Pointer Interaction. In *Proceedings of CHI2001*, pages 17–22, 2001.

[42] François Pachet and Olivier Delerue. MidiSpace: a Temporal Constraint-Based Music Spatializer. In *Proceedings of the ACM Multimedia Conference*, pages 351–359, 1998.

[43] Gian Pangaro, Dan Maynes-Aminzade, and Hiroshi Ishii. The Actuated Workbench: Computer-Controlled Actuation in Tabletop Tangible Interfaces. In *Proceedings of UIST'02*, pages 181–190, 2002.

[44] James Patten, Hiroshi Ishii, Jim Hines, and Pangaro Gian. Sensetable: A Wireless Object Tracking Platform for Tangible User Interfaces. In *Proceedings of CHI2001*, pages 253–260, 2001.

[45] James Patten, Ben Recht, and Hiroshi Ishii. Audiopad: A Tag-based Interface for Musical Performance. In *Proceedings of the 2002 International Conference on New Interfacefor Musical Expression (NIME02)*, pages 11–16, 2003.

[46] Philippe P. Piernot, Marcos R. Vescovi, Justin Willow, and Robin Petravoc. US Patent # 6,417,663: Detecting physical objects states using electromagnetic sensors, 2002.

[47] Polhemus. `http://www.polhemus.com/`.

[48] Miller Puckette. The Patcher. In *Proceedings of the 1988 International Computer Music Conference*, pages 420–429, 1988.

[49] Miller Puckette. Pure Data. In *Proceedings of the 1996 International Computer Music Conference*, pages 269–272, 1996.

[50] Jun Rekimoto. Matrix: A Realtime Object Identification and Registration Method for Augmented Reality. In *Proceedings of APCHI'98*, pages 63–68, 1998.

[51] Jun Rekimoto. SmartSkin: An Infrastructure for Freehand Manipulation on Interactive Surfaces. In *Proceedings of CHI2002*, pages 113–120, 2002.

[52] Thad Starner and Alex Pentland. Real-Time American Sign Language Recognition from Video Using Hidden Markov Models. Technical Report 375, M.I.T. Media Laboratory Perceptual Computing Section, 1995.

[53] Rahul Sukthankar, Robert G. Stockton, and Matthew D. Mullin. Self-Calibrating Camera-Assisted Presentation Interface. In *Proceedings of International Conference on Automation, Control, Robotics and Computer Vision, 2000*, pages 33–40, 2000.

[54] Ivan E. Sutherland. Skethpad: A man-machine graphical communication system. In *Proceedings of the AFIPS Spring Joint Computer Conference*, pages 329–346, May 1963.

[55] Tactiva TactaPad. `http://tactiva.com/`.

[56] The Second ACM UIST Interface-Design Contest. `http://www.acm.org/uist/uist2002/contest/`.

[57] Masashi Toyoda and Masaru Kitsuregawa. Creating a Web Community Chart for Navigating Related Communities. In *Conference Proceedings of Hypertext 2001*, pages 103–112, 2001.

[58] Brygg Ullmer and Hiroshi Ishii. The metaDESK: Models and Prototypes for Tangible User Interfaces. In *Proceedings of UIST'97*, pages 223–232, 1997.

[59] John Underkoffler and Hiroshi Ishii. Illuminating Light: An Optical Design Tool with a Luminous-Tangible Interface. In *Proceedings of CHI'98*, pages 542–549, 1997.

[60] John Underkoffler, Brygg Ullmer, and Hiroshi Ishii. Emancipated Pixels: Real-World Graphics In The Luminous Room. In *Proceedings of SIGGRAPH'99*, pages 385–392, 1999.

[61] Andries van Dam. Post-WIMP user interfaces. *Communications of the ACM*, 40(2):63–67, February 1997.

[62] Christian Vogler and Dimitris Metaxas. Adapting Hidden Markov Models for ASL Recognition by Using Three-dimensional Computer Vision Methods. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 156–161, 1997.

[63] Andrew Wilson and Aaron Bobick. Nonlinear Parametric Hidden Markov Models. Technical Report 424, M.I.T. Media Laboratory Perceptual Computing Section, 1998.

[64] Andrew Wilson and Aaron Bobick. Recognition and Interpretation of Parametric Gesture. Technical Report 421, M.I.T. Media Laboratory Perceptual Computing Section, 1998.

[65] Valerie Woods, Sarah Hastings, Peter Buckle, and Roger Haslam. *Ergonomics of using a mouse or other non-keyboard input device*. HSE Books, 2002.

[66] Matthew Wright and Adrian Freed. OpenSound Control: A New Protocol for Communicating with Sound Synthesizers. In *Proceedings of International Computer Music Conference '97*, 1997.

[67] Christian Wurster. *COMPUTERS: an illustrated history*. Taschen, 2002.

[68] Chen Xinlei, Kenji Oka, Yasuto Nakanishi, Yoichi Sato, and Hideki Koike. A Two-Handed Drawing Tool on Augmented Desk System. In *Proceedings of WISS2001*, pages 179–184. Kindai Kagakusya, 2001.